

Athman Bouguettaya  
Ingolf Krueger  
Tiziana Margaria (Eds.)

LNCS 5364

# Service-Oriented Computing – ICSOC 2008

6th International Conference  
Sydney, Australia, December 2008  
Proceedings

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Athman Bouguettaya Ingolf Krueger  
Tiziana Margaria (Eds.)

# Service-Oriented Computing – ICSOC 2008

6th International Conference  
Sydney, Australia, December 1-5, 2008  
Proceedings

Volume Editors

Athman Bouguettaya  
CSIRO ICT Centre, GPO Box 664  
Canberra, ACT 2601 Australia  
E-mail: athman.bouguettaya@csiro.au

Ingolf Krueger  
University of California, San Diego  
Dept. of Computer Science and Engineering, CSE Building  
9500 Gilman Drive, La Jolla, CA 92093-0404, USA  
E-mail: ikrueger@cs.ucsd.edu

Tiziana Margaria  
University of Potsdam, Institute for Informatics  
August-Bebel-Str. 89, 14482, Potsdam, Germany  
E-mail: margaria@cs.uni-potsdam.de

Library of Congress Control Number: 2008939869

CR Subject Classification (1998): C.2, D.2, D.4, H.4, H.3, K.4.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743  
ISBN-10 3-540-89647-3 Springer Berlin Heidelberg New York  
ISBN-13 978-3-540-89647-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper SPIN: 12577101 06/3180 5 4 3 2 1 0

# Preface

This volume contains the conference proceedings of ICSOC 2008, the 6th International Conference on Service-Oriented Computing, which took place in Sydney, Australia, December 1–5, 2008; it comprises research, industry and demo papers. ICSOC 2008 built upon the tradition of five previous successful editions that were held in Vienna, Austria (2007), Chicago, USA (2006), Amsterdam, The Netherlands (2005), New York City, USA (2004) and Trento, Italy (2003). ICSOC is recognized as the premier conference for service-oriented computing research; it covers the entire spectrum from theoretical and foundational results to empirical evaluations to practical and industrial experiences. In addition, ICSOC 2008 has successfully demonstrated the cross-disciplinary nature of service engineering by building bridges with the business community, and by attracting contributions on service-oriented systems of systems integration.

Service-oriented computing (SOC) has emerged as an approach to tackling the complexity we face in developing, operating and maintaining Internet-scale applications of high quality. SOC shifts the focus from monolithic systems to flexible integration of services using novel approaches to dynamic discovery, orchestration, assembly and management, policy and governance, quality of service, and information assurance. SOC is also a key enabler of the emerging trends towards grid and cloud computing, which play an integral role in enabling novel applications in E-Sciences, E-Government and ultra-large-scale software-intensive systems, to name just a few examples. This shift toward flexible service integration also requires a deep understanding and, often, rethinking of end-to-end systems engineering processes, including the corresponding business and economic drivers for definition of, or changes to, enterprise architectures.

The program we assembled is reflective of the breadth and depth of the research and applications of SOC, with contributions in the following areas:

- Business Service Modeling
- System of Systems Integration
- Service Engineering
- Service Assembly and Grid Services
- Service Management
- SOA Runtime
- Quality of Service
- Business and Economical Aspects of Services

In addition, we were honored to have four prominent players in the area of SOC give keynote addresses at ICSOC 2008:

- “Services for Science”, by Ian Foster, Argonne National Laboratory and University of Chicago.
- “Web Scale Computing: The Power of Infrastructure as a Service”, by Peter Vosshall, Vice-President and Distinguished Engineer at Amazon.com.

- “Services in the Long tail World: Challenges and Opportunities”, by Neel Sundaresan, Senior Director and Head at Ebay Research Labs.
- “Managing and Internet Service Bus”, by Donald F. Ferguson, Chief Architect, Enterprise IT Management Products at CA, Inc.

ICSOC 2008 received 151 contributions in the research track, of which we accepted 32 full and 20 short papers. The Industry Track selected 6 of the 22 submissions, and the Demonstration Committee proposed 6 Demonstration papers out of 11 submissions for inclusion in the proceedings. The selection process was difficult because of the large number of excellent submissions we had to choose from.

We thank the Area Coordinators for their help throughout the review process, as well as the members of the Program Committee and their sub-referees for their efforts in selecting the papers to be presented.

We also gratefully acknowledge the contributions of Martin Karusseit, Holger Willebrandt, and Zoi Choselidou in helping with the conference management system (OCS) and the notification process. We would also like to acknowledge Hakim Hacid for his untiring and thankless work of maintaining the ICSOC 2008 website. We would also like to acknowledge the active and generous support of our sponsors. To all of them, and all others who helped make ICSOC 2008 a success, we express our gratitude!

We hope you find the papers in this volume interesting and stimulating.

December 2008

Athman Bouguettaya  
Tiziana Margaria  
Ingolf Krueger

# Organization

## General Chairs

Boualem Benatallah	University of New South Wales, Australia
Vincenzo d' Andrea	University of Trento, Italy
Frank Leymann	University of Stuttgart, Germany

## Program Committee Chairs

Athman Bouguettaya	CSIRO, Australia
Ingolf H. Krueger	University of California, San Diego, USA
Tiziana Margaria	University of Potsdam, Germany

## Area Coordinators

### Service Foundations

Bernhard Steffen	Technical University Dortmund, Germany
Gianluigi Zavattaro	University of Bologna, Italy

### Business Service Modeling

Jian Yang	Macquarie University, Australia
-----------	---------------------------------

### Integrating Systems of Systems Using Services

Doug Schmidt	Vanderbilt University, USA
--------------	----------------------------

### Service Engineering

Michael Huhns	University of South Carolina, USA
---------------	-----------------------------------

### Service Assembly

Paco Curbera	IBM
--------------	-----

### Service Management

Asit Dan	IBM Research, USA
Mike Papazoglou	University of Tilburg, The Netherlands

### SOA Runtime

Priya Narasimhan	Carnegie Mellon University, USA
------------------	---------------------------------

### **Quality of Service**

Mourad Ouzzani                      Purdue University, USA

### **Grid Services**

Domenico Laforenza                CNR, Italy  
Uwe Schwiegelshohn                TU Dortmund, Germany

### **Business and Economical Aspects of Services**

Paul Maglio                            IBM Almaden, USA  
Stefan Tai                                TU of Karlsruhe, Germany

### **Program Committee**

Wil van der Aalst                      TU Eindhoven, The Netherlands  
Marco Aiello                            University of Groningen, The Netherlands  
Jörn Altmann                            Seoul National University, South Korea  
Luciano Baresi                        Politecnico di Milano, Italy  
Alistair Barros                        SAP, Australia  
Salima Benbernou                    University of Lyon, France  
Kamal Bhattacharya                  IBM Research, USA  
Ken Birman                              Cornell University, USA  
Laura Bocchi                            University of Leicester, UK  
Mario Bravetti                         University of Bologna, Italy  
Karin Breitman                        PUC Rio, Rio de Janeiro, Brazil  
Ruth Breu                                University of Innsbruck, Austria  
Frank van Breugel                    York University, Ontario, Canada  
Paul Buhler                              College of Charleston, Charleston, USA  
Tefvik Bultan                          UC Santa Barbara, California, USA  
Christoph Bussler                    BEA, USA  
Hong Cai                                 IBM China Research Laboratory, China  
Jorge Cardoso                        SAP, Germany  
Manuel Carro                          Polytechnic University of Madrid, Spain  
Fabio Casati                            ITC-IRST, Italy  
Shiping Chen                          CSIRO, Australia  
Siobhan Clarke                        Trinity College, Dublin, Ireland  
Jiangbo Dang                         Siemens, USA  
Asuman Dogac                        METU, Turkey  
Schahram Dustdar                    TU Vienna, Austria  
Kim Elms                                SAP, Germany  
Chris Gill                                Washington University, St. Louis, Missouri, USA  
Andy Gordon                          Microsoft Research, UK  
Paul Grefen                            Eindhoven University of Technology, The Netherlands  
Andrew Grimshaw                    University of Virginia, USA  
Norbert Gronau                        University Potsdam, Germany



Chihab Hanachi	University of Toulouse, France
Jingshan Huang	University of South Carolina, USA
Richard Hull	Bell Labs, USA
Dimka Karastoyanova	University Stuttgart, Germany
Bettina Kemme	McGill University, Canada
Bernd Kraemer	Free University Hagen, Germany
Christine Legner	University of St. Gallen, Switzerland
Qianhui (Althea) Liang	Singapore Management University, Singapore
Chengfei Liu	Swinburne University, Australia
Qing Liu	CSIRO, Australia
Michael Maximilien	IBM Almaden, USA
Massimo Mecella	Università di Roma “La Sapienza”, Italy
Brahim Medjahed	University of Michigan, USA
Anne Ngu	Texas State University, USA
Christos Nikolaou	University of Crete, Greece
Cesare Pautasso	University Lugano, Switzerland
Thierry Priol	INRIA, France
Wolfgang Reisig	Humboldt University Berlin, Germany
Abdelmounaam Rezgui	Virginia Tech, USA
Colette Rolland	University Paris 1, France
Robin Russell	Virginia Tech, USA
Karsten Schwan	Georgia tech, USA
Quan Z. Sheng	University of Adelaide, Australia
Munindar Singh	NCSU, USA
Ketil Stoelen	SINTEF, Norway
Jianwen Su	UC Santa Barbara, California, USA
Tarja Systä	Tampere University, Finland
Zahir Tari	RMIT, Australia
Paolo Traverso	ITC-IRST, Italy
Kunal Verma	Accenture Technology Labs, USA
Von Welch	NCSA, UIUC, USA
Mathias Weske	University of Potsdam, Germany
John Wilkes	HP, USA
Raymond Wong	University of NSW, Australia
Lai Xu	CSIRO, Australia
Yelena Yesha	University of Maryland, USA
Qi Yu	Rochester Institute of Technology, USA
Wenbing Zhao	Cleveland State University, USA
Andrea Zisman	City University London, UK

## Reviewers

Sudhir Agarwal  
 Berthold Agraeter  
 Gunes Aluc

Vasilios Andrikopoulos  
 Samuil Angelov  
 Claudio Bartolini

Domenico Bianculli  
Martin Bichler  
Aliaksandr Birukou  
Marina Bitsaki  
Benjamin Blau  
Joanna Chimiak-Opoka  
Heidi Dahl  
Florian Daniel  
Elie El-Khoury  
Rik Eshuis  
Michael Felderer  
Alexander Foelling  
Jorge Fox  
Stefan Freitag  
G.R. Gangadharan  
Christian Grimme  
Alexander Grosskopf  
Will Jan van den Heuvel  
Yi Hong  
Tormod Håvaldsrud  
Frank Innerhofer-Oberperfler  
Yildiray Kabak  
Eirini Kaldeli  
Mariana Karmazi  
Basel Katt  
Natallia Kokash  
Woralak Kongdenfha  
Gokce Banu Laleci  
Steffen Lamparter  
Joachim Lepping  
Olav Ligaarden  
Sarah Löw  
Zaki Malik

Michele Mancioffi  
Waqar Mashwani  
Mukhtiar Memon  
Harald Meyer  
Hamid Motahari  
Tuncay Namli  
Franco Maria Nardini  
Anh Nguyen-Tuong  
Aida Omerovic  
Alexander Papaspyrou  
Pantelis Petridis  
Rodion Podorozhny  
Alina Psycharaki  
Hajo Reijers  
Sebastien Saudrais  
Lars Schley  
Fabrizio Silvestri  
Haiyang Sun  
Evi Syukur  
Aries T. Tao  
Gabriele Tolomei  
Nicola Tonello  
Mazhar Tekin  
Fulya Tuncer  
Jochem Vonk  
Paul de Vrieze  
Matthias Weidlich  
Barbara Weber  
Yong Yang  
Hong Qing Yu  
Jian Yu  
Xiaohui Zhao  
Christian Zirpins

## Industry Program Chairs

Christoph Bussler	Merced Systems, Inc., USA
Don Ferguson	Computer Associates, USA
Volkmar Lotz	SAP, USA

## Industry Committee

Jorge Cardoso	SAP, Germany
Malu Castellanos	HP Labs, USA
Richard Hull	Bell Labs, USA

Mark Little	RedHat, USA
Heiko Ludwig	IBM Research, USA
Eugene M. Maximilien	IBM Almaden, USA
Kunal Verma	Accenture, USA
Sanjeeva Weerawarana	WSO2, USA
Umit Yalcinalp	SAP, USA
Michal Zaremba	SeekDa, Austria

## **Demonstration Chairs**

Malu Castellanos	HP Labs, USA
Marlon Dumas	University of Tartu, Estonia
Karsten Schulz	SAP, Australia

## **Demonstration Committee**

Jorge Cardoso	SAP Research, Germany
Mike Carey	BEA Systems, USA
Remco Dijkman	Eindhoven University of Technology, The Netherlands
Schahram Dustdar	Vienna University of Technology, Austria
Howard Foster	Imperial College, UK
Rania Khalaf	IBM TJ Watson Research Centre, USA
Peep Küngas	SOA Trader, Estonia
Julien Vayssiere	SAP Research, Australia
Jim Webber	ThoughtWorks, UK
Andreas Wombacher	EPFL, Switzerland
Olaf Zimmermann	IBM Zürich Research Laboratory, Switzerland

# Table of Contents

Web Scale Computing: The Power of Infrastructure as a Service . . . . .	1
<i>Peter Voss</i>	
Services in the Long Tail World: Challenges and Opportunities . . . . .	2
<i>Neel Sundaresan</i>	
Services for Science . . . . .	3
<i>Ian Foster</i>	
Managing and Internet Service Bus . . . . .	4
<i>Donald F. Ferguson</i>	
Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems . . . . .	5
<i>Tan Phan, Jun Han, Jean-Guy Schneider, and Kirk Wilson</i>	
Adaptation of Web Service Composition Based on Workflow Patterns . . .	22
<i>Qiang He, Jun Yan, Hai Jin, and Yun Yang</i>	
Protocol-Based Web Service Composition . . . . .	38
<i>Ramy Ragab Hassen, Lhouari Nourine, and Farouk Toumani</i>	
Design and Implementation of a Fault Tolerant Job Flow Manager Using Job Flow Patterns and Recovery Policies . . . . .	54
<i>Selim Kalayci, Onyeka Ezenwoye, Balaji Viswanathan, Gargi Dasgupta, S. Masoud Sadjadi, and Liana Fong</i>	
Building Mashups for the Enterprise with SABRE . . . . .	70
<i>Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab</i>	
Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques . . . . .	84
<i>Radu Mateescu, Pascal Poizat, and Gwen Salaün</i>	
Automatic Workflow Graph Refactoring and Completion . . . . .	100
<i>Jussi Vanhatalo, Hagen Völzer, Frank Leymann, and Simon Moser</i>	
Authorization and User Failure Resiliency for WS-BPEL Business Processes . . . . .	116
<i>Federica Paci, Rodolfo Ferrini, Yuqing Sun, and Elisa Bertino</i>	
Reasoning on Semantically Annotated Processes . . . . .	132
<i>Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Paolo Tonella</i>	

Event-Driven Quality of Service Prediction . . . . .	147
<i>Liangzhao Zeng, Christoph Lingenfelder, Hui Lei, and Henry Chang</i>	
Automatic Realization of SOA Deployment Patterns in Distributed Environments . . . . .	162
<i>William Arnold, Tamar Eilam, Michael Kalantar, Alexander V. Konstantinou, and Alexander A. Totok</i>	
The LLAMA Middleware Support for Accountable Service-Oriented Architecture . . . . .	180
<i>Mark Panahi, Kwei-Jay Lin, Yue Zhang, Soo-Ho Chang, Jing Zhang, and Leonardo Varela</i>	
<i>ubi</i> SOAP: A Service Oriented Middleware for Seamless Networking . . . .	195
<i>Mauro Caporuscio, Pierre-Guillaume Raverdy, Hassine Moun gla, and Valerie Issarny</i>	
Towards a Service-Oriented Approach for Managing Context in Mobile Environment . . . . .	210
<i>Waskitho Wibisono, Arkady Zaslavsky, and Sea Ling</i>	
An Autonomic Middleware Solution for Coordinating Multiple QoS Controls . . . . .	225
<i>Yan Liu, Min'an Tan, Ian Gorton, and Andrew John Clayphan</i>	
Transparent Runtime Adaptability for BPEL Processes . . . . .	241
<i>Adina Mosincat and Walter Binder</i>	
Organizational Constraints to Realizing Business Value from Service Oriented Architectures: An Empirical Study of Financial Service Institutions . . . . .	256
<i>Haresh Luthria and Fethi Rabhi</i>	
E-Marketplace for Semantic Web Services . . . . .	271
<i>Witold Abramowicz, Konstanty Haniewicz, Monika Kaczmarek, and Dominik Zyskowski</i>	
Business Driven SOA Customization . . . . .	286
<i>Pietro Mazzoleni and Biplav Srivastava</i>	
Sound Multi-party Business Protocols for Service Networks . . . . .	302
<i>Michele Mancioffi, Manuel Carro, Willem-Jan van den Heuvel, and Mike P. Papazoglou</i>	
Automatic Mash Up of Composite Applications . . . . .	317
<i>Michael Pierre Carlson, Anne H.H. Ngu, Rodion Podorozhny, and Liangzhao Zeng</i>	

Non-desynchronizable Service Choreographies . . . . .	331
<i>Gero Decker, Alistair Barros, Frank Michael Kraft, and Niels Lohmann</i>	
A Framework for Semantic Sensor Network Services . . . . .	347
<i>Lily Li and Kerry Taylor</i>	
Context-Driven Autonomic Adaptation of SLA . . . . .	362
<i>Caroline Herssens, Stéphane Faulkner, and Ivan J. Jureta</i>	
Determining QoS of WS-BPEL Compositions . . . . .	378
<i>Debdoot Mukherjee, Pankaj Jalote, and Mangala Gowri Nanda</i>	
An Initial Approach to Explaining SLA Inconsistencies . . . . .	394
<i>Carlos Müller, Antonio Ruiz-Cortés, and Manuel Resinas</i>	
Ontology-Based Compatibility Checking for Web Service Configuration Management . . . . .	407
<i>Qianhui Liang and Michael N. Huhns</i>	
SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services . . . . .	422
<i>Ignacio Silva-Lepe, Revathi Subramanian, Isabelle Rouvellou, Thomas Mikalsen, Judah Diamant, and Arun Iyengar</i>	
WorldTravel: A Testbed for Service-Oriented Applications . . . . .	438
<i>Peter Budny, Srihari Govindharaj, and Karsten Schwan</i>	
TCP-Compose* – A TCP-Net Based Algorithm for Efficient Composition of Web Services Using Qualitative Preferences . . . . .	453
<i>Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar</i>	
A Runtime Quality Architecture for Service-Oriented Systems . . . . .	468
<i>Daniel Robinson and Gerald Kotonya</i>	
QoS Policies for Business Processes in Service Oriented Architectures . . . . .	483
<i>Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux</i>	
Deriving Business Service Interfaces in Windows Workflow from UMM Transactions . . . . .	498
<i>Marco Zapletal</i>	
From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL . . . . .	505
<i>Man Zhang and Zhenhua Duan</i>	
Batch Invocation of Web Services in BPEL Process . . . . .	511
<i>Liang Bao, Ping Chen, Xiang Zhang, Sheng Chen, Shengming Hu, and Yang Yang</i>	

Formation of Service Value Networks for Decentralized Service Provisioning . . . . .	517
<i>Sebastian Speiser, Benjamin Blau, Steffen Lamparter, and Stefan Tai</i>	
Towards Automated WSDL-Based Testing of Web Services . . . . .	524
<i>Cesare Bartolini, Antonia Bertolino, Eda Marchetti, and Andrea Polini</i>	
Automated Service Composition with Adaptive Planning . . . . .	530
<i>Sandrine Beauche and Pascal Poizat</i>	
A Planning-Based Approach for the Automated Configuration of the Enterprise Service Bus . . . . .	538
<i>Zhen Liu, Anand Ranganathan, and Anton Riabov</i>	
Verifying Interaction Protocol Compliance of Service Orchestrations . . . .	545
<i>Andreas Schroeder and Philip Mayer</i>	
Specify Once Test Everywhere: Analyzing Invariants to Augment Service Descriptions for Automated Test Generation . . . . .	551
<i>Amit Paradkar and Avik Sinha</i>	
A Model-Driven Approach to Dynamic and Adaptive Service Brokering Using Modes . . . . .	558
<i>Howard Foster, Arun Mukhija, David S. Rosenblum, and Sebastian Uchitel</i>	
Integrated Security Context Management of Web Components and Services in Federated Identity Environments . . . . .	565
<i>Apurva Kumar</i>	
Predicting and Learning Executability of Composite Web Services . . . . .	572
<i>Masahiro Tanaka and Toru Ishida</i>	
Authorization Policy Based Business Collaboration Reliability Verification . . . . .	579
<i>Haiyang Sun, Xin Wang, Jian Yang, and Yanchun Zhang</i>	
VGC: Generating Valid Global Communication Models of Composite Services Using Temporal Reasoning . . . . .	585
<i>Nalaka Gooneratne, Zahir Tari, and James Harland</i>	
A Framework for Advanced Modularization and Data Flow in Workflow Systems . . . . .	592
<i>Niels Joncheere, Dirk Deridder, Ragnhild Van Der Straeten, and Viviane Jonckers</i>	
Model Identification for Energy-Aware Management of Web Service Systems . . . . .	599
<i>Mara Tanelli, Danilo Ardagna, Marco Lovera, and Li Zhang</i>	

LASS – License Aware Service Selection: Methodology and Framework . . . . .	607
<i>G.R. Gangadharan, Marco Comerio, Hong-Linh Truong, Vincenzo D’Andrea, Flavio De Paoli, and Schahram Dustdar</i>	
Integrated and Composable Supervision of BPEL Processes . . . . .	614
<i>Luciano Baresi, Sam Guinea, and Liliana Pasquale</i>	
Optimised Semantic Reasoning for Pervasive Service Discovery . . . . .	620
<i>Luke Steller and Shonali Krishnaswamy</i>	
COSMA – An Approach for Managing SLAs in Composite Services . . . .	626
<i>André Ludwig and Bogdan Franczyk</i>	
Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications . . . . .	633
<i>Thomas Kwok and Ajay Mohindra</i>	
SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment . . . . .	649
<i>Xin Hui Li, Tian Cheng Liu, Ying Li, and Ying Chen</i>	
Management as a Service for IT Service Management . . . . .	664
<i>Bo Yang, Hao Wang, and Ying Chen</i>	
SMART: Application of a Method for Migration of Legacy Systems to SOA Environments . . . . .	678
<i>Sriram Balasubramaniam, Grace A. Lewis, Ed Morris, Soumya Simanta, and Dennis Smith</i>	
Discovering and Deriving Service Variants from Business Process Specifications . . . . .	691
<i>Karthikeyan Ponnalagu and Nanjangud C. Narendra</i>	
Market Overview of Enterprise Mashup Tools . . . . .	708
<i>Volker Hoyer and Marco Fischer</i>	
Siena: From PowerPoint to Web App in 5 Minutes . . . . .	722
<i>David Cohn, Pankaj Dhoolia, Fenno Heath III, Florian Pinel, and John Vergo</i>	
Exploration of Discovered Process Views in Process Spaceship . . . . .	724
<i>Hamid R. Motahari Nezhad, Boualem Benatalah, Fabio Casati, Regis Saint-Paul, Periklis Andristos, and Adnene Guabtni</i>	
ROME4EU: A Web Service-Based Process-Aware System for Smart Devices . . . . .	726
<i>Daniele Battista, Massimiliano de Leoni, Alessio De Gaetanis, Massimo Mecella, Alessandro Pezzullo, Alessandro Russo, and Costantino Saponaro</i>	



WS-Engineer 2008: A Service Architecture, Behaviour and Deployment Verification Platform .....	728
<i>Howard Foster</i>	
MetaCDN: Harnessing Storage Clouds for High Performance Content Delivery .....	730
<i>James Broberg and Zahir Tari</i>	
Yowie: Information Extraction in a Service Enabled World .....	732
<i>Marek Kowalkiewicz and Konrad Jünemann</i>	
<b>Author Index</b> .....	735

# Web Scale Computing: The Power of Infrastructure as a Service

Peter Vosshall

Amazon VP and Distinguished Engineer  
vosshall@amazon.com

**Abstract.** Building the right infrastructure that can scale up or down at a moment's notice can be a complicated and expensive task, but it's essential in today's competitive landscape. This applies to an enterprise trying to cut costs, a young business unexpectedly saturated with customer demand, or a research lab wanting to test at scale. There are many challenges when building a reliable, flexible architecture that can manage unpredictable behaviors of today's Internet business. This presentation will outline some of the lessons learned from building one of the world's largest distributed systems, Amazon.com, and the evolution that gave rise to Amazon reselling its infrastructure in the form of Amazon Web Services, allowing anyone to leverage the same robust, scalable, and reliable technology that powers Amazon's business.

# Services in the Long Tail World: Challenges and Opportunities

Neel Sundaresan

Sr. Director and Head, eBay Research Labs  
nsundaresan@ebay.com

**Abstract.** This talk will focus on Internet based systems that are primarily participatory in nature. In such systems, we need to think beyond infrastructure, data, and algorithms. While these entities are well understood from the service architecture point of view, the demands of participatory systems are different. In a massive-scale system like eBay that is highly participatory in nature, user roles, actions and interactions affect and influence how the system functions and scales. While applications and platforms as service are well understood in the current, evolution through participation mandates the need for additional service orientations. For instance, *interface as a service* through programmable implementations or *user experience as a service* through programmable visual elements and interactions can be easily perceived. Machines and machine algorithms will take us part of the way but making them scalable and adaptable to change is a challenge. We need to talk about augmented intelligence where machine power coexists with and is complemented by human intelligence. Designing scalable services and applications in this dynamic context pose interesting challenges and new opportunities. This talk will focus on the unique nature of this long tail world.

# Services for Science

Ian Foster

Computation Institute  
Argonne National Laboratory  
& University of Chicago  
foster@mcs.anl.gov

**Abstract.** Computational approaches to problem solving have proven their worth in many fields of science, allowing the collection and analysis of unprecedented quantities of data and the exploration via simulation of previously obscure phenomena. We now face the challenge of scaling the impact of these approaches from the specialist to entire communities. I speak here about work that seeks to address this goal by rethinking science’s information technology foundations in terms of service-oriented architecture. In principle, service-oriented approaches can have a transformative effect on scientific communities, allowing tools formerly accessible only to the specialist to be made available to all, and permitting previously manual data-processing and analysis tasks to be automated. However, while the potential of such “service-oriented science” has been demonstrated, its routine application across many disciplines raises challenging technical problems. One important requirement is to achieve a separation of concerns between discipline-specific content and domain-independent infrastructure, so that new services can be developed quickly and existing services can respond effectively to time-varying load. Another key requirement is to streamline the formation and evolution of the “virtual organizations” that create and access content. I describe the architectural principles, software, and deployments that I am and my colleagues have produced as we tackle these problems, and point to future technical challenges and scientific opportunities. I illustrate my talk with examples from astronomy and biomedicine.

# Managing and Internet Service Bus

Donald F. Ferguson

Chief Architect, Enterprise IT Management Products  
CA, Inc.

[donald.ferguson@ca.com](mailto:donald.ferguson@ca.com)

**Abstract.** SOA and Web services have profoundly changed enterprise and commercial applications. BPEL, dynamic binding via service registries and repositories, alignment of grid computing with Web service standards, and a common approach to SOA and event driven architectures are examples of technologies that enable a new approach to applications and solutions. Many papers and talks have explained these technologies and their benefits. Systems and application management using Web services is a growing area that builds on these technologies. There are many benefits to a common SOA/Web service approach to modeling, developing, deploying, managing and optimizing SW solutions. This presentation explains the benefits.

Several significant intellectual challenges hinder realizing the promise of a SOA/Web service approach to systems and application management. One of the most important is “managing from a business service perspective.” Business professionals have a completely different definition of “service” from technical professionals. Enterprises think in terms of IT realization of “business services,” for example online banking or shipped package tracking. The business services are an interacting fabric of SOA services, and in many cases the enterprise does not fully understand which services interact in a business solution or to process a request. Many elements in the business service are not SOA services, for example databases, directories, file servers, etc.

This talk provides a deeper explanation of the business problem and challenges. The talk also explains the state of the art for solving some of the challenges. Finally, the talk concludes with suggestions for research and projects.

# Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems

Tan Phan<sup>1</sup>, Jun Han<sup>1</sup>, Jean-Guy Schneider<sup>1</sup>, and Kirk Wilson<sup>2</sup>

<sup>1</sup> Faculty of Information & Communication Technologies  
Swinburne University of Technology  
P.O. Box 218 Hawthorn, VIC 3122, Australia  
{tphan, jhan, jschneider}@swin.edu.au

<sup>2</sup> CA Labs  
One CA Plaza, Islandia, NY 11749, USA  
{Kirk.Wilson@ca.com}

**Abstract.** Enterprise software systems play an essential role in an organization's business operation. Many business rules and regulations governing an organization's operation can be translated into quality requirements of the relevant software systems, such as security, availability, and manageability. For systems implemented using *Web Services*, the specification and management of these qualities in the form of *Web Service policies* are often complicated and difficult to be aligned with the initial business requirements. In this paper, we introduce the HOPE (High-Level Objective-based Policy for Enterprises) framework that supports, in a systematic manner, the specification of quality-oriented policies at the business level and their refinement into policies at the system/service level. Quality-oriented business requirements are expressed in HOPE as quality objectives applied to business entities and further refined or translated into system-level WS-Policy statements. The refinement relies on an application-specific business entity model and application-independent domain quality models. We demonstrate the approach with a case study involving policy specification and refinement in the security domain.

## 1 Introduction

Business rules, government acts such as Sarbanes-Oxley [1], industry standards such as Basel II [2], and enterprise-specific rules mandate non-functional or quality requirements of the various entities in an organization's IT environment. These requirements can often be formulated as high-level quality objectives (*e.g.*, *Customer data must be kept confidential*) and realized using various means for IT management and governance.

In recent years, Service-Oriented Architectures (SOA) and Web Services (WS) have offered a new way of implementing enterprise business processes. Core business functionalities are codified as network-accessible Web Services and enterprise software systems become live networks of interconnected services. To ensure that WS-based SOA systems are reliable and interoperable, various industry standards have been proposed to support the specification and management of quality aspects, most notably security, reliable messaging, and transactions [3]. In general, these standards are about system-level mechanisms to achieve some non-functional qualities. Example mechanisms in

security or distributed transaction coordination are role-based access control, message encryption and signing, and content-based routing. The *Web Services Policy Framework* (WS-Policy) [4] is a standard that supports the specification of various quality properties for Web Services and service systems.

One of the issues that needs to be addressed is how to *align the high-level and often business-oriented quality objectives with the system-level realization mechanisms* offered by the WS standards. Currently, the quality objectives are often identified by practitioners who are either business analysts or IT compliance officers (hereafter referred to as *policy experts*). They have a good understanding of the business domain and regulations, and have a high-level understanding of the IT systems in general. However, policy experts are typically not SOA experts and often do not have an in-depth understanding of all the system-level realization mechanisms used to achieve the quality objectives. They view IT systems more from a business perspective and their concerns are to identify the quality objectives rather than how they can be realized. It is the system developers' responsibility to implement the quality objectives in the corresponding IT systems. The underlying processes are generally ad-hoc and, therefore, it is difficult to ensure that a system fully implements all required quality objectives. As such, a contribution of great value would be a systematic process and related techniques that can derive the system-level realization from the business-level requirements and can verify that the realization actually fulfills these requirements.

In this work, we address the issue of aligning business-oriented quality objectives with system-level WS quality properties by introducing the HOPE (High-level Objective-based Policies for Enterprises) framework. HOPE assists policy experts in specifying business policies and quality objectives and, by utilizing realization mechanisms available in the respective *quality domains*, refines them into system-level WS-Policy statements that prescribe quality properties for service-based enterprise software systems. This paper starts with a business case study as a motivating example. It then introduces the HOPE framework and the mechanisms for policy refinement, illustrated using selected examples from the case study. A prototyping tool for HOPE is also presented. The paper concludes with a summary of the main contributions and future work.

## 2 A Motivating Example

In this section, we introduce an example business process and identify applicable rules and regulations. A set of quality-oriented business policies is then derived from the rules and regulations. We discuss the limitations of current approaches with regards to specifying and realizing such policies, motivating our approach of the HOPE framework.

### 2.1 Business Case: The Mortgage Loan Approval Business Process

When a customer applies for a mortgage loan product at a hypothetical multi-national bank SwinBank, a `LoanOfficier` accepts the application and triggers the bank's loan approval business process. First, the bank arranges a professional appraiser to estimate the market value of the collateral property. Next, the customer's ID/social security number is forwarded to a credit checking unit to verify the customer's credit history. A list

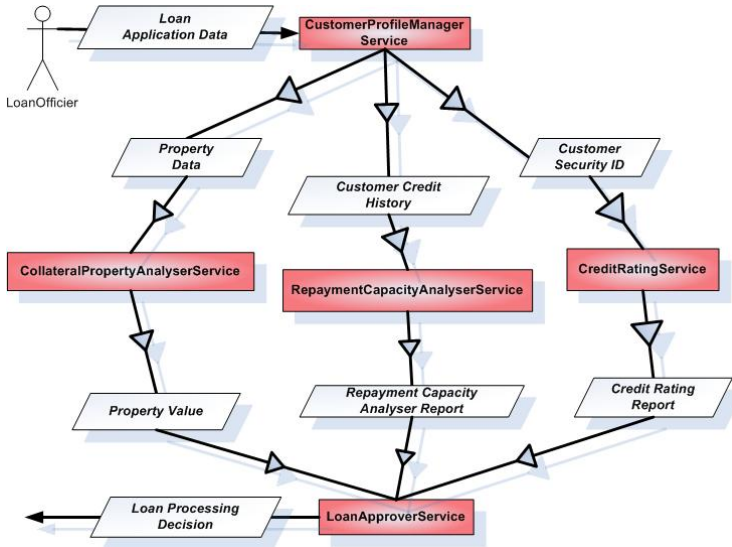


Fig. 1. The Mortgage Loan Approval business process

of credit scores from multiple credit rating agencies is then obtained. Finally, the repayment capacity of the customer is checked by judging the income against the amount to be repaid. Based on this information, an approval decision for the loan is made.

## 2.2 MortgageLoan: The Mortgage Loan Approval Application

SwinBank automates its mortgage loan approval process using SOA. The process is implemented in BPEL utilizing a number of services (depicted in Figure 1).

The service `CustomerProfileManager` provides customer account information whereas `CollateralPropertyAnalyser` calculates the value of a given property. The `CreditRating` service, acting as a gateway to other credit rating agencies' WS, forwards the customer's social security number to the agencies and obtains the customer's credit history report. The service `RepaymentCapacityAnalyser` checks the customer's repayment capacity based on his/her income and the amount of loan to be repaid, taking into account interest rate, inflation, and other factors. Finally, the service `LoanApprover` takes the output of the previous three services and makes a decision, with human input, as to whether the loan is approved or not.

## 2.3 Rules, Regulations and SwinBank Business Policies

The discussion of the mortgage loan approval business process and system for SwinBank has primarily focused on the business and application functionality. In reality, this business process is also subject to many rules and regulations that may be general or specific to the Banking Industry. The following are some of the relevant acts:



1. **Bank Secrecy Act of 1970** [5]: Any information disclosed by the applicants, any temporary data collected during the approval process, and any final decision need to be persistently stored and traceable.
2. **Australian Privacy Act 1988** [6]: Information related to loan applicants' credit information, customer identifier information must be available to only authorized personnel and not disclosed to the public.

By analyzing the rules and regulations, policy and compliance experts can identify the *business policies* applicable to the loan approval process and system. In general, business policies can be functional or non-functional (*i.e.* system's qualities). In this paper, we focus on the latter. The following are some of the non-functional quality-oriented business policies for the loan approval process and system:

**(BP1):** *Loan application data must be persistently stored.*

**(BP2):** *Information about customers' personal identifications and financial records must be kept secure during transmission.*

**(BP3):** *All activities related to loan processing must be recorded.*

**(BP4):** *Loan applicants must not be able to repudiate the lodgment of a loan and the bank must not be able to repudiate the receipt of a loan application.*

**(BP5):** *People working with customer information must be authorized.*

These business policies require `MortgageLoan` to have the corresponding quality properties related to the privacy and security of data and loan processing activities.

## 2.4 Realization of Quality-Oriented Business Policies

In SOA development, system requirements (functional and non-functional) are generally realized through services and the WS policies applicable to these services (at the *assembly or deployment* phase) [7]. The fact that some system requirements are realized through policies increases the flexibility and agility of the system. For example, WS policies can be updated to realize certain system changes without modifying the services' implementation. As the WS-Policy framework is predicated on WS interactions, only requirements that concern WS interactions can be realized through Web Service policies. Other requirements have to be realized in the services themselves. In this paper, we focus on quality-oriented non-functional requirements or *business policies*, and in particular those that can be realized through WS policies at the system-level (cf. Figure 2). For example, the requirement in **BP1** cannot be fully realized by the WS quality model and, therefore, needs to be realized programmatically or using other means (such as database transaction management).

In current SOA practice (cf. Figure 2(a)), it is the system developers' responsibility to realize the quality-oriented business policies in terms of WS policies. This process is generally ad-hoc and there is no easy way to ensure that all the relevant business-level policies are properly interpreted and implemented by the developers, who have generally limited understanding of the rules and regulations [7]. Furthermore, this process is often very tedious as well as error-prone.

To alleviate this problem, we have developed the HOPE framework with the aim to *automate* the process of refining high-level business policies into system-level quality

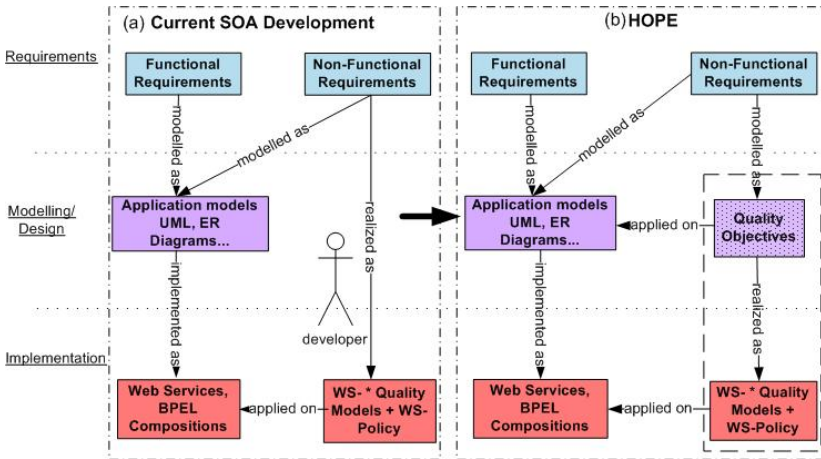


Fig. 2. Current SOA development (a) and the HOPE approach (b)

properties (cf. Figure 2). In particular, we introduced the concept of *quality objectives* to model non-functional business requirements in the form of required qualities applicable to the various entities in a system’s design models. These quality objectives are then refined into WS policies applicable to the service-based system.

### 3 The HOPE Framework

HOPE is a framework for specifying high-level, quality-oriented business policies and refining them into system-level Web Services policies for Web Services-based enterprise software systems in a systematic manner. HOPE is built on a number of underlying models as illustrated in Figure 3.

The *quality models* are domain-specific and identify, for a given domain (e.g., security), the relevant quality attributes and the mechanisms to realize them. The *application entity model* provides a layered hierarchy of business-oriented entities involved in the application, and can be extracted from the application’s design models. The business concepts in this entity model are used to specify the *quality objectives* for the system and to annotate the system’s WS elements (*portTypes*, *operations*, *messages* etc.). Based on the quality models and the WS annotations, HOPE refines and translates the high-level *quality objectives* into system-level policy statements asserting WS properties. The remainder of this section discusses these models in more detail.

#### 3.1 Domain Quality Models

For each quality domain a *quality model* defines the set of *quality attributes* concerning the domain and the *realization mechanisms* for these attributes. The quality model itself is application-independent and is often derived from *standards*, *ontologies*, *patterns*, and *best practices*, respectively, and the system-level WS quality specification WS-\*, where \* denotes the *quality domain name* for that domain. We have chosen

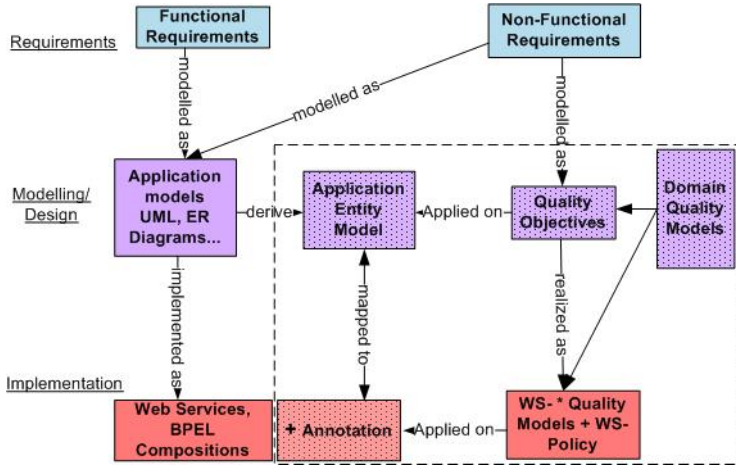


Fig. 3. The HOPE framework

XML Schema as a means to specify quality models and defined a corresponding *quality meta-model* each quality model must adhere to. In the following, we will further illustrate this approach using the security domain as an example.

**The Quality Meta-Model and the Security Quality Model.** The quality meta-model provides a formal structure to specify the entities of each quality domain. Throughout the remainder of this section, we will use a simplified security quality model based on [7,8,9,10,11] for illustration purposes (cf. Figure 4). We expect that other domain quality models can be expressed using the same set of notations and structure as defined in the security meta-model.

- **Quality.** a quality attribute or quality,  $q$ , specifies a desired quality aspect. In the security domain, the common quality attributes are *confidentiality* (preventing unauthorized access to sensitive data), *integrity* (preventing unauthorized modification of the data), *non-repudiation* (preventing a message sender from repudiating the fact that it was him who sent the message and a message receiver from repudiating the fact that it was him who received the message), *authentication* (proving the authenticity of a user), *authorization* (proving that the user is in the role he claims), and *audit* (making sure that actions are recorded and traceable).
- **Quality realization mechanism.** a quality realization mechanism defines how a quality  $q$  can be realized. It is a logical structure in *disjunctive normal form* (OR of ANDs) of *abstract quality functions*. For example, according to [12,13], (i) *confidentiality* can be realized by *encrypting* data that needs to be protected, (ii) *integrity* can be realized by *signing* the data, and (iii) *non-repudiation* of a message can be realized by *logging* the sending and receiving actions and *signing* the message.
- **Quality function.** a quality function specifies a measure that can be used to achieve one or more *qualities*. A quality function  $f$  is specified in the form  $f(fb)$  where  $fb$  denotes the *function binding* (defined below) for that function. If  $fb$  is left empty, the function is called an *abstract function*, only specifying *what* needs to be done.

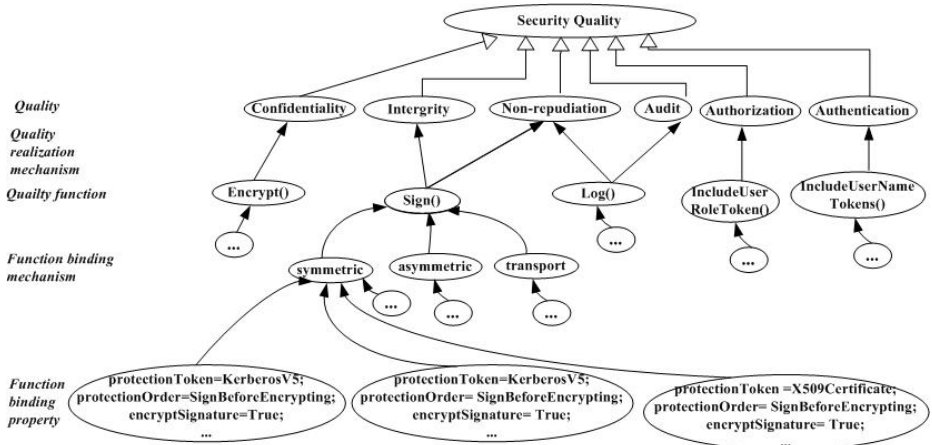


Fig. 4. The partial security meta-model

If  $fb$  is defined, the function is called a *concrete function*. For example, some of the common abstract security functions as defined in [12][13] are *encrypt*, *sign*, *log*, *includeUsernameToken*, and *includeUserRoleToken*.

- **Function Binding Mechanism 1.** a function binding mechanism represents a method of realizing a quality function based on a type of quality infrastructure. A quality domain typically has a limited set of alternatives for realizing a given quality function. For example, security functions such as *encrypt* or *sign* have the three binding mechanisms (i) *transport* (using transport security such as HTTPS), (ii) *symmetric* (using a shared key), and (iii) *asymmetric* (using a pair of public and private keys).
- **Binding Property Set.** a binding property set  $\{(p_1, v_1), \dots, (p_n, v_n)\}$  contains name-value pairs where  $p_i$  is the name of a property and  $v_i$  the corresponding value. Example security binding properties are `AlgorithmSuite=SHA1`, indicating that the SHA1 algorithm suite is used, and `SignatureProtection=TRUE`, indicating that both the signature and the signature confirmation elements must be encrypted.
- **Function Binding Tree.** a function binding tree is formed by detailing abstract functions with *binding information*. The root of a binding tree is the *abstract function* itself containing empty binding information. The direct children of the root node contain the function's *binding mechanisms*. Child-nodes of the binding mechanism nodes are leaf nodes containing all possible values of available binding properties for that mechanism. Furthermore, function binding trees can be assigned priorities for particular binding mechanisms and/or properties. For the security function *encrypt*, the binding tree is formed by having a root node being the abstract *encrypt* function, the direct children of the root nodes are the binding mechanisms *transport*, *symmetric*, and *asymmetric*, and the sub-nodes contain detailed binding properties for each of the binding mechanism (e.g., `Algorithm=SHA1`).

Binding trees can potentially become quite large as the number of possible branches is defined as the Cartesian product of all available *binding mechanisms* and all applicable *binding property* values. However, an organization often follows a certain

security profile which has a limited number of predefined binding options. For example, the *Basic Security Profile*, Version 1.0 [14] mandates the use of message level mechanisms and places some constraints on values of certain binding properties (e.g., *SHA1-based algorithms must be used for interoperability purposes*).

**System-Level Web Services Quality Models.** At the system-level, qualities are applied to Web Services in the form of *WS-Policy statements*. The WS Policy framework (WS-Policy) allows the specification of *qualities* and their realization details for WS. For each quality domain, there is a WS-\*Policy standard, such as the WS-SecurityPolicy [15] for security, that allows for the specification of the qualities and their realization details in that domain. WS-Policy is extensible and specifications for new quality domains can be defined and included in the framework. It is aimed at defining non-functional properties that govern service-service or service-client interactions, but not the implementation details of the services themselves. However, there are qualities that cannot be supported by the WS quality model, e.g. *durability* for persisting data. Therefore, when defining a quality model, the system-level WS quality model for that domain is taken into consideration in order to filter out the qualities, functions, mechanisms or properties that are not supported by the system-level model.

### 3.2 The Application Entity Model

An application's *entity model* defines application-specific business concepts. Policy experts work with this model and apply policies on the entities in the model in the form of quality objective requirements. In general, such a model is extracted from an application's analysis and design models (e.g., ER or UML diagrams) made available by business analysts or system architects during system analysis. In HOPE, a *Business-Entity* represents a business-oriented concept from the application and can be classified into one of the following basic entity types:

- *Processor*: performs business logic at request (e.g., `LoanProcessor`),
- *DataItem*: holds business data (e.g., `CustomerTaxFileNumber`), and
- *UserRole*: represents user roles in an organization, has access to *DataItems*, and can ask *Processors* to perform actions (e.g., `LoanOfficier`).

In a HOPE entity model, each entity is a direct or indirect specialization of one of the three basic entity types. Although an entity in a given application can be the specialization of more than (super-)entity, it can only be the (direct or indirect) specialization of *one* of the basic entity types. For example, an entity cannot be a specialization of both, *DataItem* and *Processor*, respectively. Using this approach, applications can be viewed as compositions of interacting entities.

A HOPE entity model can be represented as a directed graph of entities: a node corresponds to an entity and a (directed) edge represents an entity specialization. Figure 5 represents part of the entity model for the motivating example `MortgageEntityModel` introduced in Section 2. The entities `CustomerData` and `PersonalIdentifier` and their specializations `LoanApplicationData`, `LoanApplicantCreditHistory`, and `LoanApplicantTaxFileNumber` are specializations of the basic entity type *DataItem*, `LoanProcessor` and `CreditVerifier` are specializations of *Processor*, and finally `LoanOfficier` and `Teller` are specializations of the basic entity type *UserRole*.

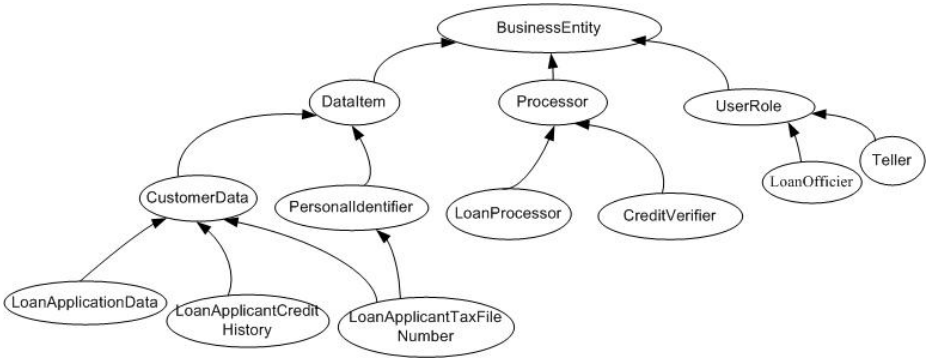


Fig. 5. Example MortgageEntityModel

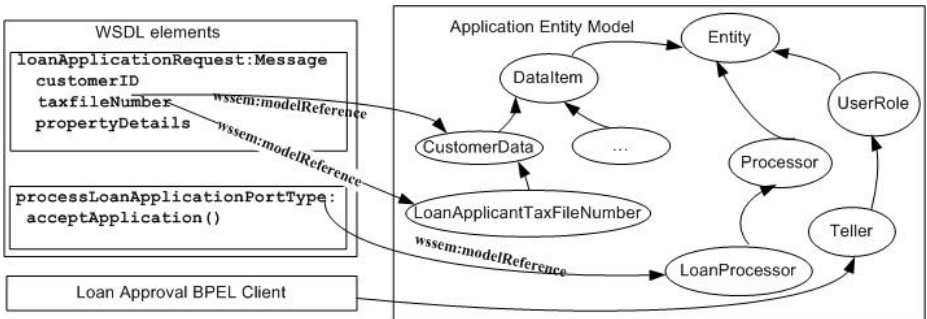


Fig. 6. Example Web Services and entity model mapping using WSDL-S

**Annotating Services and Messages.** Once the high-level entity model is defined, the Web Services elements *portTypes*, *operations*, and *messages* of an application’s implementation need to be mapped to the high-level business concepts defined in the model in order to perform policy refinement. The mapping is done via annotations using the WS Semantic (WSDL-S) framework (chosen for its simplicity and tool support) [16].

Figure 6 shows an example of mapping Web Service elements to concepts of the application entity model using WSDL-S for MortgageLoan. The WSDL-S annotations `wssem:modelReference` are used to map Web Service *messages* such as `loanApplicationRequest` and *message parts* such as `customerID` and `TaxFileNumber` (represented in the WSDL description of the service) to specializations of *DataItem*. In a similar manner, *portTypes* such as `processLoanApplicationPortType` and *operations* such as `acceptApplication` are mapped to the corresponding specializations of *Processor*. Any service client that uses these services is also annotated with *UserRole* information, indicating which user role constraints this client has to adhere to.

### 3.3 Quality Objectives and Policies

A central concept in HOPE is the *quality objective*. A quality objective, denoted by  $q[e]$ , specifies the application of the quality  $q$  on the business entity type  $e$ , meaning that

**Table 1.** Business policies and corresponding quality objectives

Business Policy	Quality Objectives
<b>BP2</b>	(“Information about customers’ . . . secure during transmission”, confidentiality[LoanApplicantTaxFileNumber], confidentiality[LoanApplicantCreditHistory], integrity[LoanApplicantTaxFileNumber], integrity[LoanApplicantCreditHistory])
<b>BP3</b>	(“All activities related to loan processing must be recorded”, audit[LoanApplicationData], audit[LoanApplicantCreditHistory], audit[LoanApplicantCreditResult])
<b>BP4</b>	(“Loan applicants must not be able . . . of a loan application”, non-repudiation[LoanApplicationData])
<b>BP5</b>	(“People working with customer . . . must be authorized”, authorization[LoanOfficier])

quality  $q$  must hold on all entities of type of  $e$  and all of its specializations. Furthermore, a policy is defined as a  $n$ -tuple  $p(t, q_1[e]_1, \dots, q_{n-1}[e_{n-1}])$  where  $t$  is the textual representation in natural language of the business policy requirement,  $q_i[e_i]$  is a quality objective and  $\{q_1[e]_1, \dots, q_{n-1}[e_{n-1}]\}$  is the set of quality objectives meeting the requirements specified by the policy. This information is made available to the policy experts when they apply a quality onto an entity.

Some qualities can only be applied to certain types of entities. For example, *manageability* qualities like *notifiability*, *controllability*, or *introspectability* can only be applied to specializations of *Processor*. In the security domain, *confidentiality*, *integrity*, and *non-repudiation* can only be applied to specializations of *DataItems* whilst *authentication* and *authorization* are only applicable to specializations of *UserRoles*.

The example business policies specified in Section 2 can be decomposed into *quality objectives* as given in Table 1. As mentioned before, **BP1** cannot be supported by HOPE. Apart from that, quality objectives of the other four policies can be refined into WS-SecurityPolicy assertions, based on the security quality model and the annotated Web Service descriptions (cf. Section 4).

## 4 Generating WS-Policy Assertions

The refinement of a quality objective into WS-Policy Assertions involves two major steps: (i) the quality objective is realised in terms of *concrete quality functions*, according to the quality model and (ii) the *concrete quality functions* are mapped to corresponding WS-Policy statements for that domain. The statements are applied on the relevant WS that will be manifested when the services operate at runtime. In this process, mapping information available in WS annotations is used to identify the relevant WS elements that correspond to the business entities in the original quality objectives.

Each quality domain has its own way of generating WS-\*Policy statements from the domain’s *concrete functions*. We will discuss the methods for generating WS-Security-Policy assertions from concrete security functions throughout the rest of this section.

The current version of the WS-SecurityPolicy [15] specification defines different types of assertions for specifying the mechanisms of applying security measures on SOAP messages. There are basically three types of assertion relevant to our mechanism: (i) *protection assertions*, (ii) *token assertions*, and (iii) *binding assertions*.

#### 4.1 Mapping an Abstract Functions to WS-Security Assertions

**Security Functions for *DataItems*:** The WS-SecurityPolicy *protection assertions*, specifying what security measures need to be applied on which parts of SOAP messages, can be used to describe security functions for *DataItems* such as *encrypt* or *sign*. For a security function  $function_X$  to be applied on the entity  $DataItem_X$  which, via annotation, is known to be carried by a collection of  $\langle Message_1, \dots, Message_N \rangle$ , the corresponding WS-SecurityPolicy protection assertion for the function  $function_X$  is

```
<functionXAssertion>
  <Xpath>Message1 </Xpath>
  ...
  <Xpath>MessageN </Xpath>
</functionXAssertion>
```

where  $\langle Xpath \rangle Message_i \langle /Xpath \rangle$  is the path pointing to the message or message part relative to the SOAP document, and the mapping between  $function_X$  and its assertion is as follows:

Quality	Realization Functions	WS-SecurityPolicy assertions
Integrity	<i>encrypt</i>	SignedElements
Confidentiality	<i>sign</i>	EncryptedElements
Non-repudiation	<i>encrypt</i> AND <i>log</i>	EncryptedElements //Log assertion has not been defined

**Security function for Processors and UserRoles:** The focus of WS-Security and, therefore, WS-SecurityPolicy, is not to protect *UserRoles* and *Processors*. However, existing mechanisms can be leveraged to support the realization of *authentication* and *authorization* by using *token assertions* as follows:

Quality	Realization Functions	WS-SecurityPolicy assertions
Authentication	<i>IncludeUsernameToken()</i> : Attach a username token in messages originated from the user.	<wsse:SecurityToken wsp:Usage="wsp:Required"> <wsse:TokenType> wsse:UsernameToken </wsse:TokenType> </wsse:SecurityToken>
Authorization	<i>includeToken – SAML</i> . Attach a SAML token messages originated from the user.	<wsse:SecurityToken wsp:Usage="wsp:Required"> <wsse:TokenType> wsse:SAMLToken </wsse:TokenType> </wsse:SecurityToken>



To associate these assertions with the WS *portTypes* and operations, we use the mechanisms specified in WS-PolicyAttachment [17]. The security functions applied on a *UserRole* are, via entity mapping information, also applied on WS, WS *portTypes*, or WS *operations* that the role might have access to (*i.e.* invoke) accordingly using a similar mechanism. For example, *LoanOfficier*, via annotation is known to have access to the *portType* *LoanProcessor*, thus quality objectives such as *authorization*[*LoanOfficier*] are translated into corresponding WS-Policy assertions that are, via WS-PolicyAttachment, applied on the *LoanProcessor portType*.

The reader may note that the WS-SecurityPolicy [15] standard does not define assertions for all well known security functions as WS-Security itself focuses more on message *confidentiality* and *integrity*. For example, an assertion for *logging* is not available. However, the standard is still evolving and it is expected that additional support will be accommodated in future versions.

## 4.2 Mapping Function Binding to WS-SecurityPolicy Binding Assertions

The general structure of a WS-SecurityPolicy binding assertion is as follows

```
<BindingMechanism>
  <Structured collection of Binding property assertions>
</BindingMechanism>
```

The *BindingMechanism* can be *symmetric*, *asymmetric* or *transport* binding. The “structured collection of Binding property assertions” is generally a logical AND of the assertions that specify the value of the binding properties. In WS-Policy syntax, this logical AND can be represented using a `wsp:Policy` or a `wsp:All` container. For a concrete security function, the function *BindingMechanism* in the quality function is mapped to the corresponding WS-SecurityPolicy *BindingMechanism* assertions and each of the *BindingProperty* is mapped to the corresponding WS-Policy *BindingProperty* assertions. The mapping is relatively straightforward and is one to one.

## 4.3 An Example Assertion Generation

Figure 7 shows the generated WS-SecurityPolicy fragment for the quality objective *integrity*[*LoanApplicantTaxFileNumber*] in the business policy **BP2**. This quality objective can be realized using the security function *sign* to sign the Web Services message parts related to *LoanApplicantTaxFileNumber*. *sign* is mapped to the WS-SecurityPolicy protection assertion *signedElements* (Line 17). We use the *entity mapping* information as in the example introduced in Section 3.2 to map the message part *taxFileNumber* of the message *processLoanRequest* to the business concept *LoanApplicantTaxFileNumber* (Line 18). We assume that *SymmetricBinding* is used (Line 2, 16) and that the preferred *binding properties* are as follows:

```
{protectionToken = KerberosV5ApReqToken11 (line 4-12),
 ProtectionOrder = SignBeforeEncrypting (line 13),
 EncryptSignature = True (line 14)}
```

meaning that a *KerberosV5ApReqToken11* is used as the protection token, the digital signature to be computed over plain text (before the message content is encrypted), and that the signature itself should also be signed, respectively.

```

1<sp:Policy>
2  <sp:>SymmetricBinding>
3    <wsp:Policy>
4      <sp:ProtectionToken>
5        <wsp:Policy>
6          <sp:Kerberos.../>
7          <wsp:Policy>
8            <sp:>WSSKerberosV5ApReqToken11/>
9            <wsp:Policy>
10           </sp:Kerberos>
11          </wsp:Policy>
12        </sp:ProtectionToken>
13        <sp:>SignBeforeEncrypting/>
14        <sp:>EncryptSignature/>
15      </wsp:Policy>
16    </sp:>SymmetricBinding>
17    <sp:>SignedElements...>
18      <sp:XPath>processLoanRequest/taxFileNumber</sp:XPath>
19    </sp:>SignedElements>
20</sp:Policy>

```

**Fig. 7.** WS-SecurityPolicy fragment for *integrity*[LoanApplicantTaxFileNumber]

## 5 Prototype Tool

We have implemented a supporting prototype for the HOPE framework. The prototype assumes the existence of domain quality models and application entity models. The tool also needs the WSDL descriptions of Web Services and service compositions of a given application. As illustrated in Figure 8, the prototype allows users to specify and manage business policies, and have them refined into WS-Policy statements. For policy editing, the tool presents the application entities and the qualities for each of the three domains (*security*, *manageability*, and *reliability*) in tabular format with one dimension being the set of qualities available in a domain and the other dimension being the list of entities in the application entity model (as seen in the top right corner of Figure 8). A user ticks a checkbox corresponding to a (*entity*, *quality*) pair to apply *quality* to *entity* to form a *quality objective*. The application entity model is also visualized (bottom left corner).

If a user applies a quality to an entity, the quality is also automatically applied to all specializations of this entity. Qualities that cannot be realized by the Web Services quality model (*i.e.* not supported by WS-\*, WS-\*Policy) are not displayed for selection. Invalid combinations, that is, qualities that are not applicable on some types of entities, as discussed in Section 3.3 are also disabled from user selection.

If a user clicks “Apply”, the tool associates the formed quality objectives with the natural language representation of the policy. It then refines the business policy into a system-level policy by generating a set of WS-Policy statements that correspond to these objectives. In the current implementation, the tool only supports refinement in the security domain and assumes the *Basic Security Profile 1.0* [14] to be applied. During refinement, the tool automatically uses the first available branch in the *binding tree*

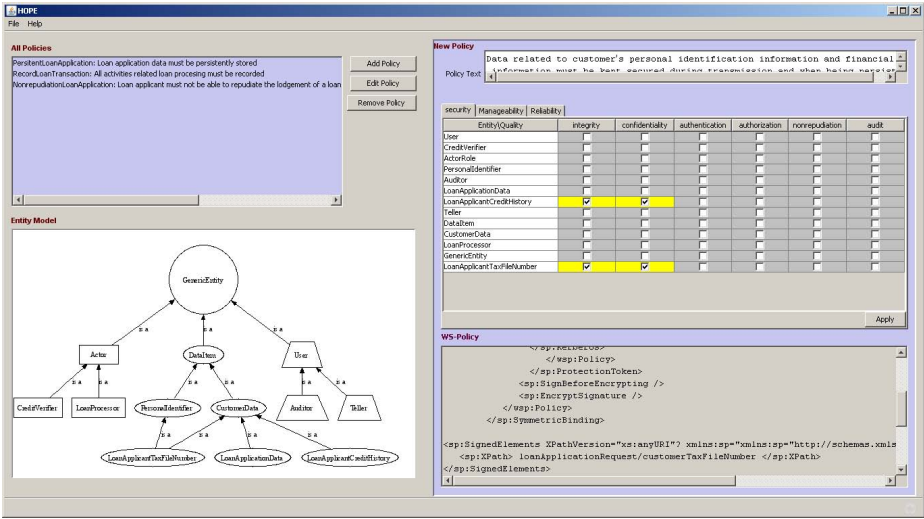


Fig. 8. Screenshot of the HOPE prototype

unless the tree is annotated with user’s preferences. In that case, the branch with the highest priority is followed.

## 6 Related Work

Our work is related to a number of areas, including business rule specification and management, business rule refinement, and Service Level Agreement (SLA) specification and management.

There has been a body work on the specification and management of SLA and Service Level Objectives that focuses on ensuring the delivery of quality services to clients. Keller and Ludwig [18] proposed the Web Service Level Agreement (WSLA) standard to allow the specification of service level agreements and objectives for Web Services. SLANG [19] is another effort of specifying end-to-end service level contracts between a client and the service. Their work considers quality of service more from a client point of view and thus focuses on the specification of rules to govern client-service interactions. Our research investigates quality of services from an enterprise system governance and management viewpoint of which the purpose is to have a dependable/interoperable SOA ecosystem including the SOA services and service clients and also all the applications built on top of the services themselves. As such, the types of interactions we are concerned with are not limited to client-service interactions.

The objective of our work is more aligned with that of business rule specification and management as we are concerned with services being compliant to high-level business rules and regulations. In this field, several business rules specification languages and frameworks have been proposed [20,21] and some rule engines have been built to support the execution of business rules. However, the target business rules are more

concerned with business logic and functional logic (e.g., “if (*FlightBookingActivity* is performed) then (*Role type is airline*)” [20]) while we focus on the non-functional quality properties related to the various entities.

In the area of policy-based specification and refinement, frameworks such as [22,23] support platform-independent specification of non-functional requirements such as those related to access control or configuration management in the form of policy statements. However, these frameworks are mainly for resource management and cannot be easily adapted for SOA systems as discussed in our previous work [24].

There have been a number of attempts to apply model-driven architecture (MDA) techniques for the modeling and translation of SOA qualities into system-level realization mechanisms [25,26]. In these approaches, quality properties of services and applications are modeled in platform-independent ways and then transformed into platform-dependent code and configurations for middle-ware to realize these qualities. However, the entities being modeled, even though being platform-independent, are still technical entities (i.e. they represent technical concepts such as *filter*, *connector*, *service*, or *proxy*), not business-oriented entities. This not only limits the participation of business analysts and IT compliance officers in the modeling process, but also makes it difficult to align the models with the original business requirements.

## 7 Conclusions and Future Work

In this paper, we introduced the HOPE framework that, in a systematic manner, assists practitioners in defining quality-oriented business policies and refining them into system-level Web Service policies in order to realize quality requirements in the service-based applications. Central to the framework are domain-specific quality models, each of which codifies the quality attributes and their realization mechanisms in a given domain. Based on these quality models and an application’s business entity model, quality-oriented business policies applicable to the application can be stated as quality objectives. Again using the quality models, quality objectives can be refined into Web Service policies as part of the application’s Web Service-based implementation. This framework assists system developers in performing such tasks with a systematic approach and associated models, techniques, and tool support.

In general, HOPE does not aim for fully automated policy refinement as decomposing a high-level business policy into a set of system-level policies requires complex modeling and reasoning. HOPE’s approach is that human decision should be leveraged when a policy statement in natural language needs to be interpreted and decomposed into a set of quality objectives. On the other hand, automation is provided (as much as possible) to refine these quality objectives into system-level statements, thereby abstracting away the complexity of the system-level infrastructure.

We expect that the HOPE framework can be adapted or generalized to be used with other policy frameworks/platforms in addition to WS-Policy. As part of future work, we will further examine the relationships among business policies, system requirements, system qualities, service/composition design, and system-level policies to improve the system development process and a system’s adaptability and evolvability.

## References

1. Sarbanes, P.: Sarbanes-Oxley Act of 2002. The Public Company Accounting Reform and Investor Protection Act. Washington, DC, US Congress (2002)
2. Basel, I.: Basel II: International Convergence of Capital Measurement and Capital Standards: a Revised Framework (2004)
3. O'Brien, L., Merson, P., Bass, L.: Quality attributes for service-oriented architectures. In: SDSOA 2007: Proceedings of the International Workshop on Systems Development in SOA Environments, Washington, DC, USA, p. 3. IEEE Computer Society, Los Alamitos (2007)
4. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Malhotra, A., et al.: Web Services Policy Framework (WS-Policy). Version 1(2), 2003–2006 (2006)
5. America, Bank secrecy act of 1970 (1970)
6. Australia, Privacy act 1988 (1988)
7. Bücker, A.: ITS Organization IBM Corporation, Understanding SOA Security Design and Implementation. Books24x7.com (2005)
8. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R., et al.: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). OASIS Standard 200401 (2004)
9. Kim, A., Luo, J., Kang, M.: Security ontology for annotating resources. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1483–1499. Springer, Heidelberg (2005)
10. I. JTC, SC27/WG3. Common Criteria for Information Technology Security Evaluation (1998)
11. Khan, K.M., Han, J.: Assessing Security Properties of Software Components: A Software Engineer's Perspective. In: Han, J., Staples, M. (eds.) Proceedings of the 17th Australian Software Engineering Conference (ASWEC 2006), Sydney, Australia, pp. 199–208. IEEE Computer Society Press, Los Alamitos (2006)
12. Meier, J., Mackman, A., Dunner, M., Vasireddy, S.: Building Secure ASP .NET Applications: Authentication, Authorization, and Secure Communication. Microsoft Patterns and Practices. Microsoft Corporation, pp. 354–362 (2002)
13. Steel, C., Nagappan, R., Lai, R.: Core Security Patterns. Prentice-Hall, Englewood Cliffs (2006)
14. McIntosh, M., Gudgin, M., Morrison, K., Barbir, A.: Basic Security Profile Version 1.0. WS-I Standard 30 (2007)
15. Kaler, C., Nadalin, A., et al.: Web Services Security Policy Language (WS-SecurityPolicy) (2005)
16. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., Verma, K.: Web Service Semantics-WSDL-S, W3C Member Submission (2005)
17. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Malhotra, A., Maruyama, H., et al.: Web Services Policy Attachment (WS-PolicyAttachment), W3C Member Submission (April 2006)
18. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management* 11(1), 57–81 (2003)
19. Lamanna, D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements. In: Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS, pp. 100–106 (2003)
20. Orriens, B., Yang, J., Papazoglou, M.P.: A Framework for Business Rule Driven Web Service Composition. In: Jeusfeld, M.A., Pastor, Ó. (eds.) ER Workshops 2003. LNCS, vol. 2814, pp. 52–64. Springer, Heidelberg (2003)

21. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosz, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission (2004)
22. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
23. Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., Lott, J.: Chaos policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement. In: Proceedings of 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), June 2003, pp. 93–96 (2003)
24. Phan, T., Han, J., Schneider, J.-G., Ebringer, T., Rogers, T.: A Survey of Policy-Based Management Approaches for Service Oriented Systems. In: Hussain, F.K., Chang, E. (eds.) Proceedings of the 19th Australian Software Engineering Conference (ASWEC 2008), Perth, Australia, pp. 392–401. IEEE Computer Society Press, Los Alamitos (2008)
25. Wada, H., Suzuki, J., Oba, K.: A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture. *International Journal of Web Services Research* 5(4), 1–31 (2008)
26. Nakamura, Y., Tsubori, M., Imamura, T., Ono, K.: Model-Driven Security based on a Web Services Security Architecture. In: Proceedings of International Conference on Services Computing, July 2005, pp. 7–15 (2005)

# Adaptation of Web Service Composition Based on Workflow Patterns

Qiang He<sup>1,2</sup>, Jun Yan<sup>3</sup>, Hai Jin<sup>1</sup>, and Yun Yang<sup>2</sup>

<sup>1</sup> School of Computer Science and Technology,  
Huazhong University of Science and Technology, Wuhan, China 430074  
hjin@hust.edu.cn

<sup>2</sup> Faculty of Information and Communication Technologies,  
Swinburne University of Technology, Melbourne, Australia 3122  
qhe@ict.swin.edu.au, yyang@swin.edu.au

<sup>3</sup> School of Information Systems and Technology,  
University of Wollongong, Wollongong, Australia 2522  
jyan@uow.edu.au

**Abstract.** Business processes consisting of component Web services are often executed in volatile environments where the quality of service parameters of the participating services might change during the execution of the business processes. Recently, research has been carried out on adapting composite Web service in volatile service-oriented computing environments. However, current approaches do not consider the internal logic of the business process and the impact of adaptation for a single service on the other component services. Other than quality of service parameters, effective adaptation requires specific information of the component services in terms of their position and interaction in the business process. The work reported in this paper is a first step in this direction. We present a novel approach to adaptation of Web service composition based on workflow patterns. This approach measures the value of changed information that updated services may potentially introduce in the business process. Experimental results show that our approach provides effective adaptation solutions by expanding the adaptation scope and considering the internal logic of business process.

**Keywords:** Adaptation, Business Process, Quality of Service, Adaptation, Service Level Agreement, Web Service Composition.

## 1 Introduction

In service-oriented computing (SOC), one of the most important functions is to create value-added services, i.e. service compositions, by composing existing services, namely component services. A composite service can be modelled as a business process with the internal logic between the component services captured using a business process modelling language tailored for Web services, e.g. BPEL [1, 12], WSCI [3] and BPSS [5]. The delivery of a composite service is achieved through the coordinated invocation of component services.

Quality of service (QoS) control for service compositions is very difficult to manage due to the cross-organisation and location-distribution of Web services. To solve this problem, service level agreements (SLA) [10, 16], referring to mutually agreed understanding and expectation about service provision, can be established between service consumers and service providers. WS-Agreement [2, 13] is the de facto SLA specification standard for Web services. In service composition scenarios, QoS control is critical, as any failure to meet local requirements of the component services may result in exceptions of the composite service. In these scenarios, the service consumer will contract SLAs with the service providers over each individual component service. The aggregation of QoS of component services is supposed to meet the global requirements of the client. However, the SOC environment is always volatile in which contracted SLAs would potentially be violated due to the inherent unreliability of the underlying Internet and internal infrastructures of service providers. When a violation of SLA occurs, the corresponding component service needs to be recovered with satisfying QoS to meet the global business process requirements. During the recovery process, cost applies. Therefore, when recovering the failed service the service provider that brings the greatest estimated profit - estimated value minus cost - should be selected. The value of changed information (VOC) [6] is used to compute the tradeoff between the expected value and the cost from updating the service. The update is performed only when it is going to pay off. Here the VOC computation shares its concepts with the value of perfect information (VPI) [15], which also attempts to decide whether new information is necessary and useful to a particular process, as explained in [6]. Usually, "update" refers to changing QoS of the services. Since recovering the services also incurs cost while bringing benefit, VOC related mechanisms can also be applied to recover services. There are two typical methods to recover failed services in an executing business process instance: 1) renegotiate with the failed service provider; 2) replace the failed service provider with a new service provider. Both methods will incur recovery cost. Generally speaking, the recovery cost occurs upon querying the information, renegotiating and negotiating with service providers, and switching service providers. The optimal solution for service adaptation is to select the solution that brings the most tradeoff, i.e. VOC minus recovery cost. The computation of the VOC and recovery cost are domain-specific and thus out the scope of this paper.

So far, little efforts have been placed on the business process adaptation which considers the impact of the adaptation for one service on the other services. When service recovery is required, e.g. current service providers claim to be incapable of providing promised services or service providers fail to deliver results as specified in the SLA, it is sometimes essential to update a group of component services because otherwise the global QoS requirements of the business process cannot be met. For example, it takes a certain amount of time to recover the failed service, which, with the addition of pre-specified time consumption of the remaining unexecuted component services, will lead to violation of the global requirements of time consumption on the whole business process and thus requires adaptation of more unexecuted component services. Therefore, a comprehensive adaptation mechanism that involves recovering and updating services is imperative to the generation of optimal adaptation solution. Besides, adaptation is supposed to be confined within a certain scope, usually the smaller the better, to limit the impact of failed services on other services and the cost



and complexity of adaptation. Hence, defining the scope and furthermore, the adaptation solution is considerably significant. Workflow patterns [17] define specific internal workflow logics – elicited from multiple homogeneous cases - and provide reusable models for developers to deliver solid, proper architecture solutions. In this paper, we discuss how to apply VOC to the generation of adaptation solutions in the scope of component services of the business process captured using pre-defined workflow patterns.

The rest of the paper is organised as follows. Section 2 introduces the major related work. Then, Section 3 analyses the requirements of adaptation in service composition scenarios by illustrating a motivating example. After that, Section 4 discusses the VOC computation based on workflow patterns followed by Section 5 which presents a method that utilises the approach presented in Section 4. Finally, Section 6 describes the experiments to demonstrate the effectiveness of the approach and Section 7 summarises the major contribution of this paper and outlines authors' future work.

## 2 Related Work

Recently, business process adaptation in dynamical and volatile environments has attracted increasing attention. Harney and Doshi [6] present a mechanism called VOC which computes the estimated value brought by the changes of the business process and compares the value to the cost required to make the changes. The update is performed only when it is expected to pay off. In [7], Harney and Doshi utilise service expiration times to reduce the computational overhead of adaptation. The improvement is based on the insight that service providers often keep the quality of their services at a certain level for a period of time. A new approach is proposed, namely VOC with expiration times (VOC<sup>e</sup>). VOC<sup>e</sup> manages to reduce the computational burden of adaptation. However, while considering adapting one individual service, the effect of VOC and VOC<sup>e</sup> is limited on just the one service without taking into account the global business process. In other words, VOC and VOC<sup>e</sup> facilitate only local adaptation solutions for business process which cannot guarantee the satisfaction of the global business process requirements.

Chafle et al. [4] introduce adaptation on different levels, including the instance level, logic level and physical level. Multiple backup workflows are prepared to substitute the failed components or workflows at any moment. By enabling this, workflow systems can adapt to environmental changes. Verma et al. [18] introduce a suite of stochastic optimisation based methods, including centralised and decentralised ones for adapting business process modelled as Markov decision processes. Exogenous events and inter-service constraints are both taken into account when performing the adaptation. Narendra et al. [14] use the aspect-oriented programming (AOP) technology to dictate modifications in component services in order to meet non-functional requirement changes in the composite service. None of the above approaches consider the cost that may occur in the adaptation of business processes and may generate worthless adaptation solutions which may bring cost more than profit. Our approach aims at addressing the above mentioned shortcomings.

### 3 A Motivating Example

In order to illustrate the needs for and the complexity of service composition adaptation, in this section we present a motivating example.

This example business process involves a trading company, a manufacturer, a road transportation company and a shipping company. The trading company wants to purchase some goods which will be produced by the manufacturer, and then delivered by road transportation company A and shipping company A in sequence. The SLAs for the manufacturing, road transportation and shipping have been negotiated before services are provided, which collectively fulfil the global requirements. Suppose that the road transportation company A suddenly claims a delay and cannot provide the service within the timeframe as promised in the SLA, this may delay the whole process. Therefore, the business process must have adaptation capabilities to recover from this exceptional situation. In addition, it is possible that the schedule of shipping company A will also be disturbed. For example, the shipping company may specify the constraint such as “Shipping service will be completed with 10 days *during May 1 to May 30.*” in SLA. Therefore, with the road transportation service rearranged, this constraint may no longer be held. Thus, the shipping service needs to be coordinated by either renegotiating over the shipping SLA between the trading company and shipping company A or selecting a new shipping company.

In the adaptation process, cost applies, including the service query cost and the negotiation cost. The trading company may recover the road transportation service by either renegotiating with road transportation company A or finding another service provider, say road transportation company B or C, to replace road transportation company A. To select from the candidate service providers for road transportation service, the trading company needs to estimate the value and cost that candidate service providers may bring. Meanwhile, the trading company needs to compute the probability that the recovery of road transportation service will lead to disturbance of the shipping service and the corresponding value and cost from adapting the shipping service. Based on the results, the trading company will decide how to recover the road transportation service, whether to update the shipping service and if so, how.

In this case, the optimal solution is not necessarily selecting the companies with the best quality but the one with the best tradeoff between the value and cost. This motivating example reveals three important factors for selecting the optimal adaptation solution. First, the trading company must estimate the tradeoff between the value and cost brought by the candidate service providers. Second, the trading company must also estimate the impact drawn from the recovered services on other services to see if they need to be updated. Third, the cost for rearranging other services must be considered.

### 4 Adaptation for Composite Service Based on Workflow Patterns

Web service composition can be modelled as executable process using Web service composition languages like BPEL and WSFL incorporating the concepts and mechanisms in the workflow community. Therefore, the logic in a composite service can be captured using workflow patterns. In this section, we consider the workflow patterns

presented in [17], and discuss and analyse how the VOC mechanism can be extended based on these patterns to facilitate adaptation in Web service composition. The services discussed are generic and their states are recoverable.

#### 4.1 Sequence Pattern

● **Pattern 1 Sequence.** A *Sequence* pattern describes the structure where a component service starts after the completion of another component service in the same process.

When service  $A$  violates the SLA, there are two ways to recover it. The business process manager can either renegotiate with the current service provider to see if it can still provide service  $A$  to meet the global business process requirements, or find a new service provider to replace it. Before performing the adaptation process, the VOC of each of the above measures needs to be computed.

Since the actual values of the updated QoS are not known until after querying the service providers, we average all the possible combination of the values of updated QoS of the service using current belief distributions which can be obtained from the pre-defined SLAs, previous interactions with the service providers or a third SLA profiling center [8]. For each candidate service provider for service  $A$  (including the original service provider), an estimated value is computed. Formally,

$$V(A|A') = \int_{\Omega_{A'}} u(e, p) \cdot Pr(E_{A'} = e, P_{A'} = p) d\Omega_{A'} \quad (1)$$

where  $V(A|A')$  is the estimated value from recover service  $A$  with service provider  $A'$ ,  $u(e, p)$  is a utility function which computes the utility of a service based on the client's preferences for execution time,  $e$ , and price,  $p$ ,  $Pr(E = e, P = p)$  denotes the belief distribution of the combination  $(e, p)$  and  $\Omega_{A'} = \langle (e_1, p_1), (e_2, p_2), \dots, (e_m, p_m) \rangle_{A'}$  represents the possible combinations of the values of execution time and price from the candidate service providers for service  $A$ . Here we take the originally contracted service provider as a candidate service provider if it is still potentially capable of providing the service with amended SLA.

Then the process manager computes the probability that service  $B$  needs to be re-arranged due to selecting service provider  $A'$  to recover service  $A$ . The probability, denoted as  $P(A|A' \rightarrow B)$ , is affected by three main aspects:

- 1) the extra resource consumption caused by the adaptation of service  $A$ . For example, delay and increased price from adapting service  $A$  will increase the need of updating service  $B$ ;
- 2) the margins of the related terms in the SLA contracted for services  $A$  and  $B$ . If the business process manager had succeeded in contracting an SLA with extra marginal violation tolerance, there would be a relatively low chance that service  $B$  needs to be adapted; and
- 3) the estimated capability of candidate service providers for service  $A$ . If it is expected to seal a deal with a candidate service provider that is capable of providing service  $A$  with better QoS, the need of updating service  $B$  will decrease.

---

<sup>1</sup> In this paper we use execution time and price for the purpose of demonstration.

In the sequence pattern, the formal computation of  $P_{Seq}(A|A' \rightarrow B)$  is:

$$P_{Seq}(A|A' \rightarrow B) = \frac{1}{|\Omega_{A'}|} \cdot \int_{\Omega_{A'}} isGlobalRequirementViolated(e, p) \cdot Pr(E_{A'} = e, P_{A'} = p) d\Omega_{A'} \quad (2)$$

where  $isGlobalRequirementViolated(e, p)$  is a function that, given the execution time and price, returns 1 if the global business process requirements will be violated and 0 otherwise. This function involves QoS aggregation [9] and multiple criteria decision making (MCDM) [11] and is implemented application-specifically.

Then we formulate the VOC due to the service adaptation as:

$$\begin{aligned} VOC_{A|A'+B|B'} &= V(A|A') + P(A|A' \rightarrow B) \cdot V(B|B') \\ &= \int_{\Omega_{A'}} u(e, p) \cdot Pr(E_{A'} = e, P_{A'} = p) d\Omega_{A'} \\ &\quad + P_{Seq}(A|A' \rightarrow B) \cdot \int_{\Omega_{B'}} u(e, p) \cdot Pr(E_{B'} = e, P_{B'} = p) d\Omega_{B'} \quad (3) \end{aligned}$$

where  $VOC_{A|A'+B|B'}$  is the value brought from adapting service  $A$  with service provider  $A'$  and service  $B$  with service provider  $B'$ .

Since adapting services  $A$  and  $B$  may be expensive, the adaptation is performed only when it is expected to pay off. The adaptation cost, including querying information, renegotiating and negotiating with service providers, and switching service providers, must be lower than the corresponding VOC. Formally, the adaptation is performed when:

$$VOC_{A|A'+B|B'} > COST(A|A'+B|B') \quad (4)$$

where  $COST(A|A'+B|B')$  is the adaptation cost from recovering service  $A$  with service provider  $A'$  and service  $B$  with service provider  $B'$ . When there are more than one qualified combination of candidates for services  $A$  and  $B$ , the one with the greatest profit, i.e.  $VOC_{A|A'+B|B'} - COST(A|A'+B|B')$ , is selected.

From formulas (1) - (4), we can observe that it is the combination of candidates for services  $A$  and  $B$  that determines the profit from the adaptation process. The business process manager needs to compute the VOC of all the possible combinations of candidates for services  $A$  and  $B$ , which is computationally intensive if the number of candidates is large. Under this circumstance, the business process manager can select several candidates according to the ranking provided by an SLA profiling centre.

## 4.2 Parallel Patterns

Besides the sequence pattern, parallel is another major pattern in any model of business processes. The authors in [17] consider the parallel patterns in terms of (1) how the branches are picked, (2) how they are executed and (3) how they converge. In this section, we analyse the adaptation mechanisms for different types of parallel patterns considering the three aspects above.

There are three typical split patterns that describe the logic of processes splitting and proceeding:

- **Pattern 2 Parallel Split.** A *Parallel Split* describes the structure where a single thread splits into multiple threads which can be executed in parallel. In this pattern, component services *A* and *B* will both be executed and can be executed simultaneously in any order.
- **Pattern 3 Exclusive Choice.** An *Exclusive Choice* pattern describes the structure where, based on a decision or process control data, only one selected branch is activated and executed.
- **Pattern 4 Multi-Choice.** A *Multi-Choice* pattern describes the structure where, based on a decision or process control data, a number of branches are chosen.

We must also consider how the branches will converge (if they will). There are five typical patterns that model the logic of the branches converging:

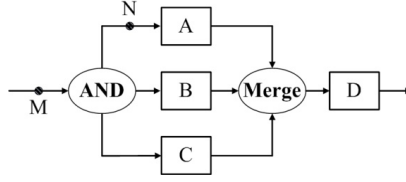
- **Pattern 5 Synchronisation.** A *Synchronisation* pattern describes the structure where multiple parallel branches converge into one single thread synchronised.
- **Pattern 6 Simple Merge.** A *Simple Merge* pattern describes the structure where more than one branches converge without synchronisation and only one of them has ever been executed.
- **Pattern 7 Synchronising Merge.** A *Synchronising Merge* pattern describes the structure where synchronising happens only when more than one branches are active (i.e. they are being executed).
- **Pattern 8 Multi-Merge.** A *Multi-Merge* pattern describes the structure where the branches converge without synchronisation and the service succeeding the merge will be activated by the completion of every incoming branch.
- **Pattern 9 Discriminator.** A *Discriminator* pattern describes the structure where the subsequent service will be activated by the first and only the first completed branch. The remaining branches will be ignored.

The combination of the split patterns and converge patterns determines the adaptation solution when services need to be recovered. Starting with the simplest combination, i.e. *Parallel Split* + *Synchronisation*, we discuss the corresponding adaptation mechanisms.

### Parallel Split + Synchronisation

In this pattern combination, there are more than one branches splitting at a certain point and then converge with synchronisation at the end of the completion of all the branches. Here we suppose there is only one service on each parallel branch. This assumption is realistic because if there are more than one services on any branch they can be considered as a composite service. When the service on one of the parallel branches needs to be recovered, the business process manager must compute the VOC of the service and select an appropriate service provider. Moreover, to guarantee global business process requirements satisfaction, the business process manager must also consider if it is necessary to update the services on other branches.

We consider the adaptation under two circumstances: 1) all the branches have not been activated and executed and the service provider for one of them claims to be unable to provide the service, or incapable of providing the service with promised



**Fig. 1.** Parallel Split + Synchronisation

quality. In this case, the adaptation happens before point M in Figure 1; 2) all the branches have been activated and one of the service providers fails to deliver the required result. In this case, the adaptation happens after points like N.

In case 1), the business process manager needs to estimate if the adaptation of failed service, i.e. service A in Figure 1, will cause the delay of activation of service D due to the time consumption of the adaptation process and the newly contracted SLA. If it is true, the business process manager can think about giving services B and C more time to complete because given more flexible time constraint, the business process manager may be able to renegotiate with service providers of B and C for a lower price or better QoS if the SLAs previously contracted for services B and C are renegotiable and modifiable.

For the pattern combination, *Parallel Split + Synchronisation*, the probabilities of additional execution time for services B and C caused by the adaptation of service A, assuming service A is replaced by service A', are computed as:

$$P_{PSP}(A|A' \rightarrow B) = \frac{1}{|E_{A'}|} \cdot \int_{E_{A'}} \cdot Pr(E_{A'} = e) \cdot isBigger(e, e_B) dE_{A'} \quad (5)$$

$$P_{PSP}(A|A' \rightarrow C) = \frac{1}{|E_{A'}|} \cdot \int_{E_{A'}} \cdot Pr(E_{A'} = e) \cdot isBigger(e, e_C) dE_{A'} \quad (6)$$

where  $e_B$  and  $e_C$  are the execution times for services B and C,  $|E_{A'}|$  is the modulo of  $E_{A'}$ , and  $isBigger(x, y)$  is a function returns 1 when x is bigger than y and 0 otherwise. Then the VOC of different adaptation strategies, involving services A, B and C, can be computed as:

$$VOC_{PSP}^{A|A'+B|B'+C|C'} = V(A|A') + P(A|A' \rightarrow B) \cdot V(B|B') + P(A|A' \rightarrow C) \cdot V(C|C') \quad (7)$$

In case 2), when the exception is detected on service A, services on other branches, i.e. services B and C, have already been activated. When the adaptation for service A is being performed, services B and C are already under execution. Therefore, the estimated time consumption from adapting service A, must be involved in the computation of  $P(A|A' \rightarrow B)$  and  $P(A|A' \rightarrow C)$ :

$$P_{PSP}(A|A' \rightarrow B) = \frac{1}{|E_{A'}|} \cdot \int_{E_{A'}} \cdot Pr(E_{A'} = e, E_{adap} = e_{adap}) \cdot isBigger(e + e_{ADAP}, e_B) dE_{A'} \quad (8)$$

$$P_{PSP}(A|A' \rightarrow C) = \frac{1}{|E_{A'}|} \cdot \int_{E_{A'}} \cdot Pr(E_{A'} = e, E_{adap} = e_{adap}) \cdot isBigger(e + e_{ADAP}, e_C) dE_{A'} \quad (9)$$

where  $e_{ADAP}$  is the estimated time to adapt service A.

Then the VOC of different adaptation solutions can be computed using formula (7) with  $V(A|A')$ ,  $V(B|B')$  and  $V(C|C')$  computed similarly as formula (1).

### Parallel Split + Multi-Merge

In a *Multi-Merge* pattern, the service succeeding the merge will be executed every time an incoming branch completes. Most of the workflow products, e.g. Eastman, Verve Workflow and Forte Conductor, implement the Multi-Merge pattern by replicating the service(s) succeeding the merge (see Figure 2 for a simple example). And the replicated services will be made sequential to the services on each of the original branches, generating several independent sequence structures. Actually, at runtime, no *Multi-Merge* structures will be found. Therefore, VOC computation for *Sequence* pattern will be applied to the created *Sequence* structures in adaptation.

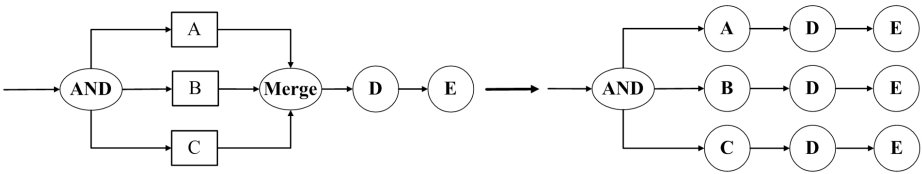
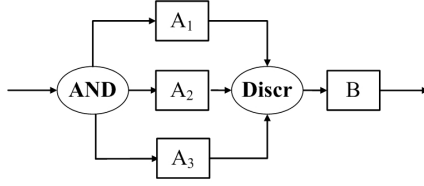


Fig. 2. Implementation of Multi-Merge pattern

### Parallel Split + Discriminator

In a *Discriminator* pattern, the service succeeding the merge waits for the first completed incoming branch and ignores the rest. In other words, the succeeding service will be activated only once when one of the incoming branches is firstly completed. We now discuss how to adapt the parallel services in terms of VOC.

In this pattern combination the fastest branch (the one with the shortest execution time) determines the start time of the service succeeding the merge. Once the succeeding service is activated, the uncompleted branches will be ignored. In fact, *Discriminator* pattern is not found often in business processes where SLA is enabled because generally the execution time of the services will be specified in the SLA and hence it can be estimated that which branch is likely to complete first and which branches will be ignored. However, there is one exception: the business process manager wants to hedge the risk of delay caused by service provider failing to deliver expected result. In this case, by employing the *Discriminator* pattern, when a branch is broken, other branches can still deliver expected result in a relatively tolerant period of time. Intuitively, the most effective way to hedge the risk is to allocate low execution time for individual branches while obtaining a high successful global execution rate of the branches. It is also the major objective of adapting the branches when a branch is broken. Since the services on different branches are functionally equivalent, they share a group of candidate service providers and a utility function. For the demonstration purpose, we use the example in Figure 3.



**Fig. 3.** Parallel Split + Discriminator

First, the normalised successful execution rate weights,  $w_1$ ,  $w_2$  and  $w_3$ , are assigned to the candidate service providers. The success rate weights range from 0 to 1, representing how much trustworthy the service providers are based on their historical performance. Service providers with better historical performance, i.e. higher successful rate, will be assigned with higher weights. The historical performance of service providers may be provided by the service providers through pre-defined SLAs or they could be learnt from previous interactions with the service providers.

Second, the VOC can be computed as:

$$VOC_{PSD}^{A|A'+B|B'+C|C'} = \int_{\Omega_{(A;B;C')}} w_1 \cdot u(e_{A'}, p_{A'}) \cdot Pr(E_{A'} = e_{A'}, P_{A'} = p_{A'}) + w_2 \cdot u(e_{B'}, p_{B'}) \cdot Pr(E_{B'} = e_{B'}, P_{B'} = p_{B'}) + w_3 \cdot u(e_{C'}, p_{C'}) \cdot Pr(E_{C'} = e_{C'}, P_{C'} = p_{C'}) d\Omega_{(A;B;C')} \quad (10)$$

Based on formula (10), the combination of candidate providers that maximises  $VOC_{PSD}^{A|A'+B|B'+C|C'} - Cost(A|A') - Cost(B|B') - Cost(C|C')$  will be selected.

Sometimes which branches will be executed is dependent on runtime decision making. Pattern 3 *Exclusive Choice* and pattern 4 *Multi-Choice* describe the two different situations in this category. In pattern 3 *Exclusive Choice*, only one branch will be chosen and executed in a running process instance and it leads to a *Simple Merge*. That makes the branches uninfluential on one another. Therefore, when one branch needs to be recovered, other branches do not need to be considered. Yet the VOC mechanism can still be applied here because, the broken branch, together with the service succeeding the merge, can be seen as a *Sequence* pattern. In pattern 4 *Multi-Choice*, multiple branches will be chosen for execution. Therefore, the business process manager needs to consider the other branches when trying to recover a branch. Next we discuss the pattern combinations involving the *Multi-Choice* pattern.

### Multi-Choice + Synchronizing Merge

In this pattern combination, multiple branches will be chosen for execution and they will be synchronised when they merge. The way the branches merge is similar to pattern 5 *Synchronisation*. The difference is that not all the incoming branches will be activated for every running instance. If one of the branches is broken and needs to be recovered, the business process manager needs to estimate the probabilities of other branches being activated and whether they can benefit from updating.

Generally, there is no way to ascertain which branches will be activated for a specific running instance until the dynamic decision is made. However, the business process manager can estimate the probability that a branch will be activated based on the historical performance of the branches. For example, if a branch was executed 80 times out of the last 100 business process instances, we consider the branch will be selected with



a probability of 80%. The probability of being selected for each branch can be normalised as weights,  $s_1, s_2, \dots, s_n$ , ranging between 0 and 1 representing how important the branches based on the probability they will be selected for execution. Take the business process in Figure 1 as an example, with *Parallel Split* replaced with *Multi-Choice*. Based on this assumption the computation of the VOC for the adaptation solution can be formalised as:

$$VOC_{MCSM}^{A|A'+B|B'+C|C'} = s_1 \cdot V(A|A') + s_2 \cdot P(A|A' \rightarrow B) \cdot V(B|B') + s_3 \cdot P(A|A' \rightarrow C) \cdot V(C|C') . \quad (11)$$

where  $V(A|A')$ ,  $V(B|B')$  and  $V(C|C')$  are computed similarly to formula (1),  $P_{MCSM}(A|A' \rightarrow B)$  and  $P_{MCSM}(A|A' \rightarrow C)$  are computed similarly to formulas (5) and (6).

### Multi-Choice + Multi-Merge

Similar to the *Parallel Split + Multi-Merge* pattern combination, in *Multi-Choice + Multi-Merge* pattern combination the branches will be transformed into several independent sequence structures before being executed. Thus, VOC computation for *Sequence* pattern will be applied when necessary.

### Multi-Choice + Discriminator

In this pattern combination, a number of branches are selected and executed in parallel based on a decision dynamically made. The first branch that completes will trigger the service succeeding the merge and after that other branches will be ignored. As discussed before, the aim of employing the *Discriminator* pattern is to achieve relatively tolerant execution time when service failure happens. Therefore, in adaptation solution determination, the branches that have higher successful execution rates and higher probabilities of being selected should be given higher preference. Here we adopt the weights used before:  $w$ , representing the successful execution rate, and  $s$ , representing the probability of branches being selected. Again, for three parallel branches, the formalised VOC computation is:

$$VOC_{MCD}^{A|A'+B|B'+C|C'} = \int_{\Omega_{A;B;C'}} s_1 \cdot w_1 \cdot u(e_{A'}, p_{A'}) \cdot Pr(E_{A'} = e_{A'}, P_{A'} = p_{A'}) + s_2 \cdot w_2 \cdot u(e_{B'}, p_{B'}) \cdot Pr(E_{B'} = e_{B'}, P_{B'} = p_{B'}) + s_3 \cdot w_3 \cdot u(e_{C'}, p_{C'}) \cdot Pr(E_{C'} = e_{C'}, P_{C'} = p_{C'}) d\Omega_{(A;B;C')} . \quad (12)$$

## 4.3 Other Patterns

In addition to the nine patterns addressed so far, there are other eleven patterns represented in [17] not mentioned. Some of them are used to describe the global properties and special activities of the business processes, including *Arbitrary Cycles*, *Implicit Termination*, *Multiple Instance*, *Cancel Activity* and *Cancel Case*. Another two patterns, *Deferred Choice* and *Milestone*, are used to specify the triggering condition of services based on decision dynamically made. The last pattern, *Interleaved Parallel Routing*, describes a set of services that are executed one by one in an arbitrary order decided at runtime. However, our work, VOC based on workflow patterns, is dedicated to analysing adaptation solution for business process based on specific influence

between services in a confined recovery scope described using workflow patterns. The above eleven patterns do not serve the goal and thus are excluded in discussion from this paper.

## 5 Adaptation Method

Figure 4 shows the pseudo code for adapting the business process using the mechanism presented in Section 4. The algorithm takes one input - the service that requires recovery, denoted as  $S_0$ . The algorithm starts with identifying the pattern that  $S_0$  belongs to (line 3). After the pattern identification, the VOC of different adaptation solutions is computed (line 7 for Sequence pattern and line 18 for Parallel pattern). The adaptation solution that is estimated to bring the most profit will be performed (lines 9 and 20). If the adaptation solution within the current scope specified with workflow patterns is not satisfactory, the component services in the current scope will be considered as a composite service (lines 12-13 and lines 23-24) and adaptation will be performed in a larger scope. The algorithm returns **true** if the business process is successfully adapted and **false** if all the unexecuted component services have been taken into account and still no satisfactory adaptation solution found.

```

Algorithm for Adapting Business Process
Input:  $S_0$  // service requiring adaptation
1.  $V_{s0}[] \leftarrow \text{Candidates}(S_0)$  //fetch candidates for  $S_0$ 
2. while end of business process not reached
3.   identify the pattern that  $S_0$  belongs to
4.   if it is sequence
5.      $S_1 \leftarrow S_0.\text{Next}$ 
6.      $V_{s1}[] \leftarrow \text{Candidate}(S_1)$  //fetch candidates for  $S_0$  and  $S_1$ 
       //Iterate through the combinations of  $V_{s0}[]$  and  $V_{s1}[]$ 
7.     if  $\text{Max}(\text{VOC}(S_0|S_0'+S_1|S_1') - \text{Cost}(S_0|S_0'+S_1|S_1')) > 0$ 
8.       and Global requirements met
9.         Update  $S_0$  with  $S_0'$  and  $S_1$  with  $S_1'$ 
10.        return true
11.     else
12.        $S_0 \leftarrow S_0 + S_1$ 
13.        $V_{s0}[] \leftarrow \text{Combine}(V_{s0}[], V_{s1}[])$  //combine candidates of  $S_0$  and  $S_1$ 
14.     end if
15.   end if
16.   if it is parallel
       //fetch candidates for  $V_{s1}[], \dots, V_{sn}[]$ 
17.      $V_{s1}[], \dots, V_{sn}[] \leftarrow \text{Candidate}(S_1), \dots, \text{Candidate}(S_n)$ 
       //Iterate through the combinations of candidates for  $S_0, S_1, \dots, S_n$ :
18.     if  $\text{Max}(\text{VOC}(S_0|S_0'+S_1|S_1'+\dots+S_n|S_n') - \text{Cost}(S_0|S_0'+S_1|S_1'+\dots+S_n|S_n')) > 0$ 
19.       and Global requirements met
20.         Update  $S_0$  with  $S_0'$ ,  $S_1$  with  $S_1'$ ,  $\dots$ ,  $S_n$  with  $S_n'$ 
21.        return true
22.     else
23.        $S_0 \leftarrow S_0 + S_1$ 
24.        $V_{s0}[] \leftarrow \text{Combine}(V_{s0}[], V_{s1}[])$  //combine candidates of  $S_0$  and  $S_1$ 
25.     end if
26.   end if
27. end while
28. return false
end algorithm

```

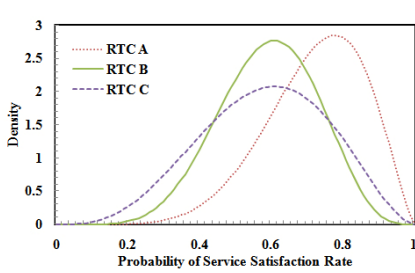
**Fig. 4.** Pseudo code for adapting a business process

## 6 Experimental Evaluation

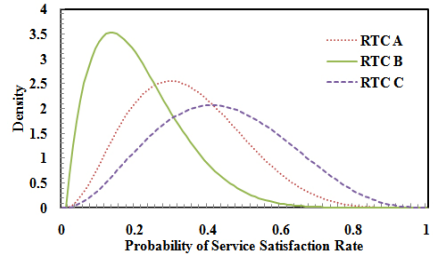
To evaluate the performance of our workflow based VOC approach for adapting business processes, we conducted experiments in a simulated volatile environment. The experimental evaluation is aimed at showing that our approach is effective in guaranteeing the satisfaction of the global business process requirements.

We utilised the goods purchase example presented in Section 3 for evaluation. This example is compliant with the *Sequence* pattern. Due to space limit, the results for other patterns are not presented. We evaluated the satisfaction rate of the global business process requirements with our adaptation approach enabled which considers both recovering the road transportation service and updating the shipping service. Considering that in different situations the difficulty levels of recovering and updating services might vary significantly, we model the road transportation companies' distribution and the shipping companies' distribution over their service satisfaction rates at two difficulty levels, i.e. easy and difficult, using *beta* distribution functions presented in Figure 5. Intuitively, service providers expose relatively high and low service satisfaction rates in respective easy and difficult situations.

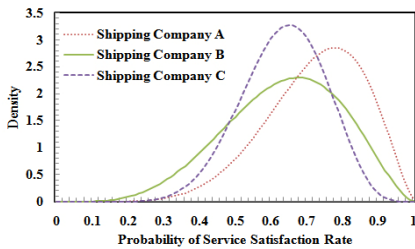
We ran 1,000 independent business process instances for each experiment within a simulated volatile environment. Since the difficulty of adapting two sequential services might vary in different situations, we conducted comprehensive experiments with all



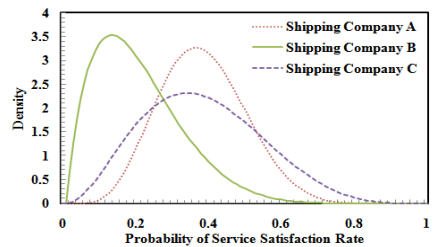
(a) Easy service satisfaction rates of road transportation companies



(b) Difficult service satisfaction rates of road transportation companies

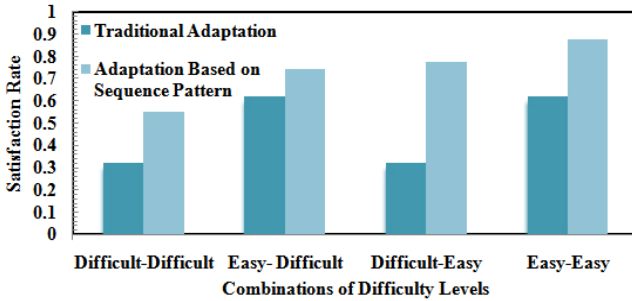


(c) Easy service satisfaction rates of shipping companies



(d) Difficult service satisfaction rates of shipping companies

Fig. 5. Probability density functions



**Fig. 6.** Comparison between traditional adaptation and our adaptation for Sequence pattern

the four combinations of difficulty levels. We measured the satisfaction rate of the global requirements by calculating the successful cases, i.e. cases where global requirements can be met after the adaptation, out of the overall cases.

Figure 6 compares the satisfaction rates of global business process requirements in different situations with and without our approach enabled. The results demonstrate that in all situations our adaptation approach provide a more effective solution to the problem of global business process requirements satisfaction. In situations with different combinations of difficulty levels, including difficult-difficult, easy-difficult, difficult-easy and easy-easy, our approach provides an increment of 22%, 12%, 46% and 25% respectively in the satisfaction rates of global business process requirements.

## 7 Conclusion and Future Work

In open SOC environments, services can be volatile. Due to the inherent unreliability of the underlying Internet and internal infrastructures of service providers, SLA violation might happen. In service composition scenarios recovering just the failed service might not satisfy the global requirements of the composite service. Hence, business process adaptation needs to consider updating a certain scope of component services while recovering the failed one. Therefore, to determine the adaptation solution, we need to identify the adaptation scope and analyse the profit from different adaptation solutions. In this paper, we have discussed how the value of changed information (VOC) is extended and applied to business process adaptation based on workflow patterns. Specifically, we have analysed and presented how to compute VOC for sequence and different parallel pattern scenarios. When the adaptation is expected to pay off, it is performed within a certain scope defined by workflow patterns. In doing so, the business process adaptation can deliver satisfactory results while being kept within a reasonable scope. The experimental results show that our approach can significantly improve the satisfaction rates of global business process requirements in different situations.

In the future, we will apply the workflow-pattern-based VOC computation mechanism to our experimental prototype to test and analyse the performance of our approach. We will also attempt to improve the accuracy of VOC computation by utilising SLA profiling centre to provide historical and real-time performance of service providers.

**Acknowledgments.** This work is partly funded by the Australian Research Council Discovery Project Scheme under grant No. DP0663841, National Science Foundation of China under grant No.90412010 and ChinaGrid project from Ministry of Education of China.

## References

1. Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1 (2003), <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>
2. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Toshiyuki, N., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement): World-Wide-Web Consortium, W3C (2007), <http://www.ogf.org/documents/GFD.107.pdf>
3. Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacs-Nagy, P., Trickovic, I., Zimek, S.: Web Service Choreography Interface (WSCI) 1.0: World Wide Web Consortium, W3C (2002), <http://www.w3.org/TR/wsci/>
4. Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., Srivastava, B.: Adaptation in Web Service Composition and Execution. In: IEEE International Conference on Web Services, pp. 549–557. IEEE Computer Society, Chicago (2006)
5. Clark, J., Casanave, C., Kanaskie, K., Harvey, B., Clark, J., Smith, N., Yunker, J., Riemer, K.: ebXML Business Process Specification Schema Version 1.01: OASIS (2001), <http://www.ebxml.org/specs/ebBPSS.pdf>
6. Harney, J., Doshi, P.: Adaptive Web Processes Using Value of Changed Information. In: 4th International Conference on Service-Oriented Computing, pp. 179–190. Springer, Chicago (2006)
7. Harney, J., Doshi, P.: Speeding Up Adaptation of Web Service Compositions Using Expiration Times. In: 16th International Conference on World Wide Web, pp. 1023–1032. ACM, Banff (2007)
8. He, Q., Yan, J., Kowalczyk, R., Jin, H., Yang, Y.: Lifetime Service Level Agreement Management with Autonomous Agents for Services Provision Information Sciences (to appear, 2008)
9. Hwang, S.-Y., Wang, H., Tang, J., Srivastava, J.: A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows. Information Sciences 177(23), 5484–5503 (2007)
10. Jin, L.-J., Machiraju, V., Sahai, A.: Analysis on Service Level Agreement of Web Services. Technical Report, HP Laboratories (2002), <http://www.hpl.hp.co.uk/techreports/2002/HPL-2002-180.pdf>
11. Köksalan, M., Zionts, S.: Multiple Criteria Decision Making in the New Millennium. Springer, Heidelberg (2001)
12. Khalaf, R., Mukhi, N., Weerawarana, S.: Service-Oriented Composition in BPEL4WS. In: 12th International World Wide Web Conference (Alternate Paper Tracks), Budapest, Hungary (2003)

13. Ludwig, H., Dan, A., Kearney, R.: Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In: 2nd International Conference on Service Oriented Computing, New York, USA, pp. 65–74 (2004)
14. Narendra, N.C., Ponnalagu, K., Krishnamurthy, J., Ramkumar, R.: Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming. In: 5th International Conference on Service-Oriented Computing, pp. 546–557. Springer, Vienna (2007)
15. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
16. Sturm, R., Morris, W., Hander, M.: Foundations of Service Level Management. SAMS (2000)
17. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B.P.B.A.: Workflow Patterns. *Distributed and Parallel Databases* 14(1), 5–51 (2003)
18. Verma, K., Doshi, P., Gomadam, K., Miller, J.A., Sheth, A.P.: Optimal Adaptation in Web Processes with Coordination Constraints. In: IEEE International Conference on Web Services, pp. 257–264. IEEE Computer Society, Chicago (2006)

# Protocol-Based Web Service Composition

Ramy Ragab Hassen, Lhouari Nourine, and Farouk Toumani

LIMOS - CNRS UMR 6158  
Universit Blaise Pascal, Clermont-Ferrand  
{ragab,nourine,ftoumani}@isima.fr

**Abstract.** We study the problem of web service protocol composition. We consider a formal framework where service business protocols are described by means of Finite State Machines (FSM) and focus on the protocol synthesis problem, i.e., how to generate automatically a new target service protocol by reusing some existing ones. We consider a general case of this problem where the number of instances of existing services that can be used in a given composition is not bounded *a priori*. We motivate the practical interest of investigating such a problem and then we prove its decidability by providing a sound and complete composition algorithm. Since the main composition algorithm is not primitive recursive, which means that no theoretical complexity bound can be computed, we evaluated experimentally the performance of the algorithm on synthetic data instances and present preliminary results in this paper.

## 1 Introduction

Web services is an emerging computing paradigm that tends to become the dominant technology for interoperation among autonomous and distributed applications in the Internet environment [1]. Informally, a *service* is a self-contained and platform-independent application (i.e., program) that can be described, published, and invoked over the network by using standards network technologies. One of the ultimate goals of the web service technology is to enable rapid low-cost development and easy composition of distributed applications, a goal that has a long history strewn with only partial successes. To achieve this goal, there has been recently numerous research work [2,3,4,5,6,7,8,9,10] on the challenges associated with web service composition. The research problems involved by service composition are varied in nature and depends on several issues such as the kind of the composition process, e.g., manual v.s. automatic, the model used to describe the services, etc (e.g., see [3]). A line of demarcation between existing works in this area lies in the nature of the composition process: manual v.s. automatic. The first category of work deals generally with low-level programming details and implementation issues (e.g., WS-BPEL[4]) while automatic service composition focuses on different issues such as composition verification [2,7,8], planning [9,10] or synthesis [4,5,6].

---

<sup>1</sup> <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>

In this paper we investigate the problem of automatic web service composition. We consider more particularly the composition synthesis problem, i.e., how to generate automatically a new target service by reusing some existing ones. We consider this problem at the web service business protocol abstraction level. A web service business protocol (or simply, a service protocol) is used to describe the external behavior of a service. Recent works have drawn attention to the importance of the state machine based formalisms for modeling the external behaviors of web services [4,11]. Continuing with this line of research, we build our work upon a formal framework where web service business protocols are described by means of Finite State Machines (FSM) and we concentrate on the following protocol synthesis problem: *given a set of  $n$  available web service protocols  $P_1, \dots, P_n$  and a new target protocol  $P_T$ , can the behavior described by  $P_T$  be synthesized by combining (parts of) the behaviors described by the available protocols*. This problem has already been addressed in recent literature [4,5,6] under the restriction that the number of instances of an existing service that can be used in a composition is bounded and fixed *a priori*. We call this restricted form of the composition problem *the bounded instances protocol (composition) synthesis problem*. It should be noted that this restricted setting is not realistic and has severe practical limitations that may impede the usage of automatic service composition by organizations. Indeed, as illustrated in section 3 of this paper, some very simple cases of web service composition cannot be solved in such a restricted setting.

**Contributions.** In this paper, we focus on the general case of protocol synthesis problem by relaxing the restriction on the number of protocol instances that can be used in a given service composition (i.e., we consider the unbounded instances case). The decision problem underlying composition existence in such an unrestricted setting is still an open problem. This paper makes the following contributions: (i) we show that the composition existence problem can be formalized as that of checking simulation between an FSM and a **product closure** of a FSM (i.e., an iteratively infinite product of FSMs), (ii) we propose a suitable model, called Product Closure State Machine (PCSM), to describe product closure of a FSM as an infinite state machine, (iii) we extend existing works in formal models area to prove the decidability of testing simulation between a FSM and a PCSM, and (iv) we provide a sound and complete web service composition algorithm and present first experimental results regarding performance evaluation of this algorithm.

**Organization.** The remainder of the paper is organized as follows. Section 2 introduce basic notions. Section 3 defines the service composition problem dealt with in this paper and points out the main theoretical and practical limitations of current state of the art. Section 4 presents the main technical results: proof of the decidability of the considered composition problem as well as a protocol-based web service composition algorithm. Section 5 describes first experimental results and Section 6 concludes and draws some directions for future works.



## 2 Preliminaries

We recall some basic notions that will be useful for the rest of this paper. A **State Machine**  $M$  is a tuple  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$ , where  $\Sigma_M$  is a finite alphabet,  $S_M$  is a set of states,  $\delta_M \subseteq S_M \times \Sigma_M \times S_M$  is a set of labeled transitions (actions),  $F_M \subseteq S_M$  is the set of final states and  $q_M^0 \in S_M$  is the initial state. If  $S_M$  is finite then  $M$  is called a Finite State Machine (FSM).

We define below the notions of intermediate and hybrid states of an FSM  $M$  which will be useful in the remainder of this paper. Let  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$  be a FSM. Then: (i) the set of *hybrid states* of  $M$ , denoted  $H_s(M)$ , contains all the final states of  $M$  that have at least one outgoing transition, and (ii) the set of *intermediate states* of  $M$ , denoted  $I_s(M)$ , contains the states of  $S_M \setminus F_M$  that have at least one incoming and one outgoing transitions. For example, the hybrid states of the FSM  $P_1$  depicted at figure [1\(a\)](#) are  $H_s(P_1) = \{VehicleSelected\}$  while the intermediate states the FSM  $P_2$  depicted at figure [1\(b\)](#) are  $I_s(P_2) = \{PaymentEstimation\}$ .

We provide below a definition of the notion of simulation relation between state machines. Let  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$  and  $M' = \langle \Sigma_{M'}, S_{M'}, F_{M'}, q_{M'}^0, \delta_{M'} \rangle$  be two state machines. A state  $q_1 \in S_M$  is simulated by a state  $q'_1 \in S_{M'}$ , noted  $q_1 \preceq q'_1$ , iff: (i)  $\forall a \in \Sigma_M$  and  $\forall q_2 \in S_M$  s.t.  $(q_1, a, q_2) \in \delta_M$  there is  $(q'_1, a, q'_2) \in \delta_{M'}$  s.t.  $q_2 \preceq q'_2$  and (ii) if  $q_1 \in F_M$ , then  $q'_1 \in F_{M'}$ .  $M$  is simulated by  $M'$ , noted  $M \preceq M'$ , iff  $q_M^0 \preceq q_{M'}^0$ .

Let  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$  and  $M' = \langle \Sigma_{M'}, S_{M'}, F_{M'}, q_{M'}^0, \delta_{M'} \rangle$  be two FSMs. The asynchronous **product** (or simply, product) of  $M$  and  $M'$ , denoted  $M \times M'$ , is a FSM  $\langle \Sigma_M \cup \Sigma_{M'}, S_M \times S_{M'}, F_M \times F_{M'}, (q_M^0, q_{M'}^0), \lambda \rangle$  where the transition function  $\lambda$  is defined as follows:  $\lambda = \{((q, q'), a, (q_1, q_1')) : ((q, a, q_1) \in \delta_M \text{ and } q' = q'_1) \text{ or } ((q', a, q_1') \in \delta_{M'} \text{ and } q = q_1)\}$ .

Let  $k > 0$  be a positive integer. The  $k$ -iterated product of a state machine  $M$  is defined by  $M^{\otimes k} = M^{\otimes k-1} \times M$  with  $M^{\otimes 1} = M$ . Recall that a state of  $M^{\otimes k}$  is given by a tuple over  $(S_M)^k$ . A **product closure** of an FSM  $M$ , noted  $M^{\otimes}$ , is defined as follows:  $M^{\otimes} = \bigcup_{i=0}^{+\infty} M^{\otimes i}$ . It is worth noting that for any finite positive integer  $k$ , the  $k$ -iterated product  $M^{\otimes k}$  is still an FSM. However, this property does not hold for  $M^{\otimes}$  since product closure leads to a state machine with an infinite number of states.

Let  $\mathcal{R} = \{P_1, \dots, P_n\}$  be a set of FSMs. In the sequel we use  $\odot(\mathcal{R})$  to denote the union of the asynchronous product of all the subsets elements of  $\mathcal{R}$ , i.e.,  $\odot(\mathcal{R}) = \bigcup_{\{P_{i_1}, \dots, P_{i_n}\} \subseteq \mathcal{R}} (P_{i_1} \times \dots \times P_{i_n})$ .

## 3 Web Services Composition

In this section we first define the service composition problem dealt with in this paper and then we point out the main theoretical and practical limitations in current state of the art.

**Web Services Protocol Model.** We consider web services described by means of their protocols. The main goal of a web service protocol is to describe the

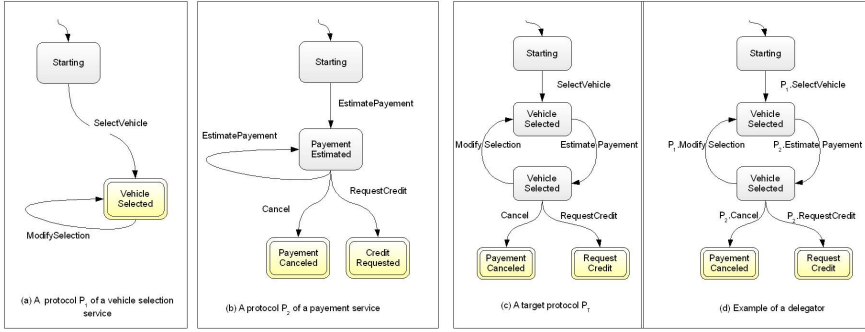
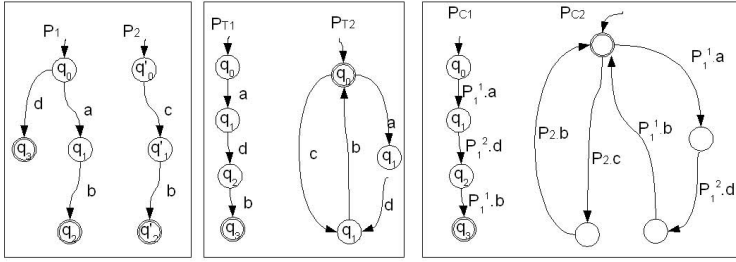


Fig. 1. An example of protocol composition

ordering constraints that govern messages exchanges between a service and its clients (i.e., messages choreography constraints). In this paper, we use the traditional deterministic finite state-machine formalism to represent messages choreography constraints (the protocol). States represent the different phases that a service may go through during its interaction with a requester. Transitions are triggered by messages sent by the requester to the provider or vice versa. Each transition is labeled with a message name. Usually the message names are followed by message polarity [11] to denote whether the message is incoming (e.g., the plus sign) or outgoing (e.g., the minus sign). For simplicity reasons, and w.l.o.g., we do not consider message polarities in this paper (i.e., incoming and outgoing messages are considered to be distinguished activities). Therefore, we obtain a web service protocol model equal the so-called *Roman model* [4], i.e., a FSM where transitions are labeled by “abstract” activities. For instance, figure 1(b) depicts the protocol of an hypothetical financing web service. The protocol specifies that the financing service is initially in the **Start** state, and that clients begin using the service by executing the activity estimate payment, upon which the service moves to the **Payment Estimated** state (transition *EstimatePayment*). In the figure, the initial state is indicated by an unlabeled entering arrow without source while final (accepting) states are double-circled.

**The Protocol Synthesis Problem.** Let us now turn our attention to the web service composition problem. We first illustrate this problem on an example. We assume a repository of two available services  $S_1$  and  $S_2$ , respectively, described by their protocols  $P_1$  and  $P_2$  depicted at figure 1(a) and (b). We consider the development of a new web service  $S_T$  whose protocol  $P_T$ , called a target protocol, is depicted at figure 1(c). An interesting question is to see whether or not it is possible to implement the service  $S_T$  by combining the functionality provided by the available services  $S_1$  and  $S_2$ . Dealing with this composition problem at the business protocol abstraction level, leads to the following question: is it possible to generate the protocol  $P_T$  by combining (parts of) the available protocols  $P_1$  and  $P_2$ . In our illustrative case the answer is yes and an example of the composition of the target protocol  $P_T$  using the protocols  $P_1$  and  $P_2$  is



**Fig. 2.** Example of the composition bounded case

depicted at figure 1(d). In this case,  $P_T$  is called the target protocol while  $P_1$  and  $P_2$  are called the component protocols. The service composition (or *protocol synthesis*) problem is defined in [4] as the problem of generating a *delegator* of a target service using available services. A delegator is a FSM where the activities are annotated with suitable *delegations* in order to specify to which component each activity of the target service is delegated. Continuing with our example, figure 1(d) shows a *delegator* that enables to *compose* the protocol  $P_T$  using the available protocols  $P_1$  and  $P_2$  of figure 1(a) and ((b). For instance, this delegator specifies that the activity `selectVehicle` of the target protocol is delegated to the protocol  $P_1$  while the activity `estimatePayment` is delegated to the protocol  $P_2$ .

The notion of a delegator is defined formally in [4] and the composition synthesis problem is expressed as the problem of finding a “correct” delegator for a given target protocol using a set of available protocols. A crucial question regarding this problem lies in the number of instances of the available services that can be used in a composition (i.e., to build a delegator). Figure 2 shows two examples of delegators, namely  $P_{C1}$  and  $P_{C2}$ , that use several instances of available services to respectively compose target protocols. More precisely, the delegator  $P_{C1}$  uses two instances of the protocol  $P_1$ , namely  $P_1^1$  and  $P_1^2$ , to compose the target protocol  $P_{T1}$ . The delegator  $P_{C2}$  uses however (may be infinitely) many instances of the protocols  $P_1$  and  $P_2$  to compose the protocol  $P_{T2}$ . Indeed, each execution of the loop  $a.d.b$  (respectively,  $c.b$ ) of the target protocol  $P_{T2}$  is realized by two new instances of the available protocol  $P_1$  (respectively, one new instance of  $P_2$ ). Hence, the number of instances of  $P_1$ , respectively  $P_2$ , that can be used to compose  $P_{T2}$  is unbounded and hence cannot be fixed *a priori*.

We provide below a definition of a generic protocol synthesis problem that makes explicit the number of instances of protocols allowed in a composition. Let  $\mathcal{R}$  be a repository of services protocols, i.e.,  $\mathcal{R} = \{P_i, i \in [1, n]\}$ , where each  $P_i = \langle \Sigma_i, S_i, F_i, s_i^0, \delta_i \rangle$  is a protocol. For each  $P_i \in \mathcal{R}$ , we denote by  $P_i^j$  the  $j^{th}$  instance of the protocol  $P_i$ . Given a protocol repository  $\mathcal{R}$ , we note by  $\mathcal{R}^m = \bigcup_{i=1}^n \{P_i^1, \dots, P_i^m\}$ , with  $m \in \mathbb{N}$ .

**Definition 1. generic protocol composition problem** Let  $\mathcal{R}$  be a set of available service protocols and  $P_T$  be a target protocol and let  $k \in \mathbb{N}$ . A (generic) protocol synthesis problem, noted  $Compose(\mathcal{R}, S_T, k)$  is the problem of deciding whether there exist a composition of  $P_T$  using  $\mathcal{R}^k$ .

Note that, instances of this generic composition problem are characterized by the maximal number of instances of component protocols that are allowed to be used in a given composition. We distinguish in the following between two main cases, namely the bounded instances and the unbounded instances ones.

**Protocol Synthesis Problem: The Bounded Case.** Existing works [4,5,6] that investigated the protocol synthesis problem make the simplifying assumption that  $k$ , the number of instances of a service that can be involved in the composition of a target service is bounded and fixed *a priori*, i.e., they address the problem  $Compose(\mathcal{R}, S_T, k)$ . Note that this particular case, called the *bounded instance protocol synthesis problem*, can be reduced w.l.o.g to the simplest case where  $k = 1$ . Indeed, if  $k > 1$  the problem  $Compose(\mathcal{R}, S_T, k)$  can be straightforwardly reduced to the problem  $Compose(\mathcal{R}^k, S_T, 1)$ . The following proposition gives a formalization of the bounded protocol synthesis problem using the  $k$ -iterated product operator.

**Proposition 1.** *Let  $Compose(\mathcal{R}, S_T, k)$  be a protocol synthesis problem with  $k$  a finite positive integer. The problem  $Compose(\mathcal{R}, S_T, k)$  has a solution iff  $S_T \preceq \odot(\mathcal{R}^k)$ .*

The work of [4] shows that the problem  $Compose(\mathcal{R}, S_T, 1)$  can be reduced to that of testing the satisfiability of a suitable formula of Deterministic Propositional Dynamic Logic (DPDL). In [5], the PDL-based framework proposed in [4] is extended to deal with a more expressive protocol model. Interestingly, in [6] the protocol synthesis problem is reduced to the problem of testing a simulation relation between the target protocol and the product of the existing protocols. Using such a reduction, [6] shows the EXPTIME completeness of the bounded instances protocol synthesis problem.

It is worth noting that the setting of bounded instances is very restrictive in the sense that some simple protocol synthesis problems, in which the solution may use an unbounded number of instances of component protocols, cannot be solved. As an example, the rather simple composition problem depicted at figure 2, and which consists in the synthesis of the target protocol  $P_{T1}$  using the available protocols  $P_1$  and  $P_2$ , cannot be solved by the current state of the art approaches although a solution (i.e., the delegator  $P_{C2}$ ) is not complex to construct. These strong limitations motivated our work on the unbounded instance case of the protocol synthesis problem.

**Protocol Synthesis Problem: The Unbounded Case.** In the remainder of this paper we study the protocol synthesis problem in the case where the number of protocol instances that can be used in a composition are not bounded *a priori* (i.e., the problem  $Compose(\mathcal{R}, S_T, +\infty)$ ). In other words, given a repository  $\mathcal{R} = \{P_1, \dots, P_n\}$  of service protocols, we consider the generation of new composite protocols that can be obtained by an asynchronous product of any subset of protocols in  $\mathcal{R}^{+\infty}$ . More precisely, we consider the decision problem underlying the general protocol synthesis problem, i.e., the problem  $Compose(\mathcal{R}, S_T, +\infty)$ .

*Problem 1.* Let  $\mathcal{R}$  and  $S_T$  defined as previously. Is the problem  $Compose(\mathcal{R}, S_T, +\infty)$  decidable?

One way to answer this open question is to consider the related 'simulation relation' decision problem. Indeed,  $Compose(\mathcal{R}, S_T, +\infty)$  has a solution if  $S_T$  is simulated by a product of any subset elements of  $\mathcal{R}^{+\infty}$  (i.e.,  $S_T \preceq \odot(\mathcal{R}^{+\infty})$ ). Such a characterization of solutions can also be expressed using the product closure operator as stated below.

**Theorem 1.** *The problem  $Compose(\mathcal{R}, S_T, +\infty)$  has a solution iff  $S_T \preceq \odot(\mathcal{R}^{+\infty})$  (or equivalently,  $S_T \preceq (\odot(\mathcal{R}))^{\otimes}$ ).*

The main difficulty here comes from the fact that a product closure of an FSM is not an FSM. We shall prove in next section that checking simulation between an FSM  $M$  and a product closure of  $M$  (i.e.,  $M^{\otimes}$ ) is decidable. This enables to derive the decidability of the protocol synthesis problem.

## 4 Decidability Problem and Composition Algorithm

In this section we are interested by the problem of testing the existence of a simulation relation between an FSM and a product closure of an FSM. To investigate this problem, we need first to define a suitable state machine model that enables to describe a product closure of an FSM. Various state machine-based representations may be suitable to tackle our problem such as, for example, shuffle automata, introduced in [12] to recognize the so-called shuffle languages, or Petri Nets. However, as we deal only with a specific form of shuffle automata, i.e., automaton of the form  $M^{\otimes}$  where  $M$  is an FSM, we use in our work a simpler tool. We introduce below a state machine, a *PCSM* (Product Closure State Machine), that enables to describe the product closure of an FSM. More precisely, a PCSM is an infinite state machine that describes: (i) all possible executions of a product closure of an FSM, and (ii) the branching choices at each state of the execution of such an state machine. It should be noted that PCSMs are a particular form of the so-called Vector Addition Systems (VAD) [14], which are nothing other than a variant mathematical notation of Petri Nets. The PCSM notation turned out to be more convenient to handle proofs and complexity analysis in our context.

Informally, the product closure  $M^{\otimes}$  enables to run an infinite number of parallel instances of  $M$ . A product closure  $M^{\otimes}$  may then be described by an FSM similar to  $M$  with an unbounded stack of tokens in each state. The tokens number of a stack describes the number of parallel instances having reached that state. Let  $w \in \Sigma_M^*$  the input of  $M^{\otimes}$ , a symbol  $a \in w$  is recognized by the execution of such a state machine in two cases : (i) creation of a new instance of  $M$ : if there is an outgoing transition labeled  $a$  from the initial state of  $M$  to a state  $q$ . Upon such a transition, a token is added to  $q$ , or (ii) moving an existing instance of  $M$ : if there exists two states  $q$  and  $q'$  such that  $(q, a, q') \in \delta_M$  and  $q$  has one or more tokens, then upon this transition, a token is moved from  $q$  to  $q'$ .

Unlike finite state machines, where the *instantaneous description (ID)* of a given state machine is given by its current state, an ID of a PCSM involves the set of its states as well as the number of tokens in each state (number of instances having reached that state when recognizing a word). We introduce below the notion of configuration that enables to capture an ID of a PCSM. Let  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$  be an FSM and let  $|I_s(M)| = l$  and  $|H_s(M)| = n$  be respectively the set of intermediate and hybrid states of  $M$ . We assume states of  $I_s(M)$  (respectively,  $H_s(M)$ ) ordered according to the lexicographical order and relabeled accordingly with integers from 1 to  $l$  (respectively, from  $l + 1$  to  $l + n$ ). The configurations of  $M^\otimes$  are formally defined below.

**Definition 2. (Configuration)** A configuration  $C$  of the product closure  $M^\otimes$  is a tuple of size  $l + n$  of positive integers. The  $i^{th}$  element of  $C$ , written  $C[i]$ , denotes the number of tokens (i.e., instance of  $M$ ) that are at state  $i$ . We say that  $C[i]$  is the witness of the state  $i$  in a configuration  $C$ . Note that, if  $i \leq l$  (respectively,  $i > l$ ) then  $C[i]$  is a witness of an intermediate state (respectively, an hybrid state).

A configuration  $C$  is an initial (respectively, final) configuration of  $M^\otimes$  iff  $C[i] = 0, \forall i \in [1, l + n]$  (respectively, iff  $C[i] = 0, \forall i \in [1, l]$ ).

Note that, a configuration keeps only the information about intermediate and hybrid states. Indeed, it is useless to store information about the number of tokens (i.e., instances of  $M$ ) that are in final, not hybrid, states since such instances can no longer contribute to the realization of the target service. In the same spirit, as the number of instance of  $M$  that can be created is infinite (i.e., the set of tokens in the initial state is infinite) we do not describe the initial state in a configuration unless it is also an intermediate state.

Continuing with the example of figure 3, the FSM  $M$  contains only one intermediate state (state  $q_1$ ) and one hybrid state (state  $q_2$ ). Hence, a configuration associated with  $M^\otimes$  is a pair of integers where the first (respectively, the second) integer is the witness of the state  $q_1$  (respectively,  $q_2$ ). For instance, a configuration  $C = (2, 3)$  indicates an instantaneous description of  $M^\otimes$  in which there are two instances of  $M$  at state  $q_1$  and three instances at state  $q_2$ .

Using the notion of configuration, we formally define below PCSMs.

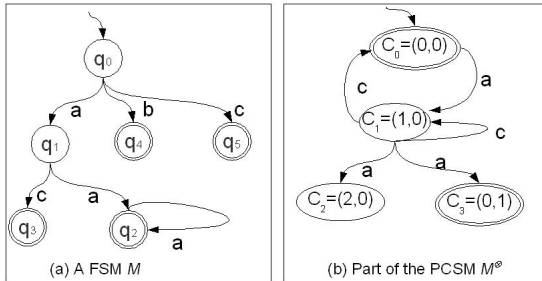


Fig. 3. An FSM and its corresponding PCSM

**Definition 3. (PCSM)** Let  $M = \langle \Sigma_M, S_M, F_M, q_M^0, \delta_M \rangle$  be a FSM with  $|I_s(M)| = l$  and  $|H_s(M)| = n$ . The associated PCSM of  $M$  is an infinite state machine  $M^\otimes = \langle \Sigma_M, \mathcal{C}, F_C, C_0, \phi \rangle$ , where:

- $\mathcal{C}$  is an (infinite) set of states consisting of all the configurations of  $M^\otimes$ ,
- $F_C$  is the set of final configurations of  $M^\otimes$ , i.e.,  $\{C \in \mathcal{C} \mid C[i] = 0, \forall i \in [1, l]\}$ ,
- $C_0$  is the initial state of  $M^\otimes$  and corresponds to the initial configuration, i.e.,  $C_0[i] = 0, \forall i \in [1, l+n]$ ,
- $\phi \subseteq \mathcal{C} \times \Sigma_M \times \mathcal{C}$  is an infinite set of transitions. The set  $\phi$  is built as follows. Let  $C_1$  and  $C_2$  be two configurations in  $\mathcal{C}$ . We have  $(C_1, a, C_2) \in \phi$  iff  $(q, a, q') \in \delta_M$  and one of the following conditions holds:
  - $q = q_M^0$  and  $q' \in (F_M \setminus H_s(M))$  with  $C_1[i] = C_2[i], \forall i \in [1, l+n]$ , or
  - $q = q_M^0$  and  $q' \in (I_s(M) \cup H_s(M))$  with  $C_2[q'] = C_1[q'] + 1, C_1[i] = C_2[i], \forall i \in [1, l+n]$  and  $i \neq q'$ , or
  - $\{q, q'\} \subseteq (I_s(M) \cup H_s(M))$  with  $C_1[q] > 0, C_2[q] = C_1[q] - 1, C_2[q'] = C_1[q'] + 1, C_1[i] = C_2[i], \forall i \in [1, l+n]$  and  $i \notin \{q, q'\}$ , or
  - $q \in (I_s(M) \cup H_s(M))$  and  $q' \in (F_M \setminus H_s(M))$  with  $C_2[q] = C_1[q] - 1, C_1[i] = C_2[i], \forall i \in [1, l+n]$  and  $i \neq q$ .

Figure 3(b) describes a part of  $M^\otimes$ , the PCSM of the FSM  $M$  depicted at figure 3(a). As mentioned before, configurations of  $M^\otimes$  are pairs  $(i, j)$  where  $i$  (respectively,  $j$ ) is the witness of the state  $q_1$  (respectively,  $q_2$ ). The infinite state machine  $M^\otimes$  is initially in the configuration  $C_0 = (0, 0)$  then it can, for example, execute the activity  $a$ , upon which it moves to the configuration  $C_1 = (1, 0)$ . At this stage,  $M^\otimes$  has two possibilities to execute the activity  $c$ : (i) by moving the current instance of  $M$  that is at state  $q_1$  into the final state  $q_3$ , or (ii) by creating a new instance of  $M$  and moving it from state  $q_0$  into the final state  $q_5$ . Note that, as the final states  $q_3$  and  $q_5$  are not described in configurations, case (i) make the  $M^\otimes$  moving back to the configuration  $C_0$  while case (ii) makes it looping on configuration  $C_1$ .

**Simulation Decision Problem.** We provide below the main result of this section.

*Problem 2.* Let  $A$  and  $M$  be two FSMs. Is it decidable whether  $A \preceq M^\otimes$  or, equivalently, is decidable whether  $q_A^0 \preceq C_0$ ?

This section answers positively to this problem by providing a sound and complete algorithm that checks the existence of a simulation relation between a FSM and a product closure of a FSM.

Note that, the main difficulty to devise our algorithm comes from the fact that we have to check the existence of a simulation relation between an FSM and a PCSM, this latter one being an infinite state machine. The corner stone of our proof is to show that to check the existence of such a simulation relation we need only to explore a finite part of the corresponding PCSM. We propose an algorithm made of three main procedures : Check-Sim, Check-Candidate and Check-Cover. When checking the simulation between a given state  $q$  and a configuration  $C$ , the Check-Sim procedure will recursively generate new simulation tests by making calls to the Check-Candidate procedure for each transition  $(q, a, q')$  in  $A$ . This latter procedure enables to check if the state  $q'$  is simulated

by at least one configuration  $C'$  such that  $(C, a, C')$  is in  $M^\otimes$ . Informally speaking, the execution of the algorithm can be seen as a tree where the nodes are labeled with pairs  $(q, C)$  and correspond to the calls of the Check-Sim algorithm. As an example, figure 4(b) shows an execution of a Check-Sim between the initial state  $q_1$  of the FSM of figure 4(a) and the initial configuration  $C_0 = (0, 0)$  of the product closure of the FSM of figure 3(a).

A crucial question is then to ensure that the algorithm terminates. Observe that for each state  $q'$ , the number of candidates  $C'$  generated by the Check-Candidate procedure is linear in the size of  $M$  since for any configuration  $C$  of a PCSM  $M^\otimes$ , the number of outgoing transitions is finite and bounded by the total number of transitions in  $M$ . Therefore, to ensure termination of the algorithm it remains to show that there are no infinite branches in the execution tree of the algorithm. In the simple case where  $A$  is a loop-free FSM, it is easy to see that the corresponding execution tree of the algorithm is finite since the length of the branches are bounded by the size of the maximal path in  $A$ . For the general case, a state  $q$  belonging to a loop in  $A$  may appear an unbounded many times in a branch of the execution tree of the algorithm. Such a case is illustrated on the figure 4(b) where the branch depicted in bold involves many times the state  $q_1$  which belongs to the loop  $(ab)^*$  of the FSM  $A$ . An important technical contribution of this work is to provide necessary and sufficient conditions that enable to cut such infinite branches. This is achieved by the second terminating condition of the Check-Sim (i.e., the call to the Check-Cover procedure) which is based on the following property: if a state  $q$  appears infinitely many times in a given branch then there is necessarily a sub-path in this branch from a node  $(q, C)$  to a node  $(q, C')$  such that  $C'$  is a cover of  $C$ . Interestingly, this condition characterizes the cases where a loop in  $A$  is simulated by  $M^\otimes$ . Continuing with the example of figure 4(b), the bold branch which is potentially infinite is cut at node  $(q_1, (0, 1))$  since the configuration  $(0, 1)$  is a cover of the configuration  $(0, 0)$  which appear previously in a node  $(q_1, (0, 0))$  in the same branch. Note that, to verify such a condition, the Check-Cover procedure maintains for each state  $q$  in a given branch a list, noted  $L(q)$ , of all the configurations  $C'$  corresponding to the nodes  $(q, C')$  of this branch. In our example, we have at node  $(q_1, (0, 1))$  of the bold branch the sequence  $L(q_1) = [(0, 0), (1, 0)]$ .

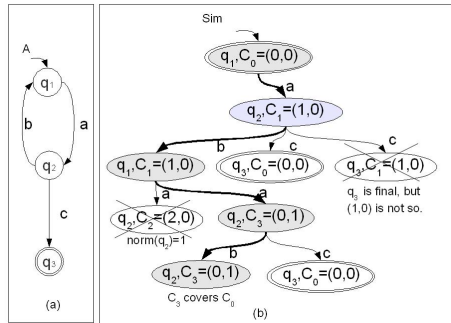


Fig. 4. Simulation of an FSM by a PCSM



---

**Algorithm 1.** Check-Sim

---

**Input.** Two FSM  $A$  and  $M$ , a state  $q$  of  $A$ , a configuration  $C$  of  $M^\otimes$ **Output.** boolean**begin**

```

  if  $q \in F_A \setminus H_s(A)$  then
    return( $\sum_{i=1}^{|I_s(M)|} C[i] = 0$ );
  if  $Check-Cover(q, C)$  then
    return(true);
  for each transition  $(q, a, q')$  in  $\delta_A$  do
    if not( $Check-Candidate(q', C, a)$ ) then
      return(false);
  return(true);

```

**end**

---

The correctness of the algorithm Check-Sim is stated in the following theorem.

**Theorem 2.** *The algorithm Check-Sim halts and is sound and complete.*

The proof of this theorem, omitted here for lack of space, is available in [16]. It is worth noting that the proposed proof is constructive in the sense that if the answer is true, the algorithm may be easily modified to exhibit a simulation relation between its inputs. This is an interesting point in the context of the protocol synthesis problem since such a simulation relation can be effectively used to build a delegator.

---

**Algorithm 2.** Check-Candidate

---

**Input.** a state  $q'$  of  $A$ , a configuration  $C$  of  $M^\otimes$ ,  $a \in \Sigma_M$ **Output.** boolean**begin**

```

  Candidates= $\emptyset$ ;
  for each transition  $(C, a, C')$  in  $\phi$  do
    if  $\sum_{i=1}^{|I_s(M)|} C'[i] \leq norme(q')$  then
      Candidates= Candidates  $\cup \{C'\}$ ;
  flag=0;
  while Candidates $\neq \emptyset$  and (not flag) do
     $C'$  =first element in Candidates;
    flag= Check-Sim( $q', C'$ );
  return(flag);

```

**end**

---

**Theorem 3.** *Let  $A$  and  $M$  be two FSMs. It is decidable whether  $A \preceq M^\otimes$ .*

Finally, in the following corollary, we derive the main result of this work regarding the addressed web service composition problem.<sup>4</sup>

**Algorithm 3.** Check-Cover

---

```

Input. a state  $q$  of  $A$ , a configuration  $C$  of  $M^\otimes$ 
Output. boolean
begin
  for  $C' \in L(q)$  do
    if  $C' \triangleleft C$  then
       $\perp$  return(true);
     $\perp$  return(false);
end

```

---

**Table 1.** Description of the test sets

Test ID	#S	#M	#H	#L	Number of variants	Total number of generated tests
<b>Test 1</b>	200	2, 4, 8 .. 4096	c	c	12	12000
<b>Test 2</b>	10, 25, 50, 100, 200	2, 4, 8 .. 4096	c	c	60	60000
<b>Test 3</b>	100	10	from 0 to 4	c	5	5000
<b>Test 4</b>	100	10	c	0, 1, 2, 3	4	4000

**Corollary 1.** *Let  $\mathcal{R}$  and  $S_T$  defined as previously. The problem  $\text{Compose}(\mathcal{R}, S_T, +\infty)$  is decidable.*

## 5 Experimental Evaluation

We implemented our algorithm as part of ServiceMosaic<sup>2</sup>, a model-driven prototype Caise tool for modeling, analyzing, and managing web services. We developed two main components: (i) **WS-protocol-generator** that enables to generate synthetic web service protocols according to several input parameters, such as the number of transitions per services, number of services, etc, and (ii) **WS-protocol-composer** an implementation of our composition algorithm. These components have been implemented using the JavaTM platform version 6 and the Eclipse framework where they are deployed as plug-ins.

**Evaluation Goals.** We can observe that the time complexity of our composition algorithm depends on the size of the execution tree of the algorithm **Check-sim**. The sizes of such a tree vary depending on two main parameters: the degrees of the nodes (i.e., the number of childrens of a given node) and the depth of the tree (i.e. the sizes of the paths between the root and the leaves).

To better understand this issue, we focused our first experiments on the analysis of the impact of the following parameters on the execution time of the algorithm:

<sup>2</sup> <http://servicemosaic.isima.fr>

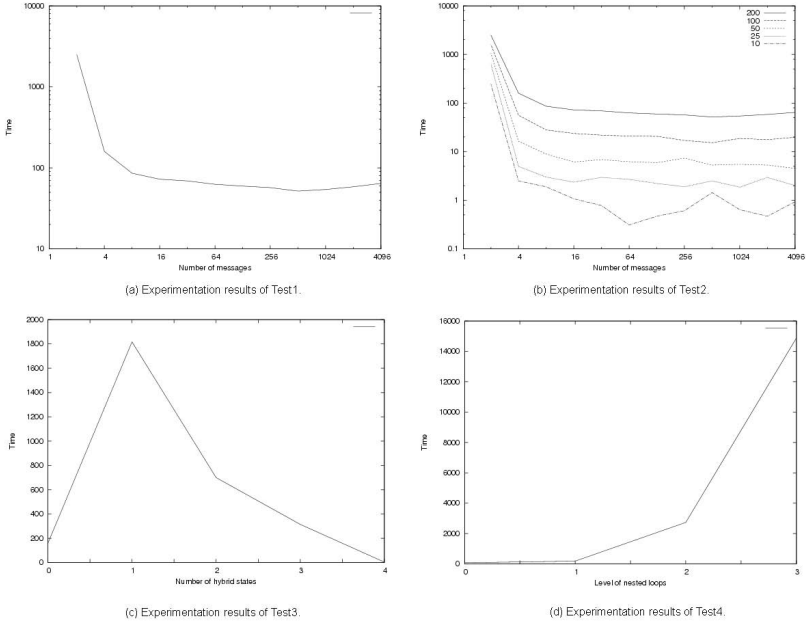
- Number of services in the service repository, noted  $\#S$ ,
- Total number of distinct message labels that appear in the services of the repository, noted  $\#M$ ,
- Number of hybrid states in each service in the repository, noted  $\#H$ ,
- Level of nested loop in each service in the repository, noted  $\#L$ .

Indeed, the degree of a node depends on the number of candidates computed by the procedure **Check-Candidate**. It corresponds to the number of transitions labeled by the same message in the services of the repository. Note that the degree does not depend on the number of active instances of a service, since using any of them leads to the same configuration. To increase the node degree one can either increase the number of services (i.e., the value of  $\#S$ ) or decrease the number of message labels (i.e., the value of  $\#M$ ). Secondly, The depth and the density of the tree depends on the presence of loops. Indeed, our proof was based on the Dickson lemma [15] which ensures the finiteness of the **Check-sim** procedure when hybrid states are present. This motivates the use of the parameters  $\#H$  and  $\#L$  in our tests. The case where only intermediate states are considered, the depth of the execution tree is bounded by a factorial function. As will be seen below, this theoretical deduction was confirmed by the results of our experimental tests.

**Building the Test Sets.** To achieve the aforementioned goals, we constructed 4 test sets each of which focusing on the study of some specific parameters among the ones mentioned above. Each test set describes the main features of the studied composition problem. The description of the test sets, summarized at table 1, as well as the results of the experimental evaluation are presented in the remainder of this section. The experiments have been achieved on Xeon double process HT 3GHz and 2GO of RAM. In the presented results, the execution times are given in milliseconds.

*Test 1.* This test set enables to assess the impact of the number of the distinct message labels that appear in the available services. For this test set (first line of the table 1, we defined a first variant with a target service and a repository of 200 available services taking their message from an alphabet of 4096 distinct labels. Then starting with this first variant, we generate other variants by relabeling at each step the messages in order to reduce the total number of distinct labels by magnitude of 2. The total number of variants is then equal to 12 (i.e.,  $\#M = 4096$ ,  $\#M = 2048$ ,  $\#M = 1024$ ,  $\dots$ ,  $\#M = 2048$ ). Note that, the occurrence of symbol  $c$  in the table 1 indicates a constant value, generated randomly, and used for the different variants of the test set.

For each of the variant of **Test 1**, we generated and runned 1000 instances. The result is reported on figure 5(a). Each point of the given curve denotes the average execution time of the 1000 instances of the corresponding variant. Observe that when  $\#M$  decreases below a given threshold, namely 8 in the figure, this leads to an exponential blow up in the execution time while above this threshold, the values of the parameter  $\#M$  seem to have less impact on the performance of the algorithm.



**Fig. 5.** Experimental evaluation results

*Test 2.* In addition to the number of messages labels ( $\#M$ ), this test set enables to assess the impact of the number of services available in the service repository ( $\#S$ ). We considered five variants of this test set obtained by varying the value of the parameter  $\#S$  (respectively, 10, 25, 50, 100 and 200). As previously, for each value of  $\#S$ , we define several variants for different values of  $\#M$  (ranging from 4096 to 2). We generated and executed 1000 instances of each variant (i.e., a total number of 60000 tests). The average execution time of each variant is reported on figure 5(b). Unsurprisingly, the results show that number of available services to explore during the composition process impacts the global performance of the algorithm. Moreover, this test set confirms the trend observed previously regarding the impact of the number of the distinct message labels on the performance.

*Test 3.* Test 3 studies the impact of the number  $\#H$  of hybrid states (respectively, the level  $\#L$  of nesting) in the available protocols. As previously, we generated a first variant of Test 3 with  $\#S = 100$  and  $\#M = 10$  and  $\#H = 0$  (i.e., no hybrid state). Then, we generate other variants by modifying the first one by increasing the number of hybrid states (from 0 to 4). Therefore, we obtain a total number of 5 variants. We generate and executed 1000 instances of each variant. The results are depicted at figure 5(c).

Interestingly, we can observe two main phases in the results depicted on this figure. In the first phase (from  $\#H = 0$  to  $\#H = 1$ ), the augmentation of

the number of hybrid states leads to a proportional increase of the execution time while we observe the converse behaviour in the second phase (i.e., when  $\#H > 1$ , the execution time decreases while  $\#H$  increases). In fact, above a given threshold, adding hybrid states increases the number of accepting states making the complete conversation (i.e., accepted words) shorter.

*Test 4.* *Test 4* studies the impact of the level  $\#L$  of nested loops in the available protocols. For this test set, we generate 4 variants with a fixed set of 100 services and 10 message labels. We distinguish 4 variants with respect to the values of  $\#L$  (ranging from 0 to 3). We executed 1000 instances of each variant and reported the average execution time in figure 5(d). As it can be expected, it turned out that the level of nesting leads to an exponential blow up in the performance of the algorithm.

## 6 Conclusion

We have studied the web service protocol synthesis problem in the general case where the number of protocol instances that can be used in a composition is unbounded. We made a reduction of this problem to that of checking simulation between a FSM and a product closure of a FSM. To cope with this later problem, we first proposed PCAs as a suitable tool for describing the behavior of a product closure of an FSM and built upon this formal framework to prove the decidability of checking the simulation relation between a FSM and a PCA.

Our preliminary experimental results show that not only the total number of services in a repository, but also other parameters, such as the number of message labels or the level of the nested loops, may influence heavily the performance of the algorithm.

As a perspective of this work, we point out several interesting issues: (i) the algorithmic issues related to the optimization of the proposed algorithm as well as the development of suitable implementation strategies, (ii) complexity, by identifying particular cases that either reduce the complexity of the problem or can be solved using classical simulation algorithms, and (iii) extension of our technique to more expressive models that enable for example modeling message exchanges and impacts on the real world such as the *Colombo* model 5.

## References

1. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: *Web Services - Concepts, Architectures and Applications*. Springer, Heidelberg (2004)
2. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: *WWW 2003*. ACM, New York (2003)
3. Dustdar, S., Schreiner, W.: A survey on web services composition. *International Journal of Web and Grid Services* 1(1), 1–30 (2005)
4. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic service composition based on behavioral descriptions. *IJCIS* 14(4), 333–376 (2005)

5. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In: VLDB, pp. 613–624 (2005)
6. Muscholl, A., Walukiewicz, I.: A lower bound on web services composition. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 274–286. Springer, Heidelberg (2007)
7. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: WWW 2002, pp. 77–88 (2002)
8. Hamadi, R., Benatallah, B.: A Petri net-based model for web service composition. In: Australasian Database Conference, pp. 191–200 (2003)
9. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
10. McIlraith, S., Son, T.: Adapting Golog for Composition of Semantic Web Services. In: KR 2002, pp. 482–493 (2002)
11. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. DKE 58(3), 327–357 (2006)
12. Jedrzejowicz, J., Szepietowski, A.: Shuffle languages are in P. TCS 250(1-2), 31–53 (2001)
13. Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. JCSS 28(3), 345–358 (1984)
14. Karp, R., Miller, R.: Parallel Program Schemata. JCSS 3(2), 147–195 (1969)
15. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with  $n$  distinct prime factors. Amer. Journal Math. 35, 413–422 (1913)
16. Ragab, R., Nourine, L., Toumani, F.: Web services composition is decidable, <http://www.isima.fr/ragab/RNTReport08.pdf>

# Design and Implementation of a Fault Tolerant Job Flow Manager Using Job Flow Patterns and Recovery Policies

Selim Kalayci<sup>1</sup>, Onyeka Ezenwoye<sup>2</sup>, Balaji Viswanathan<sup>3</sup>, Gargi Dasgupta<sup>3</sup>,  
S. Masoud Sadjadi<sup>1</sup>, and Liana Fong<sup>4</sup>

<sup>1</sup> Florida International University, Miami, FL, USA  
{skala001, sadjadi}@cs.fiu.edu

<sup>2</sup> South Dakota State University, Brookings, SD, USA  
onyeka.ezenwoye@sdstate.edu

<sup>3</sup> IBM India Research Lab, New Delhi, India  
{gdasgupt, bviswana}@in.ibm.com

<sup>4</sup> IBM Watson Research Center, Hawthorne, NY, USA  
llfong@us.ibm.com

**Abstract.** Currently, many grid applications are developed as job flows that are composed of multiple jobs. The execution of job flows requires the support of a job flow manager and a job scheduler. Due to the long running nature of job flows, the support for fault tolerance and recovery policies is especially important. This support is inherently complicated due to the sequencing and dependency of jobs within a flow, and the required coordination between workflow engines and job schedulers. In this paper, we describe the design and implementation of a job flow manager that supports fault tolerance. First, we identify and label job flow patterns within a job flow during deployment time. Next, at runtime, we introduce a proxy that intercepts and resolves faults using job flow patterns and their corresponding fault-recovery policies. Our design has the advantages of separation of the job flow and fault handling logic, requiring no manipulation at the modeling time, and providing flexibility with respect to fault resolution at runtime. We validate our design with a prototypical implementation based on the ActiveBPEL workflow engine and GridWay Meta-scheduler, and Montage application as the case study.

## 1 Introduction

Currently, many complex computing applications are being developed as job flows that are composed of multiple lower-function jobs. The execution of these job flows requires functional support of a job flow manager and a job scheduler. Due to the typical long running nature of job flows, the support for fault tolerance and recovery policies is especially important, and requires coordination between workflow engines and job schedulers. Very often, the failure of a job within a flow cannot be treated in isolation and recovery actions may need to be applied to preceding and dependent jobs as well. The interaction of multi-layered grid services with dynamic distributed resources makes fault-tolerance a critical and challenging aspect of job flow management. In this paper, we address fault tolerant issues at runtime using recurrent job flow patterns, fault tolerant patterns, and a transparent proxy.

Many exemplary job flows can be found in grid and cluster computing environments, such as the Montage application [1], workflows<sup>1</sup> in e-Science [2], and many commercial job flows using Z/OS JCL [3]. Execution of job flows requires functional support of a job flow manager and a job scheduler, as either two components of an integrated software or two separate software components. One approach to handle flow-level compensation is to include failure management logic during the development of job flow model. This approach adds the complexity of fault recovery logic to the application flow. Wei et al. [4] investigated how to incorporate fault handling and recovery strategy for jobs at modeling time without requiring the user to embed the recovery logic in the application flows by using a two-staged methodology. However, their work requires modification of the original flow to incorporate fault-handling policies at modeling time. An alternate approach is to handle workflow failures at runtime, without explicit changes to workflow process logic. In TRAP/BPEL [5], an intermediate proxy traps calls from the workflow engine, and on behalf of it, implements a runtime failure handling approach for stateless web services. However, unlike stateless web services, defining recovery policies for jobs is more challenging, as different jobs may fail at different stages of execution and may require different types of recovery actions. Long-running jobs often may require elaborate cleanup phases on account of failure.

In this paper, we present the architecture of a fault-tolerant job flow manager using job flow patterns and web services. A salient feature of our architecture is that the fault-handling for job flow execution can be introduced as late as , requiring no modifications to workflows during the modeling time. Thus, our design also has the advantages of the separation of job flow and fault handling logic, and flexibility in fault resolution at runtime. Our technique first examines the workflow automatically during deployment time to identify and label common, recurrent job flow patterns based on a knowledge base that incorporates up-to-date job flow patterns [6]. Next, we introduce a proxy that *transparently* intercepts and monitors job submissions and resolves faults at runtime using user-defined recovery policies, the identified job flow patterns, and their corresponding fault-tolerant patterns. Depending on the recovery policies, recovery actions will be selected from the alternative fault-tolerant patterns during runtime. To validate our design, we implemented a job flow management system using the open-source software, including ActiveBPEL [7] workflow engine and GridWay Meta-scheduler [8]. We used the Montage application as the case study.

The rest of this paper is organized as follows. Section 2 provides a brief background on job flow and fault tolerant patterns. Section 3 provides an architectural overview of our job flow management system. Section 4 further elaborates on the job flow management design. Section 5 introduces our prototypical implementation. Section 6 presents our experimental results. Section 7 surveys related work. Section 8 concludes the paper with a short summary of our work and suggests some directions for future work.

## 2 Job Flow and Fault Tolerant Patterns

Workflow failures have been broadly categorized as work item failures, deadline expiry, resource unavailability, external triggers, and constraint violation [9]. Some of

---

<sup>1</sup> In this paper, the terms job flows and workflows will be used interchangeably.



these failures can be handled well by specifying recovery actions at modeling time. However, in an uncontrolled grid environment, exceptions may occur due to a variety of reasons. Handling all this at modeling time is infeasible due to the high complexity it will add to the workflow and the pre-knowledge of all different failure scenarios that can arise. Prior literature [10] characterizes workflow exceptions for web services into a set of patterns. These patterns identify the individual task that the exception is based and all other dependent tasks that need to be handled; and specify the recovery action to be undertaken. In [6], we classified grid failures into some common patterns and identified common grid fault-tolerance patterns to be applied to them.

## 2.1 Job Flow Patterns

Below, we briefly summarize the relevant abstract, reusable patterns presented in [6], which arise in a job flow management system. The patterns are related to the submission of jobs from the job flow manager to the job scheduler, the exchange of monitored information regarding job states between the entities, the staging of data required for these jobs, and their execution on the scheduler resources.

**Job Submission and Monitoring.** A job submission by the flow manager to the job scheduler involves invoking the corresponding job scheduler interfaces to perform the functions of submission of the job to the resource management layer and monitoring for any state changes. Examples of different submission patterns are synchronous job submission and asynchronous submission with polling/notification.

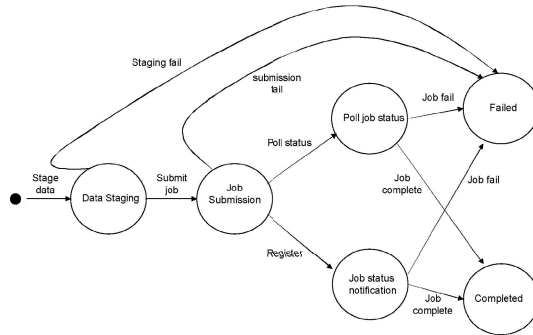
**Data Staging.** Many grid jobs require input data, and in the absence of a shared file system, these datasets need to be staged in at the site of execution. A typical data staging pattern in job flows comprises staging in data from either producer jobs or from defined inputs, followed by a job submission pattern.

**Job Execution.** Job execution completion status is captured in the job state and in the job state transitions. Some job execution failures are best handled by resubmitting the job either at the same domain or redirecting the job to a new domain. Others may require additional handling such that getting information from the job definitions.

## 2.2 Fault Tolerant Patterns

Fig. 1 shows a state transition diagram that models the patterns identified in Section 2.1. A failure in any one or more of these activities entails a transition to the *Failed* state and each such transition represents a fault-pattern. Thus, the specific fault-patterns observed due to failure at the resource management layer can be classified broadly as job submission failure, data staging failure, job execution failure, job notification failure, and job query failure. For each of these faults, we have outlined recovery actions that are generically repeatable and can be applied as fault tolerant patterns [6]. Some of the identified *fault tolerant patterns* include:

**Retry job.** A job is re-submitted for execution upon the occurrence of an exception during job submission or execution. In this pattern, jobs are submitted to the same domain. The resubmission of the job may require modifications in the job specification and resource requirements. For example, submission failures that arise from the



**Fig. 1.** Statechart diagram capturing the typical job flow patterns

temporary unavailability of the scheduling service can be recovered by re-submitting the job whereas execution errors require more detailed analysis of job state, status, and exit codes as well as a significant amount of domain expertise for fault-handling. When a job is re-submitted, the job flow manager automatically needs to re-poll or re-register for the new submission.

**Redirect job.** A job is redirected to a different domain for execution upon the occurrence of an exception during job submission or execution. This fault-tolerant pattern may be selected because all of its previous attempts of re-submitting to the same domain have failed or because it has been decided that the submission of the job to the current domain has a low probability of success. The input data of the redirected job is re-staged at the new target domain. Output data may need to be staged out.

**Retry query.** Polling for job status and/or registration for job status notification is resumed upon job re-submission. The newly submitted job is transparently re-poll (possibly using the newly returned job ID). This involves translating and modifying the original polling messages from the flow manager to map to the re-polling of the newly re-submitted job.

**Force-fail.** When no further progress is possible, the job state is changed to Failed.

### 3 Architecture Overview

In this section, we introduce the architecture of our fault-tolerant job flow manager, which is depicted in Fig. 2. This architecture incorporates two distinct advantages with respect to other related works. The first advantage is to avoid introducing the complexity of the recovery process to the application flows. Hence, the approach would maintain the separation of concerns. Please note that in Fig. 2, the components and arrows with solid line indicate those portions of the architecture related to the core business logic of the job flow and those with dashed lines relate to the fault-tolerance and recovery portion of the job flow. Secondly, the approach provides the flexibility of defining job flow patterns, fault-tolerant patterns, and recovery policies, which may be stored into the knowledge base of the system, as late as deployment

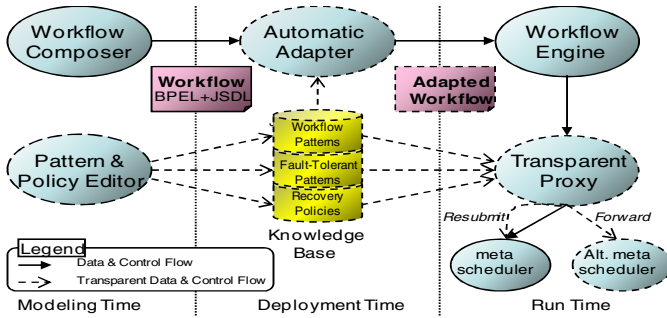


Fig. 2. The architecture of our fault-tolerant job flow manager

and runtime. The sophistication of recovery policies can grow with the knowledge of the flow and fault patterns. In addition to the above advantages, our architecture follows the two-layer design for job flow orchestration and scheduling introduced in [6]. The job flow manager is responsible for execution of the job flow according to the specified logic while being agnostic to resource allocation decisions. In contrast, the job scheduler component matches user work requests to grid resources, performs allocation, and enables distributed execution of the individual jobs in a workflow.

We note that there are many custom job flow and job definition languages used in the literature to express job flow and individual jobs. The OASIS's Web Service Business Process Execution Language (WS-BPEL or BPEL) [11] has attracted many researchers to explore for job flow specification, as there are numerous implementations (*e.g.* ActiveBPEL [7] and Websphere Process Server [12]). Thus, BPEL is used to express the flow of jobs, their dependencies, and flow of data among the jobs. A job definition describes a unit of job (*e.g.* an executable file together with parameters and resource requirements) to be submitted to a scheduler. The GGF's Job Submission Description Language (JSDL) [13] has been accepted by many researchers as the de-facto job language. Without loss of generality, our design uses BPEL and JSDL as our reference languages for job flow and job definition.

As illustrated in the left side of Fig. 2, first, a domain expert will use the *Workflow Composer* to specify the business logic of the application using BPEL+JSDL. The domain expert should only be concerned about the business logic of the application and should not be concerned about handling faults and exceptions. We note that a BPEL+JSDL workflow is still a valid BPEL. The job descriptions are treated as complex types in XML, which in turn are used as the parameters to some *Invoke* constructs in BPEL. Therefore, any BPEL editor can be used as the workflow composer. We also note that such editors may not have the capability to compose job descriptions in JSDL, but it is possible to compose job descriptions in JSDL in other editors and then copy/paste the corresponding XML document.

During deployment time, the resulting workflow is passed to the *Automatic Adapter*, which in turn automatically generates a functionally equivalent workflow with the context information that is needed for the Proxy to monitor the interaction between the flow manager and the meta-schedulers. The automatic adapter has an algorithm that identifies the known workflow patterns (*e.g.* job submission) within the workflow. The most updated workflow patterns are stored in the *Knowledge Base*.

New workflow patterns can be added to the knowledge base using the *Pattern & Policy Editor*. The generated workflow, called *adapted workflow*, would then include the context information (e.g. the *pattern* and *job id* - details to be provided in the next section), but it does not have any knowledge of how to handle faults at runtime. Instead, the adaptation incorporates some *generic* interceptors at sensitive join-points in the original BPEL+JSDL workflow. The most appropriate place to insert interception hooks in a BPEL+JSDL workflow is at the interaction join-points (i.e. at scheduler service invocation). The inserted code is in the form of standard BPEL constructs to ensure the portability of the modified process. This adaptation permits the BPEL+JSDL workflow behavior to be modified at runtime by the *Transparent Proxy*.

At runtime, the BPEL+JSDL workflow will be executed by the *Workflow Engine*. The workflow engine can be any BPEL engine without any modification or extension, as we did not extend BPEL in our work. Note that during the automatic adaptation of the workflow, all the calls originally targeted for the local *Meta-scheduler* are redirected to the *Transparent Proxy* [6]. Therefore, the Transparent Proxy will intercept all the calls to the Meta-scheduler. The Proxy will appear as a Meta-scheduler to the workflow process, and as a workflow process to the Meta-scheduler; hence, the name *transparent*. Its main responsibility includes submission of the intercepted jobs to the local Meta-scheduler and notifying the workflow process of the job status when it receives job status updates from the Meta-scheduler. In addition, it implements a pattern-matching algorithm that monitors the behavior of the intercepted calls and provides fault-tolerant behavior when faults occur. The algorithm is based on the classification of exception handling introduced in Section 2, the *Recovery Policies*, the context information embedded in the adapted workflow, the *Workflow Patterns*, and their corresponding *Fault-Tolerant Patterns*. For example, following the recovery policies governing the current faulty situation, the Transparent Proxy may resubmit the job to the same Meta-scheduler or redirect it to another Meta-scheduler.

## 4 Detailed Design

In this section we provide the detailed design of our fault-tolerant job flow manager.

**Job flow Adaptation.** Fig. 3 shows a code snippet of an example job flow that includes a job submission invocation to the underlying Meta-scheduler. Prior to the invocation construct, the JSDL definition for the job is copied to the input variable (*jobReqMsg*) of the Meta-scheduler. This invocation is replaced during the adaptation process by the Automatic Adapter with a corresponding invocation to the Transparent Proxy; thus, the invocations meant for the Meta-scheduler will be intercepted by the Proxy. Note that the input message for Meta-scheduler is now sent to the Proxy. Fig. 4 shows the same section of the code as in Fig. 3 after the adaptation process. In Fig. 4, the invocation to the Meta-scheduler is replaced with that of the Proxy. Note that invocations to the Proxy have an additional parameter, which is the job identifier (*jobID*). The Proxy uses this jobID to monitor and maintain the state of each job from inception to completion; in other words, from data-staging to completion as depicted in Fig. 1. This unique jobID helps the Proxy to keep track of the job as it progresses through each pattern.

```

<bpel:assign>
<bpel:copy>
  <bpel:from>
    <bpel:literal>
      <jSDL:JobDescription>
      ...
      <jSDL:JobDescription>
    </bpel:literal>
  </bpel:from>
  <bpel:to part="SubmitJobRequest" variable="JobReqMsg">
    <bpel:query>ns1:JSDLDocument</bpel:query>
  </bpel:to>
</bpel:copy>
<bpel:assign>
<bpel:invoke partnerLink="JobSubmitPartner"
  portType="ns1:JobManagement"
  operation="submitJob"
  inputVariable="JobReqMsg"
  outputVariable="JobResMsg" />

```

Literal copy of JSDL document

JSDL assigned to job request message

Metascheduler Invocation

**Fig. 3.** Job flow snippet code prior to adaptation

```

<bpel:assign>
...
<bpel:copy>
  <bpel:from part="SubmitJobRequest" variable="JobReqMsg"/>
  <bpel:to part="SubmitJobRequest" variable="ProxyJobReqMsg"/>
</bpel:copy>
<bpel:copy>
  <bpel:from>001</bpel:from>
  <bpel:to part="JobID" variable="ProxyJobReqMsg"/>
</bpel:copy>
<bpel:assign>
  <!-- replaces regular submit job -->
  <bpel:invoke partnerLink="ProxyJobManagementPartner"
    portType="ns1:ProxyJobManagement"
    operation="submitJob"
    inputVariable="ProxyJobReqMsg"
    outputVariable="JobResMsg"/>

```

Job request message copied to proxy job request message

Job identifier assigned to proxy job request message

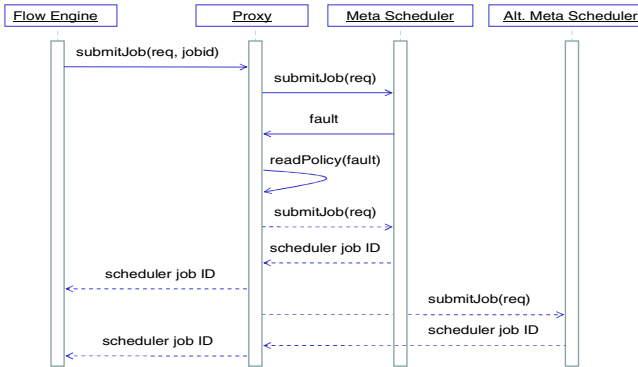
Proxy Invocation

**Fig. 4.** Job flow example snippet code after the adaptation: invocation to the Proxy

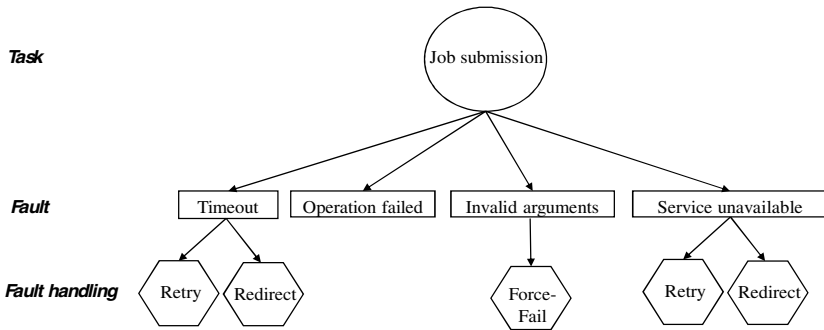
**Interface Design.** In [14], we introduced a set of APIs for meta-schedulers that supports interoperability among different meta-schedulers and demonstrated how jobs originating from a meta-scheduler can be scheduled to run on resources under the control of other meta-schedulers. In this work, we utilize this common meta-scheduler interface for communication between the Workflow Engine, Transparent Proxy, and Meta-scheduler. The job submission interface provided by the underlying Meta-scheduler layer, defines the following operations. The *submitJob* operation submits a job to the Meta-scheduler in JSDL format. If the job gets successfully submitted, the Meta-scheduler returns the corresponding job id for this job. The *getProperties* operation queries the status of a certain job by specifying its job id. The *queryJobs* operation queries the status of all submitted jobs during the current session. Certain filters for the jobs to be queried can be provided optionally. The *cancelJob* operation cancels the execution of a certain job specified by its jobID.

The Transparent Proxy interface extends the interface of the Meta-scheduler. As mentioned before, the operations of the Proxy interface require an additional parameter for job identification (*jobID*). The Proxy, acting as an intermediary between the Workflow Engine and the Meta-scheduler, requires the jobID in order to keep track of the interactions between the Workflow Engine and the Meta-scheduler; the intended pattern is implied by the invoked operation (*e.g. submitJob* operation for *job submission* pattern and *queryJobs* for *poll job status* pattern). The Proxy also maintains state information about individual jobs.

**Fault Handling and Recovery Policies.** On receipt of an invocation, the Proxy stores any necessary information and redirects the call to the local Meta-scheduler. Any reply from the Meta-scheduler is routed through the Proxy. In the event of a fault, the Proxy can enact fault-tolerance actions as defined in the recovery policy. The sequence diagram in Fig. 5 shows the interactions between the Workflow Engine, Transparent Proxy, Meta-scheduler, and Alternative Meta-scheduler during a job submission.



**Fig. 5.** The interactions between Workflow Engine, Transparent Proxy, Meta-scheduler, and Alternate Meta-scheduler during job submission



**Fig. 6.** A graphical representation of an example recovery policy for job submission

As shown in the figure, the Proxy redirects a job submission call to the Meta-scheduler and initiates a timer to measure the time of the invocation; the duration of the timer is defined in the recovery policy for this particular invocation. In the event of a fault (or timeout), the Proxy will either retry the job submission to the same Meta-scheduler or redirect the job to an alternative Meta-scheduler. The Proxy consults the recovery policy to determine what action to take. The number of retries, length of retry interval, and maximum number of attempts to redirect the job may be defined in the recovery policy.

Fig. 6 shows a graphical model of an example recovery policy where a set of possible faults are associated with the job submission task. Recovery actions are defined for those specified fault events. For example, a retry of job submission is specified for a timeout fault. If the retries are exhausted for the same fault, the Proxy is then required to redirect the job submission to an alternative Meta-scheduler. A failed operation follows its normal fault handling process in the workflow and the Proxy does not get involved in this situation. A Force-Fail (section 2.2) is enacted for an “invalid arguments” fault and a Retry/Redirect operation is required if the desired Meta-scheduler is unavailable. Similar recovery policy specifications can be specified for

the data-staging and job-status-polling stages of the job. This policy model can be easily represented in XML within the policy document. The advantages of this model are: (1) simplification of representation; (2) ease of extension; and (3) flexibility since recovery actions do not have to be hardcoded in the Proxy.

## 5 Prototypical Implementation

This section presents the details of our fault-tolerant job flow management prototype setup at Florida International University (FIU) that consists of a job flow manager, a Proxy and a scheduler component. For building this testbed, we used the ActiveBPEL flow engine (v4.1), and the Meta-scheduler built based on the Globus Toolkit (GT4) [15], and GridWay Meta-scheduler (v5.2.3). We used the DRMAA [16] API for job submission, monitoring, and controlling.

To detect faults during job submission, job monitoring, or job execution, the Proxy maintains an internal state machine and several timers for each job. The state machine at the Proxy is same as the one shown in Fig. 1. On fault-detection, the Proxy consults its policy knowledge base and takes the necessary actions specified in the policy file. Policies, defined in XML, are generic and thus apply to all jobs. The snippet of a generic policy file used by the Proxy is shown in Fig. 7.

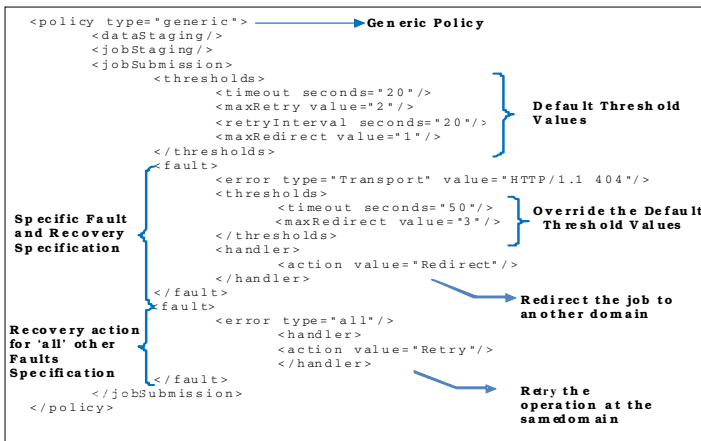


Fig. 7. A generic policy example snippet

In this policy example, a list of fault handlers can be specified (*fault* elements in Fig. 7), each capable of handling a specific fault (*error* elements). The fault handlers are matched in sequence and the first matching fault handler is applied. Fault handling is done by performing the associated sequence of recovery actions (*action* elements). A default handler which matches all faults (*error type="all"*) can apply a generic set of recovery actions for all unmatched faults.

We used the Montage application [1] for this experimentation. The application structure, as shown in Fig. 8, consists of the computational workflow of re-projection of input images (*mProject*), modeling of background radiation (*mDiffFit*), rectification of images (*mBackground*) and co-addition of re-projected, background corrected images into a final mosaic (*mAdd*). Activities like *mProject* and *mDiffFit* can run as parallel tasks. The most computationally intensive step is that of *mProject* while the most data-intensive steps are that of *mOverlaps* and *mBackground*. The inherent parallelism among jobs in its different stages makes Montage a very suitable candidate application for grid enablement.

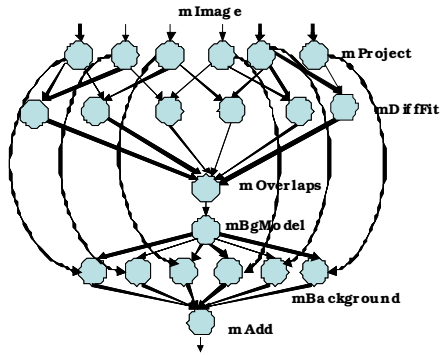


Fig. 8. Montage Application

## 6 Experimental Results

We setup our Montage application execution environment on two different platforms: the first, called GCB, is an eight node cluster, with dual P4@3GHz and 1 GB of memory per node, running GT4, Rocks, and CentOS 4.4. The second, called Skywarp, is a single node P4 with 1 GB of memory as the alternative execution environment. Both environments have the same Meta-scheduler setup and Montage executables and libraries.

### 6.1 Proxy Overhead and Opportunistic Behavior Analysis

When there is no fault to be handled by the Proxy, the Proxy intercepts all interactions between Flow Engine and the Meta-scheduler; causing a small amount of overhead during job flow execution. To calculate this performance overhead, we executed the same workflow on GCB, both with and without our fault tolerant infrastructure. Table 1 presents the average statistics from 5 separate runs without any faults during the execution. As indicated in the column with the heading “No Slowdown”, the Proxy introduces a very small overhead in execution time.

In some cases the presence of a Proxy may not be necessary for the successful completion of the workflow. However, it may provide better performance (depending on the system load at the time). One such scenario is the slowdown during the service call directed towards the Meta-scheduler, resulting in a long delay before getting back

Table 1. Average Montage workflow runtime table on GCB with/without Proxy

	No Slowdown	1 Slowdown	2 Slowdowns
No Proxy	18 min. 44 sec.	19 min. 12 sec.	19 min. 43 sec.
With Proxy	18 min. 46 sec.	19 min. 01 sec.	19 min. 14 sec.



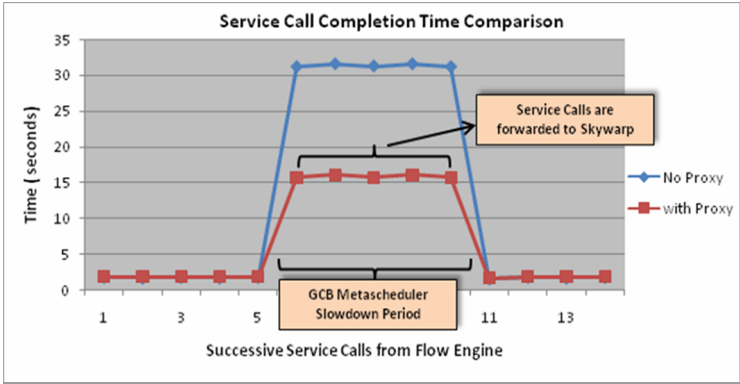


Fig. 9. Average total service calls completion time for successive calls

with a reply. In this case, the Proxy can detect the slowdown in the Meta-scheduler service and redirect the requested service to the alternative Meta-scheduler, which is idle at the time. We call this the *opportunistic behavior* of the Proxy. For the class of short-running jobs, we present results for two such slowdowns in Table 1. Each slowdown lasted for 30 seconds, while the Proxy’s call timeout value was configured at 10 seconds. The recovery policy was set to Redirect. Table 1 (columns 2, 3) demonstrates the opportunistic behavior of Proxy that results in shorter runtimes in case of slowdown/interruption in service. Figure 9 shows the series of individual request completion time with and without the Proxy.

### 6.2 Fault-Recovery Scenarios and Experimental Results

This section details the specific fault and recovery scenarios we experimented with while inducing faults in our test-bed for the Montage workflow:

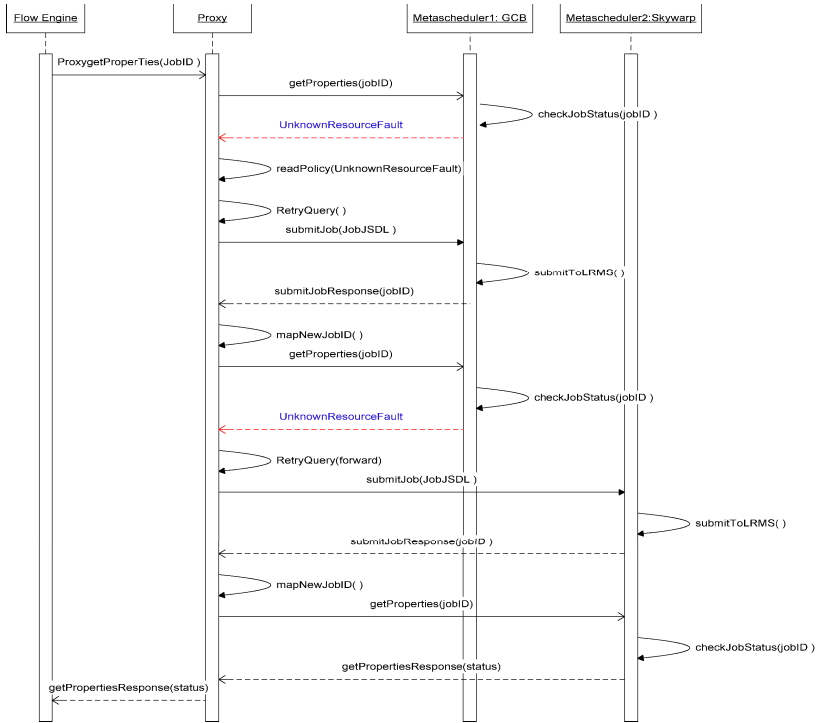
1. Job submission faults:

- a. The particular web service is not available and the service returns a transport level error (such as HTTP 503: *Service Unavailable* Message).
- b. The submit request message gets to the Job Submission Web Service, but the service internally decides to send a Service Unavailable Fault message.
- c. Timeout value that matches for the current job is exceeded.

*Proxy Action:* Retry the job request on the same domain for a maximum of  $N$  times, with  $M$  seconds interval between each retry. If after  $N$  retries job submission doesn’t succeed, try another fault-tolerant pattern (*i.e.*, Redirect).

2. Job status query faults:

- a. An “UnknownResourceFault” message is returned by the operation “get-Properties”. This may occur due to an unknown job id (*i.e.*, a job id not recognized by Meta-scheduler).



**Fig. 10.** Sequence diagram for scenario 2.a. Proxy first enacts the “Retry Query” on the same domain (GCB), then on another domain (Skywarp).

*Proxy Action:* Retry the corresponding job on the same domain. Again, the policy can specify the number of retries. If this action does not complete successfully, repeat the same action on another domain. The sequence diagram for Scenario 2.a can be seen in Fig. 10, where N (number of retries) is set to be 1.

### 3. Other Job faults :

- a. The recovery actions for 1.a, 1.b and 1.c (e.g. Resubmit Job pattern) fail. In this case the job is redirected to a new domain.
- b. A “No Service” fault occurs when a job submission request is sent to a domain, but the service does not exist due to either the service was never deployed, or it was migrated. (For e.g. HTTP 404 error: Page Not Found). In this scenario, there is no benefit in retrying the job at the same domain, and hence can be directly redirected.

*Proxy Action:* Redirect the job submission request to another domain. Policies specify the number of maximum redirecting attempts value.

Table 2 summarizes results from our experiments that simulate the above mentioned failure and recovery scenarios. Each row reports the number of patterns applied by the Proxy in different fault scenarios and the average execution times observed across multiple runs of the Montage workflow. Test 1 represents the base case with no faults. For all the other cases, the same policy file was used with the *timeout* = 20 seconds, *retryInterval* = 20 seconds and *maxRetry* = 2. In order to measure the time values in isolation from the specific service that was faulted, each enumerated fault within Test 2-4 was generated at the same point (i.e. before the same service call for the specific task) throughout the workflow.

Total execution time values for Test 3 and Test 4 are comparable as the Proxy behavior is similar for the “Service Unavailable” and “Connection Error” fault types. However, Test 2 has a higher execution time than both Test 3 and Test 4, because while the faults are reactively and immediately raised, the timeouts are passively observed. Test 5 has much higher execution time than the remaining tests, substantially because of the large number of Redirects. In this test, upon a “Service Unavailable” or “Connection Error” fault, a job redirection occurs after the Retry Job pattern fails. This means that each extra Redirect introduces at least 40 seconds ( $\text{maxRetry} * \text{retryInterval}$ ) to the total execution time.

**Table 2.** Runtime table for 5 different simulations of the Montage workflow. Number of “Retry Job”, “Redirect” and “Retry Operation” patterns applied by the Proxy is given for each service and each fault type.

	Service (Pattern) \ Fault Type	Timeout	Service Unavailable	Connection Error	No Service	Total Execution Time
Test 1	<i>SubmitJob</i> (Retry Job/Redirect)	0/0	0/0	0/0	-/0	18 min. 46 sec.
	<i>GetProperties</i> (Retry Operation/Redirect)	0/0	0/0	0/0	-/0	
Test 2	<i>SubmitJob</i> (Retry Job/Redirect)	3/1	0/0	0/0	-/0	23 min. 18 sec.
	<i>GetProperties</i> (Retry Operation/Redirect)	3/1	0/0	0/0	-/0	
Test 3	<i>SubmitJob</i> (Retry Job/Redirect)	0/0	3/1	0/0	-/0	22 min. 06 sec.
	<i>GetProperties</i> (Retry Operation/Redirect)	0/0	3/1	0/0	-/0	
Test 4	<i>SubmitJob</i> (Retry Job/Redirect)	0/0	0/0	3/1	-/0	21 min. 56 sec.
	<i>GetProperties</i> (Retry Operation/Redirect)	0/0	0/0	3/1	-/0	
Test 5	<i>SubmitJob</i> (Retry Job/Redirect)	0/0	2/1	2/1	-/1	25 min 43 sec.
	<i>GetProperties</i> (Retry Operation/Redirect)	0/0	2/1	2/1	-/1	

## 7 Related Work

Condor [17] addresses faults that occur due to job failure, communications faults and other unusual and erroneous conditions via job resubmissions and restarts. DAGMan, the workflow engine in Condor, is responsible for enforcing the dependencies between the jobs defined in the workflow. In case of job failure, DAGMan can retry a job for a given number of times, or a job flow generated as a rescue DAG can be potentially modified and re-submitted at a later time.

The approach in BPEL4JOB [4] investigated the incorporation of fault handling policies in BPEL workflows at flow model development time. It used a two staged methodology: embedding only the recovery policies in the application flow and then expanding the application flow to include the recovery processes. Unlike our approach to handle faults at execution time, this approach would require modification of the application flow and modeling tooling.

Since modeling-time fault-handling requires knowledge of all faults a-priori, an alternate approach is to handle these failures at runtime. The Pegasus project [18] provides support for just-in-time planning, which follows a lazy approach for mapping sub-flows to the currently available resources. This approach works better than static mapping in dynamic resource conditions. However, if a job fails at runtime, it blindly re-schedules the entire sub-flow without looking into the source of the problem. In [19], the authors study the impact of runtime optimizations made at the scheduler for handling workload surges, while minimizing the reconfiguration overhead. However, identification of failure causes and fault-tolerant patterns is not addressed. Prior work in [9] characterizes workflow exceptions for web services into a set of patterns that specify how the individual task on which the exception is based and all other dependent tasks should be handled and what recovery action (if any) is to be undertaken. In the workflow domain, worklets [10] have been proposed to build extensible dynamic fault-handling services for Yet Another Workflow Language (YAWL). This work presents a detailed taxonomy of exception patterns in the web services domain from which a dynamic runtime selection is made depending on the context of the exception and the particular work instance.

## 8 Conclusion and Future Work

In this paper, we proposed the approach of addressing the fault tolerant issues at deployment and runtime, in comparison to various fault recovery strategies at the modeling time. We adapt job flows at deployment time and automatically incorporate context information to be used by the Transparent Proxy, which intercepts potential faults at runtime. The Transparent Proxy searches the populated knowledge-base with recurrent job flow patterns and fault tolerant patterns, and then finds the pre-defined recovery strategies from the recovery policies to handle the fault. This approach has two distinct advantages: (1) maintaining separation of concerns between application flow and recovery strategies; and (2) flexibility of defining job flow patterns, fault patterns, and recovery strategies as late as deployment or runtime. We validated our approach with a prototypical implementation using ActiveBPEL workflow engine, GridWay Meta-scheduler and the Montage application and presented experimental

data collected on the validation system. Our current experimentation uses a selected set of job flow and fault patterns, and a limited set of recovery strategies expressed simply in a policy file. Possible future work would include exploration of additional flow and fault patterns and more sophisticated recovery strategies using semantic and data information of the job flows.

**Acknowledgement.** This work was supported in part by IBM, the National Science Foundation (grants OISE-0730065, OCI-0636031, HRD-0833093, and HRD-0317692). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF and IBM.

## References

1. Berriman, G.: Montage: A Grid enabled image mosaic service for the national virtual observatory. *Astronomical Data Analysis Software and Systems XIII* (2003)
2. Taylor, I.J., et al. (eds.): *Workflows for e-Science*. Springer, Heidelberg (2007)
3. Brown, G.D.: *Z/OS JC*, 5th edn. Wiley Publisher, Chichester (2002)
4. Tan, W., Fong, L., Bobroff, N.: BPEL4Job: a fault-handling design for job flow management. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007*. LNCS, vol. 4749, pp. 27–42. Springer, Heidelberg (2007)
5. Ezenwoye, O., Sadjadi, S.M.: TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services. In: *International Conference on Web Information Systems and Technologies (WEBIST-2007)*, Barcelona, Spain (2007)
6. Dasgupta, G., Ezenwoye, O., Fong, L., Kalayci, S., Sadjadi, S.M., Viswanathan, B.: Design of a Fault-Tolerant Job-Flow Manager for Grid Environments Using Standard Technologies, Job-Flow Patterns, and a Transparent Proxy. In: *Proceedings of 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Redwood City, CA (July 2008)
7. ActiveBPEL, <http://www.activevos.com/community-open-source.php>
8. Huedo, E., Montero, R.S., Llorente, I.M.: The GridWay Framework for Adaptive Scheduling and Execution on Grids. In: *Workshop on Adaptive Grid Middleware, Intl. Conf. Parallel Architectures and Compilation Techniques (PACT 2003)* (September 2003)
9. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow Exception Patterns. In: Dubois, E., Pohl, K. (eds.) *CAiSE 2006*. LNCS, vol. 4001. Springer, Heidelberg (2006)
10. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Dynamic and Extensible Exception Handling for Workflows: A Service-Oriented Implementation. *BPM Center Report BPM-07-03*, BPMcenter.org (2007)
11. Jordan, D., et al.: *Web Services Business Process Execution Language Version 2.0* (2007), <http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf>
12. IBM Websphere Process Server, <http://www-306.ibm.com/software/integration/wps/>
13. Anjomshoaa, A., et al.: Job Submission Description Language (JSDL) Specification v1.0. Proposed Recommendation from the JSDL Working Group (2005), <http://www.gridforum.org/documents/GFD.56.pdf>
14. Bobroff, N., Fong, L., Kalayci, S., Liu, Y., Martinez, J.C., Rodero, I., Sadjadi, S.M., Villegas, D.: Enabling Interoperability among Meta-Schedulers. In: *IEEE 8<sup>th</sup> International Symposium on Cluster Computing and the Grid (ccGrid)* (May 2008)

15. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. In: Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing, SIAM, Lyon, France (August 1996)
16. Rajic, H., et al.: Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA. Working Group - The Global Grid Forum (2003)
17. Couvares, P., et al.: Workflow Management in Condor. In: Taylor, I.J., et al. (eds.) Workflows for e-Science. Springer Press, Heidelberg (2007)
18. Deelman, E., et al.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal* 13(3), 219–237 (2005)
19. Dasgupta, G., Dasgupta, K., Viswanathan, B.: Data-WISE: Efficient management of data-intensive workloads in scheduled Grid environments. In: Proceedings of IEEE/IFIP Network Operations and Management Symposium, NOMS (2008)

# Building Mashups for the Enterprise with SABRE

Ziyan Maraïkar<sup>1,\*</sup>, Alexander Lazovik<sup>2,\*\*</sup>, and Farhad Arbab<sup>1</sup>

<sup>1</sup> Centrum voor Wiskunde en Informatica, The Netherlands  
{maraïkar, farhad}@cwi.nl

<sup>2</sup> INRIA Saclay, Parc Club Orsay Université, France  
lazovik@lri.fr

**Abstract.** The explosive popularity of mashups has given rise to a plethora of web-based tools for rapidly building mashups with minimal programming effort. In turn, this has spurred interest in using these tools to empower end-users to build *situational applications* for business. Situational applications based on Reo (SABRE) is a service composition platform that addresses service heterogeneity as a first-class concern by adopting a mashup’s data-centric approach. Built atop the Reo coordination language, SABRE provides tools to combine, filter and transform web services and data sources like RSS and ATOM feeds. Whereas other mashup platforms intermingle data transformation logic and I/O concerns, we aim to clearly separate them by formalising coordination logic within a mashup. Reo’s well-defined compositional semantics opens up the possibility of constructing a mashup’s core logic from a library of prebuilt connectors. Input/output in SABRE is handled by service stubs generated by combining a syntactic service specification such as WSDL with a constraint automaton specifying service behaviour. These stubs insulate services from misbehaving clients while protecting clients against services that do not conform to their contracts. We believe these are compelling features as mashups graduate from curiosities on the Web to situational applications for the enterprise.

## 1 Introduction

A recent trend in web applications has been the emergence of so-called *mashups*. Mashups are web applications that literally mash-up or combine disparate web services, RSS and ATOM feeds and other data sources in new and interesting ways. They compose these services in ways usually unanticipated by their original authors. Mashups are thus, ad-hoc by very nature.

More formal approaches to service-oriented computing (SOC), such as WS-BPEL[9], assume the availability of uniform service interfaces and service meta-data available in centralised registries. Yet the Utopian promise of uniform service

---

\* Supported by NWO project BRICKS-AFM3.

\*\* This work was carried out during the tenure of an ERCIM “Alain Bensoussan” Fellowship Programme.

interface standards, metadata and universal service registries, in the form of the SOAP, WSDL and UDDI, standards has proven elusive. Instead, prominent web service providers like Google, Yahoo, Amazon and eBay have opted to use lightweight protocols like RSS and ATOM to push data to consumers, while exposing their service offerings as simple REST-style[10] APIs. In turn, each of these service provider APIs have their own syntax and semantics.

A holistic approach to SOC on the Web, must therefore embrace service heterogeneity as a first-class concern, instead of relegating it to a mere implementation detail. Where mashups succeed is in adopting a data-centric approach to service composition. In a sense, they follow the UNIX tradition of gluing arbitrary programmes together using pipes and text processing tools like *awk* and *sed*, to meet any need. We believe this is a useful approach that complements the traditional control-driven coordination and orchestration exemplified by BPEL.

*Situational applications based on Reo* (SABRE) aims to have the best of both world by marrying formal SOC with the data-driven approach of mashups. SABRE's contributions are threefold. First, it structures mashups by introducing a clear separation of concerns between service coordination and input/output. Secondly, by using the Reo coordination language[2] (described in §4) we are able to graphically define not just data mangling, but also the *semantics* of service composition, that is, the synchronisation constraints we wish to impose between services. Thirdly, SABRE's automatically generated service stubs isolates faults caused either by services that do not conform to their contracts or by misbehaving clients. We put these ideas into practice by providing a rich suite of tools to graphically define a mashups' data-driven logic and service interfaces with precise formal semantics, automatically generate service stubs and a deployment environment running in an Java application container such as Apache Tomcat.

The rest of this paper is organised as follows. In §2 we review the state of the art in mashup platforms and traditional service composition. A discussion of applicability of mashup platforms for service-oriented computing, together with a motivating example appear in §3. In §4 we consider Reo as a means of formalising coordination of services within mashups. In §5 we apply constraint automata as a unifying formalism for Reo as well as a behavioural specification of services. We describe a reference implementation for the SABRE framework and its architectural issues in §6. We conclude in §7, with a summary of the paper and a discussion of our further work.

## 2 Related Work

### 2.1 Mashup Platforms

Mashups can, and are, built using traditional web scripting languages like Perl, PHP and JavaScript. Their explosive popularity has however, given rise to a number of “mashup platforms” for building mashups, with minimal programming. This trend parallels the rise of rapid application development (RAD) tools to ease the development of graphical user interfaces in the 90's.



The use of mashups as *situational applications* in enterprise environments is addressed in [11]. They use the phrase “enterprise information mashup fabric” to describe mashup platforms for the enterprise. DAMIA[1] is a concrete realisation of these ideas for building mashups on corporate intranets. Both SABRE’s tools and runtime environment have similarities to DAMIA. However, we are more concerned with enabling compositional construction of mashups with well defined semantics. In this respect the SABRE tools share a common purpose with the SENSORIA Development Environment[21] (SDE) for semantic-based development of service-oriented systems. Whereas SDE is a general-purpose SOC tool suite, SABRE focuses on exclusively on tools for data-driven service coordination.

SABRE also borrows ideas from commercial mashup platforms. Google mashup Editor<sup>1</sup>(GME) is a textual mashup tool that uses XML based mark-up, combined with HTML and optionally, JavaScript. Each user-interface element in the mashup is defined using a `module` tag that groups an input data source with a template specifying the format of the resulting output. Yahoo Pipes<sup>2</sup>, another commercial offering, takes a graphical approach to mashup creation. A variety of data sources can be plumbed through so-called *pipes* that filter and transform data. Predefined pipes are available to connect to data sources like RSS feeds, REST and SOAP web services, perform string, number, and date manipulation, and filter and sort data. Complex pipes may be composed of primitives and invoke external services. Pipes superficially resemble channels in Reo (§4) and SABRE’s editor closely resembles the Pipes Editor, but Reo and its tool suite described in §6 actually predates Yahoo Pipes.

Most mashup platforms commingle the core coordination logic of the mashup with external input/output and user interface concerns. Furthermore, none of them consider the semantics of the services being used. Consequently none support true compositional construction of a mashup’s logic. For instance, while Yahoo Pipes supports composing pipes, its notion of composition is limited to data transformation operations. Arguably, this notion of composition is sufficient for building the types of simple mashups commonly found on the Web. As mashups migrate into enterprise environments, however, more formal notions of service composition and coordination become highly desirable. SABRE aims to provide the features necessary to support this use case, without unduly burdening the mashup developer.

## 2.2 Service Coordination

Service coordination refers to managing interactions among different business processes and any atomic services that they may entail. Currently WS-BPEL[9] and WS-CDL[12] are the most widely used languages dealing with orchestration and choreography, respectively. While BPEL is a powerful standard for composition of services, it lacks support for actual coordination of services. Orchestration and choreography, have received considerable attention in the web

<sup>1</sup> <http://googlemashups.com>

<sup>2</sup> <http://pipes.yahoo.com>

services community, and separate standards (e.g., WS-CDL) are being proposed for them. However, orchestration and choreograph are in fact, different facets of coordination. Thus, it is questionable whether such fragmented solutions for various aspects of coordination, which involve incongruent models and standards for choreography and orchestration, can yield a satisfactory SOA. Most efforts up to now have been focused on statically defined coordination (compositions), as in BPEL. To the best of our knowledge the issues involved in dynamic coordination of web services with continuously changing requirements have not been seriously considered. The closest attempts consider automatic or semi-automatic service composition, service discovery, etc. However, all these approaches mainly concentrate on how to compose a service, and do not pay adequate attention to the coordination of existing services.

In SABRE we use the channel-based exogenous coordination language Reo [2] to specify mashup logic. Reo supports a specific notion of composition that enables coordinated composition of individual services as well as composed business processes. It is claimed that BPEL-like languages maintain service independence, but in practice they hard-wire services through the connections that they specify in the process itself. Reo in contrast, allows us to concentrate only on important protocol decisions and define only those restrictions that actually form the domain knowledge, leaving more freedom for process specification, choice of individual services, and their run-time execution. In traditional SOC, it is often difficult and costly to make any modification to the process, due to the complex relationships among its participants. This is a by-product of forcing a process designer to explicitly define all steps in precise execution order in the process specification, resulting from the use of essentially sequential, imperative, and process oriented languages and tools. It is extremely difficult to adapt such over-specified processes to accommodate even minor deviations in implementation. By placing interaction and its coordination at the centre of attention, Reo lifts the level of abstraction for the specification of composed processes.

Several other formalisms have also been developed for composition and coordination of distributed entities e.g., based on Petri nets and  $\pi$ -calculus, and coordination based on mobile channels[19]. However, these general frameworks do not particularly cater to certain issues in service-oriented computing, e.g., non-deterministic nature of services or late binding of service implementations.

### 3 Mashups Versus Service-Oriented Computing

The upshot of the discussion in §2 is that while mashups and traditional BPEL-style SOC have their strengths, each has many shortcomings that need to be addressed. Since SABRE attempts to bridge the gap between classical SOC and mashups, we begin by trying to identify how SOC best-practises apply to the mashup building process.

**Separation of concerns.** By virtue of being ad-hoc there is tight coupling between the data mangling logic, the utilised services and feeds, and user

interface elements that render the data in mashups. SOC regards having a clear separation between these concerns as highly desirable.

**Composition and coordination.** In the context of a mashup, composition usually boils down to ad-hoc “data mangling”; that is, filtering combining and transforming various inputs to generate desired outputs. Furthermore, a mashup’s logic should be composable from reusable blocks to facilitate quick, modular construction. No mashup platforms known to us, has any notion of service semantics. Therefore existing mashup platforms cannot support the notion of service coordination. In SOC is control-driven coordination exemplified by BPEL is the norm, but this style does not mesh well with the data-driven nature of mashups.

**Service contracts and fault isolation.** A service should have both a syntactic specification, using WSDL for example, and a behavioural specification of its semantics. Such precise service contracts shields services from misbehaving clients, while ensuring services actually adhere to their contracts. This helps isolate faults due to misbehaving clients or services. Although there is extensive research on behavioural specification, no industry standards currently exist.

**Service binding.** There is an inherent trade-off between flexibility in service binding vs. dealing with service heterogeneity. The late-binding approach advocated in SOC assumes uniform service interfaces, and universal registries. Conversely, mashups handle heterogeneity at the cost of being tightly coupled to the services they use. Ideally, we would like a limited form of late-binding at least for standard input source like RSS feeds.

For the remainder of this paper we use the “Sports-fan Dashboard” example to demonstrate mashup development in SABRE, and highlight how we achieve the the first three improvements identified above. Our example mashup shows rele-



Fig. 1. Example sports-fan mashup user interface

vant information based on the fixture schedule of a sports team. The Dashboard uses an RSS or ATOM feed containing a schedule of team fixtures. A fixture calender for Ajax FC for example, is available on Google Calendar<sup>3</sup>. Once the user chooses a match of interest the Sports-fan Dashboard display the following information: (i) a map showing the venue; (ii) news articles about the match; (iii) weather forecast for the day of the match; (iv) option to purchase tickets online, if the weather forecast is “good”<sup>4</sup>. Figure 1 shows a screenshot of the running application.

## 4 Specifying Mashup Logic with Reo

We formalise the the core logic within a mashup by encoding it in Reo. Reo is *exogenous* in that it imposes a coordination pattern on components, without any knowledge of the internals of the components and without the components having any knowledge of the coordination. This makes Reo ideal for coordinating services from a data-centric perspective. Coordination in Reo is specified by a *connector* consisting of nodes and primitives. Formally, a Reo connector is defined as follows:

**Definition 1 (Reo connector).** A connector  $\mathcal{C} = \langle \mathcal{N}, \mathcal{P}, E, node, prim, type \rangle$  consists of a set  $\mathcal{N}$  of nodes, a set  $\mathcal{P}$  of primitives, a set  $E$  of primitive ends and functions:

- $node : E \rightarrow \mathcal{N}$ , assigning a node to each primitive end,
- $prim : E \rightarrow \mathcal{P}$ , assigning a primitive to each primitive end,
- $type : E \rightarrow \{src, snk\}$ , assigning a type to each primitive end.

A *node* is where the execution of different primitives is synchronised. Data flow at a node occurs, iff (i) at least one of the attached sink ends provides data and (ii) all attached source ends are able to accept data. A node does a destructive read at one of its sink ends and replicates the data obtained to every one of its source ends.

Typically, Reo primitives are *channels*. Channels can be attached to nodes to compose connectors. The ends of a channel can be either source ends, which accept data or sink ends which produce data. The actual semantics of a channel depends on its type. Reo does not restrict the possible channels used as long as their semantics is provided. Nodes, however, have the fixed semantics defined above, which specifies their routing constraints. Table 1 describes the behaviour of some common Reo channels. The top three channels represent synchronous communication. A channel is called *synchronous* if it delays the success of the appropriate pairs of operations on its two ends such that they can succeed only simultaneously. Note that a Reo connector built from synchronous channels is stateless and its execution is instantaneous in an all-or-nothing fashion. The

<sup>3</sup> <http://www.google.com/calendar/embed?src=jdtvbcrk1vtpn5ec4ptnnfnd61c0udfppt.calendar.google.com>

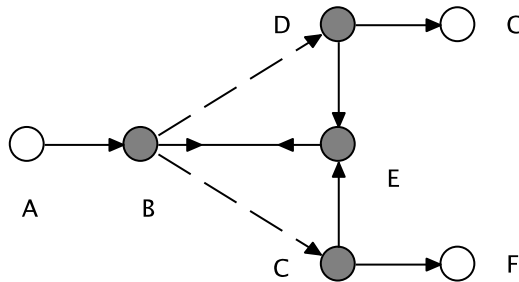
<sup>4</sup> Admittedly this was contrived to show the utility of channel composition in Reo.

**Table 1.** Behaviour of common Reo channels

<i>Sync</i>		Simultaneously accepts data on one end and passes it out the other end
<i>SyncDrain</i>		Simultaneously accepts data on both ends
<i>SyncSpout</i>		Simultaneously produces data on both ends
<i>LossySync</i>		Behaves as a <i>Sync</i> if a <i>take</i> operation is pending on the output end, otherwise the data is lost
<i>Filter</i>		Passes data matching a filter pattern and loses the rest
<i>FIFO</i>		Buffers a single data item.

*FIFO* channel enables us to add stateful behaviour to a connector. An extensive discussion of Reo, various channel types and numerous examples can be found in [2]. Formal semantics for Reo has been given using constraint automata[6], connector colouring[7] and structured operational semantics[17].

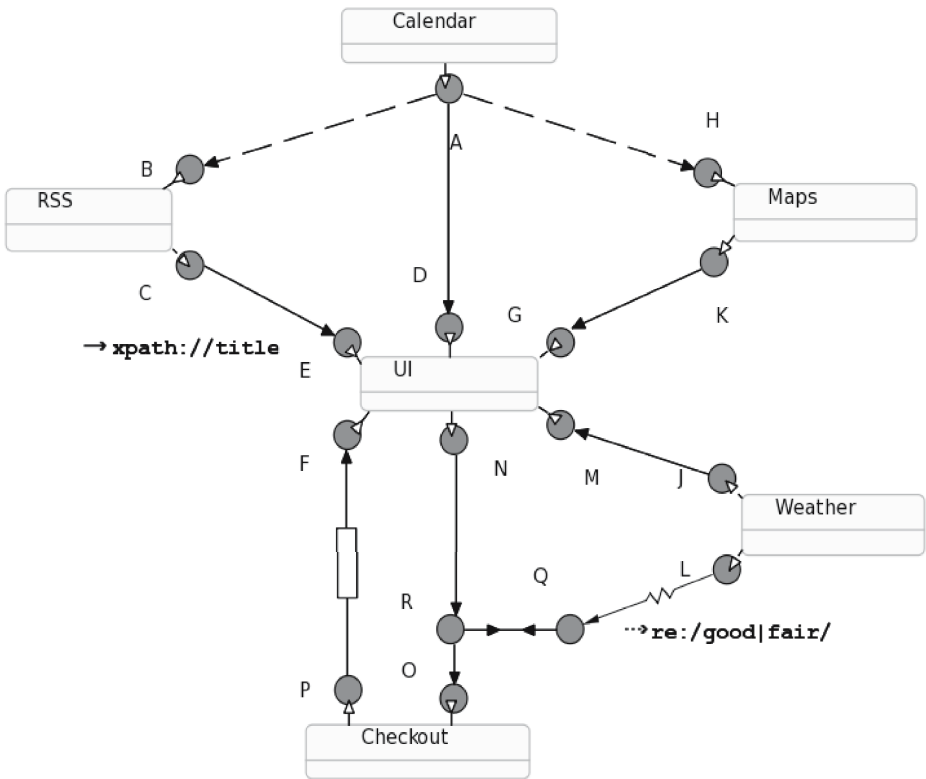
Once channels are composed into a complex connector, it can be used disregarding its internal details. As an example consider the XOR element shown in Figure 2, built out of five sync channels, two lossy sync channels, and a sync drain. The intuitive behaviour of this connector is that data obtained as input through *A* is delivered to one of the output nodes *F* or *G*. If both *F* and *G* is willing to accept data then the merger at node *E* non-deterministically selects which side of the connector will succeed in passing the data. The sync drain channel *B-E* and the two *C-E*, *D-E* channels ensure that data flows at only one of *C* and *D*, and hence *F* and *G*.

**Fig. 2.** XOR connector

Services are independent distributed entities that utilise Reo channels and connectors to communicate. The service implementation details remain fully internal to individual elements, while the behaviour of the whole system is coordinated by the Reo circuit. A deeper treatment of using Reo for service coordination is given in [14]. We discuss how we interface services with Reo in SABRE in §5.1.

### 4.1 Building the Sports-Fan Dashboard in SABRE

Augmenting the Reo tool suite with filter and transformer channels, gives SABRE the data mangling functionality common to other mashup platforms. Filter channels take a filter expression e.g. a data type or regular expression to match against. If a datum matches the filter expression it passes through the channel, otherwise it is dropped. A transformer channel, likewise, accepts a data transformation expressed e.g. as a *sed*-like text replacement or using XPath or XSLT. Each input datum is rewritten according to the transform expression as it passes through. Filters and transformers can also execute user-defined functions. For example, a geo-coding channel that converts place names to latitude and longitude can invoke an external service. Semantically, a filter acts as a specialised lossy sync channel while a transformer acts as a sync.



**Fig. 3.** Reo connector implementing Sports fan mashup's coordination

Figure 3 shows the coordination in our sports fan application defined in Reo. The connector works as follows. Whenever the calendar service has a new event, it is written to node A. From A, the data is passed to node D when the user

interface is ready to accept it. In parallel, data about the new event is also transferred to the RSS and map services, if they are on-line. Otherwise, the lossy channels A-B and A-H just discard the data. The map service provides the location information to the user interface (through the K-G channel) and to the weather service to retrieve the local forecast (K-J channel). The transformer C-E uses an XPath expression to extract titles from the RSS feed for display in the user interface. Whenever the user presses the “Proceed to payment” button, the connector passes the required data through channel N-R. The SyncDrain channel Q-R prevents data flow through the N-R-O pair of channels if the output of the weather service does not “approve” it through the filter channel L-Q, which accepts the data only if the forecast weather forecast is either “good” or “fair”. When the ticket reservation is approved (via a Google Checkout test payment account), the result is put into the FIFO channel P-F. The user interface application then receives the current payment status from the buffer.

Using Reo as the basis for coordination offers a number of advantages. A unique feature of SABRE is that connector in Figure 3 completely specifies not just the data mangling logic, but also synchronisation constraints between services. Just as in Yahoo Pipes, SABRE’s transformer and filter channels can be composed to effect aggregate data filtering and transformation operations. However, thanks to Reo’s *semantic* compositionality, we can also define much more powerful constructs such as the XOR connector in Figure 2. Using library of such predefined connectors, a mashup’s coordination logic can be composed with precise formal semantics.

## 5 Behavioural Specification of Service Using Constraint Automata

Numerous formalisms for behavioural specification have been proposed such as I/O automata[16] and open workflow nets[15]. We use constraint automata[6] to specify the behaviour of services that interact with a SABRE mashup. Constraint automata enables the to use the same formalism as an operational model for the core mashup logic modelled in Reo and for behavioural specification of services we interface with.

**Definition 2 (Constraint Automaton[6]).** *A constraint automaton (over the data domain  $Data$ ) is a tuple  $A = (Q, \mathcal{N}, \rightarrow, Q_0)$  where*

- $Q$  is a set of states,
- $\mathcal{N}$  is a finite set of port names,
- $DC(\mathcal{N}, Data)$  the data constraints, are sets of port name - data assignments,
- $\longrightarrow$  the transition relation of  $A$  is a subset of  $Q \times 2^{\mathcal{N}} \times DC \times Q$
- $Q_0 \subseteq Q$  is the set of initial states.

*For every transition  $(q, \mathcal{N}, g, p) \in \longrightarrow$  we require that: (1)  $\mathcal{N} \neq \emptyset$ , and (2)  $g \in DC(\mathcal{N}, Data)$ .*

A thorough treatment of using constraint automaton as an operational model for Reo connectors can be found in [6]. Intuitively, states represent the configurations of the connector, the transitions the possible one-step behaviour. The meaning of  $q \xrightarrow{(N,g)} p$  is that in configuration  $q$  the port names  $A_i \in N$  have the possibility to perform I/O operations that meet the guard  $g$  and that lead from configuration  $q$  to  $p$ , while the other ports  $A_j \in \mathcal{N} \setminus N$  do not perform any I/O-operation. We discuss the use of constraint automata for service specification below.

### 5.1 Interfacing Web Services with the Sports-Fan Dashboard

Input/output considerations are an integral part of mashup design. We use constraint automata to specify service behaviour in the typical fashion that labelled transition systems are used as formal models for reactive systems. SABRE uses stubs automatically generated from behavioural specifications to bind to services. We extend the method of specifying service behaviour using constraint automata described in [20] to allow for asynchronous service invocation.

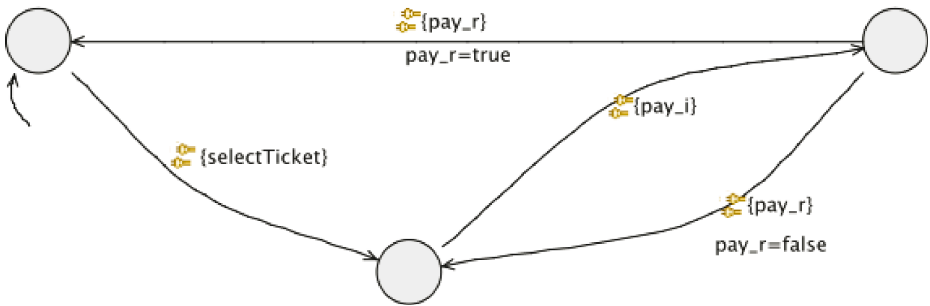


Fig. 4. Constraint automaton specifying the behaviour of the Checkout Service

We use the “Checkout” service in Figure 3 to demonstrate behavioural specification using constraint automata. Suppose this service consists of two operations: `selectTickets` and `pay`. We map each synchronous operations in the service interface to a constraint automaton port with the same name. Asynchronous operations are mapped to two port names suffixed by `_i` and `_r`, corresponding to the operation invocation and return, respectively. In Figure 4, we specify that `selectTickets` is a synchronous operation by placing the single port name on a transition. Following `selectTickets`, a client must invoke the asynchronous `pay` operation that returns a boolean value indicating success or failure. We use constraints to specify data-dependent state changes. For example, based on the return value of the `pay` invocation, we either request payment again, or permit the client to select another ticket to purchase.



## 6 Implementation

The SABRE implementation consists of a mashup design tool and a runtime environment for mashup execution. The design tool is an enhanced version of the Eclipse Coordination Tools [3] (ECT) — a suite of graphical tools for Reo. Built on the Eclipse Graphical Modelling Framework, ECT consists of graphical editors for Reo connectors, an animator for visualising a connector’s semantics, transformation from Reo to constraint automata, and a model checker for constraint automata. The addition of filter and transformer channels brings the data mangling features found in graphical mashup builders like Yahoo Pipes to ECT’s Reo editor. We are integrating the Smooks data transformation framework<sup>5</sup> into SABRE, to enable more powerful filter constraints and data transformations to be specified in a declarative fashion.

SABRE’s execution environment depicted in Figure 5, consists of a Reo engine and a management interface, hosted in a servlet container such as Tomcat. Reo has several executable implementations available, any of which may be used to run a SABRE mashup. ReoCP is a constraint programming engine that directly executes a Reo circuit based on the colouring semantics for Reo [8]. CASP [3] generates Java code from a constraint automaton representation of a Reo connector. A distributed Reo implementation [18] on Scala Actors is ongoing. Any one of these engines may be plugged into the SABRE runtime via a common interface, depending on the specific deployment needs of the application. For example, a deployment requiring run-time changes to the connector may choose to use ReoCP, while deploying a very large connector is best done using distributed Reo.

The management interface lets a user deploy Reo connectors, and start and stop connector instances via a web browser. Once a connector is deployed and started, the runtime initialises the Reo engine with the given connector and instantiates service stubs and other server-side components. Stubs for services (ovals on the left) and user interface elements (ovals on the right), depicted in Figure 5, communicate with the engine via synchronous read and write operations on ports (arrows) of the Reo connector being run by the engine. The SABRE runtime also maps ports to URLs, which may be used by remote components to read and write data to ports using HTTP GET and PUT operations respectively.

SABRE generates Java service stub classes from a constraint automaton and a Java interface<sup>6</sup> declaring operations a service provides. Each start state of the automaton is mapped to a thread which listens for reads and writes on the ports of the outgoing transitions of the current state. Once all ports of a transition are active, the thread invokes the corresponding operation(s) with the parameters received on invocation ports and/or writes any return values to return ports.

For user interface creation we envisage a library of common user interface elements like maps and clickable lists. A user interface element may either execute

<sup>5</sup> <http://milyn.codehaus.org/Smooks>

<sup>6</sup> WSDL specification can be easily translated to Java using tools like Apache Axis.

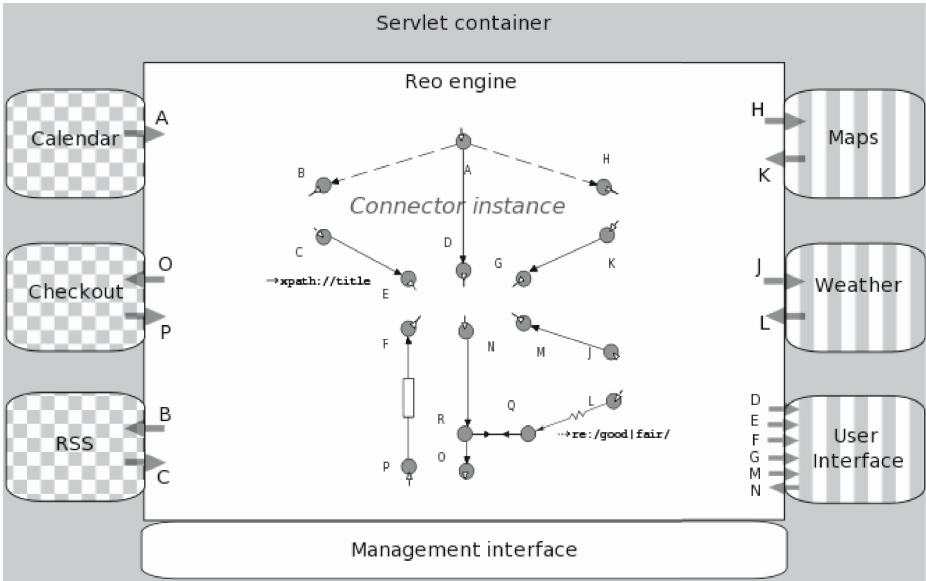


Fig. 5. Sports-fan Dashboard deployed on the SABRE runtime environment

locally on the same server as the mashup, or remotely on the user's browser, such as a component using the Google Maps JavaScript API. Such browser-based UI elements may use JavaScript's `XMLHttpRequest` object to perform I/O via the URL mapped ports, with the Reo connector executing on the server<sup>7</sup>.

## 7 Conclusion and Future Work

We have presented a framework for rapid composition of heterogeneous data and services on the Web. Rather than approach service coordination from the traditional control flow perspective, we take a data-centric view inspired by mashups. The SABRE framework permits compositional construction of mashups with precise semantics and fault isolation, without sacrificing rapid development and flexibility inherent in mashups. The synchronous semantics of Reo gives us simple transactions that can ensure that a chain of components connected by channels all execute atomically. Compensation is another major concern in specification and implementation of business processes that involve long running transactions. We intend to adapt the schemes used to translate the compensation constructs available in the BPMN standard to provide a compensation mechanism for SABRE [4]. In summary, SABRE improves on the current state of the art in mashup construction platforms by leveraging the strengths of Reo, by using it as the basis for formalising coordination logic in a mashup.

<sup>7</sup> This is the same technique known as *AJAX* in common parlance.

Specifying service behaviour in constraint automata is onerous for casual mashup development. An alternative is to describe a service by a UML sequence diagram and then extract the behavioural specification in the form of a constraint automaton [5]. Ongoing work on Reo also makes it possible dynamically reconfigure connectors based on graph transformations [13]. Dynamic reconfiguration opens up possibilities such as run time service discovery and binding. Finally, we would like to add more prepackaged components along the lines of Yahoo Pipes and streamline SABRE's graphical interface to make it accessible to non-technical end-users.

## References

1. Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y., Simmen, D., Singh, A.: Damia - a data mashup fabric for intranet applications. In: VLDB, pp. 1370–1373 (2007)
2. Arbab, F.: Reo: a Channel-based Coordination Model for Component Composition. *Mathematical Structures in Computer Science* 14, 329–366 (2004)
3. Arbab, F., Koehler, C., Maraikar, Z., Moon, Y., Proença, J.: Modeling, testing and executing Reo connectors with the Eclipse Coordination Tools. In: Proceedings of FACS, SCP (to appear, 2008)
4. Arbab, F., Kokash, N., Sun, M.: Towards using Reo for compliance-aware Business Process Modelling. In: Proceedings of ISOLA (to appear, 2008)
5. Arbab, F., Meng, S.: Synthesis of connectors from scenario-based interaction specifications. In: Chaudron, M.R.V., Szyperski, C., Reussner, R. (eds.) CBSE 2008. LNCS, vol. 5282, pp. 114–129. Springer, Heidelberg (2008)
6. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by Constraint Automata. *Sci. Comput. Program* 61(2), 75–113 (2006)
7. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. *Electr. Notes Theor. Comput. Sci.* 154(1), 101–119 (2006)
8. Clarke, D., Proença, J., Lazovik, A., Arbab, F.: Deconstructing Reo. In: Proceedings of FOCLASA. ENTCS (to appear, 2008)
9. Curbera, F., Goland, Y., Klein, J., Leymann, F.: Business process execution language for web services. Technical report, IBM (2002), <http://www.ibm.com/developerworks/library/ws-bpel/>
10. Fielding, R.: Architectural styles and the design of network-based software architectures. PhD thesis, Chair-Richard N. Taylor (2000)
11. Jhingran, A.: Enterprise information mashups: Integrating information, simply. In: VLDB, pp. 3–4 (2006)
12. Kavantzaz, N., Burdett, D., Ritzinger, G.: Web services choreography description language (WS-CDL) version 1.0. Working draft, W3C (2004), <http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427>
13. Koehler, C., Costa, D., Proença, J., Arbab, F.: Reconfiguration of Reo connectors triggered by dataflow. In: Proceedings of GT-VMT. Electronic Communications of the EASST, vol. 10 (2008)
14. Lazovik, A., Arbab, F.: Using Reo for service coordination. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 398–403. Springer, Heidelberg (2007)

15. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services (chapter Behavioral Constraints for Services). In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer, Heidelberg (2007)
16. Lynch, N., Tuttle, M.: An introduction to input/output automata. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands (1980)
17. Mousavi, M., Sirjani, M., Arbab, F.: Formal semantics and analysis of component connectors in Reo. ENTCS 154(1), 83–99 (2006)
18. J. Proença Towards Distributed Reo. In: CIC workshop (2007), <http://homepages.cwi.nl/~proenca/distributedreo>
19. Scholten, J., Arbab, F., de Boer, F., Bonsangue, M.: A component coordination model based on mobile channels. Fundam. Inform. 73(4), 561–582 (2006)
20. Sun, M., Arbab, F.: Web Services Choreography And Orchestration In Reo And Constraint Automata. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), pp. 346–353. ACM, New York (2007)
21. Wirsing, M., Clark, A., Gilmore, S., Hölzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)

# Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques

Radu Mateescu<sup>1</sup>, Pascal Poizat<sup>2,3</sup>, and Gwen Salaün<sup>4</sup>

<sup>1</sup> INRIA/VASY project-team, aile de l'Ingénieur, bât. LE2I, Dijon, France  
`radu.mateescu@inria.fr`

<sup>2</sup> INRIA/ARLES project-team, France  
`pascal.poizat@inria.fr`

<sup>3</sup> IBISC FRE 3910 CNRS – Université d'Évry Val d'Essonne, France

<sup>4</sup> University of Málaga, Spain  
`salaun@lcc.uma.es`

**Abstract.** Software Adaptation is a hot topic in Software Engineering since it is the only way to compose non-intrusively black-box components or services with mismatching interfaces. However, adaptation is a complex issue especially when behavioral descriptions of services are considered. This paper presents optimised techniques to generate adaptor protocols, being given a set of service interfaces involved in a composition and an adaptation contract. In this work, interfaces are described using a signature, and a protocol that takes value passing into account. Our proposal is completely supported by tools that automate the generation and the verification of the adaptor protocols. Last, we show how our adaptation techniques are implemented into BPEL.

## 1 Introduction

Service composition is a central issue in Service Oriented Computing. Reuse of existing entities is mandatory not to implement again the same blocks of software, and then help developers to reduce development time, respect delays, and have their companies save money by diminishing software design costs. However, direct reuse and composition of existing services is in most of cases impossible because their interfaces present some incompatibilities. *Software Adaptation* [3] is a very promising solution to compose in a non-intrusive way black-box components or (Web) services whose functionality is as required for the new system, although they present interface mismatches. Adaptation techniques aim at automatically generating new components called *adaptors*, and usually rely on an *adaptation contract* which is an abstract description of how mismatches can be worked out. All the messages pass through the adaptor which acts as an orchestrator, and makes the involved services work correctly together by compensating mismatches.

**Contributions.** Model-based behavioral adaptation approaches are either restrictive or generative. *Restrictive approaches* [5,2] try to solve the problem by

cutting off (pruning) the behaviors that may lead to mismatch, thus restricting the functionality of the services involved. *Generative approaches* [4,7] try to accommodate the protocols without restricting the behavior of the services, by generating adaptors that act as mediators, remembering and reordering events and data when necessary. In the current state of the art, restrictive approaches are fully automated and are directly related to programming languages, but they do not support advanced adaptation scenarios. On the other hand, generative approaches suffer from the computational complexity of generating adaptors, often lack of tool support, and are not related to implementation languages. In this paper, we propose model-based adaptation techniques that are both generative and restrictive since we support complex adaptation scenarios (such as message reordering), while removing incorrect behaviors. We also diminish the computational complexity of adaptor generation by using on-the-fly exploration and reduction techniques to avoid the generation of the full state space of the adaptor under construction. Last, let us emphasize that our approach is fully supported by tools we implemented, and adaptors are finally implemented using service implementation languages.

**Approach.** In this paper, we first present a model of services that makes it possible to describe signatures (operation names and types) and behaviors (interaction protocols). Protocols are essential because erroneous executions or deadlock situations may occur if the designer does not take them into account while building composite services. More than only considering messages exchanged in protocols, it is important to include value passing (parameters) coming with messages since this feature may raise composition issues too (unmatching number of parameters, different ordering, etc). Next, we introduce the contract notation that is used to describe how mismatches appearing in signatures and protocols can be worked out by defining correspondences between messages but also between message parameters. Then, from a set of service protocols and a contract, we present our approach to generate adaptor protocols which relies on (i) encodings into the LOTOS process algebra [14], and (ii) on-the-fly exploration and reduction techniques. Verification of contracts is also possible by using CADP [13] a rich verification toolbox for LOTOS. Last but not least, we show how adaptors can be implemented in the WS-BPEL (BPEL for short) service orchestration language. Our proposal is supported by tools (Fig. 1) that automate the extraction of abstract interfaces from XML description of services (BPEL2STS), the generation of the LOTOS encoding (Compositor), the efficient computation of the adaptor protocol from the LOTOS specification (Scrutator), the verification of the adapted system (Evaluator), and the generation of BPEL from adaptor models (STS2BPEL). The only step of our approach which requires manual intervention is the adaptation contract construction.

**Outline.** The remainder of this paper is structured as follows. Section 2 presents our model of services. Section 3 introduces the contract notation which is used for adaptation purposes. In Section 4, we present the adaptor generation and verification techniques. Section 5 focuses on adaptor implementation. Section 6

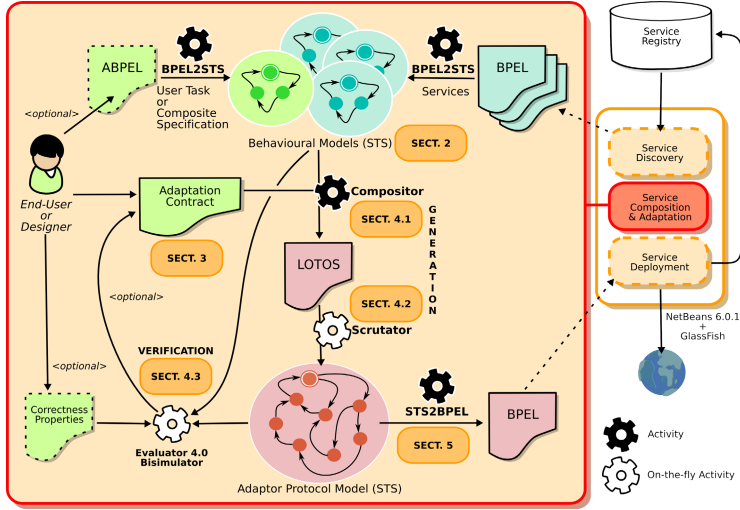


Fig. 1. Overview of our approach

compares our approach to related work, and Section 7 ends the paper with some concluding remarks.

## 2 Service Model

In this section we present our service interface model. We assume that service interfaces are given using both a signature and a protocol. *Signatures* correspond to operation profiles described using WSDL, *i.e.*, operation names associated with argument and return types relative to the messages and data being exchanged when the operation is called. Additionally, we propose that protocols are represented by means of *Symbolic Transition Systems* (STSs) which are Labelled Transition Systems (LTSs) extended with value passing (data parameters coming with messages). Communication between services is represented using *events* relative to the emission (denoted using !) and reception (denoted using ?) of *messages* corresponding to operation calls. Events may come with a set of data terms whose types respect the operation signatures. In our model, a *label* is either the internal action ( $\tau$ ) or a tuple  $(SI, M, D, PL)$  where  $SI$  is a service identifier,  $M$  is a message name,  $D$  stands for the direction (!,?), and  $PL$  is either a list of data terms if the message corresponds to an emission, or a list of variables if the message is a reception.

An STS is a tuple  $(A, S, I, F, T)$  where:  $A$  is an alphabet that corresponds to message events relative to the service provided and required operations,  $S$  is a set of states,  $I \in S$  is the initial state,  $F \in S$  are final states, and  $T \in S \times A \times S$  is the transition function. This formal model has been chosen because it is simple, graphical, and it can be easily derived from existing implementation platforms' languages, see for instance [10,21,9] where such abstractions for Web services

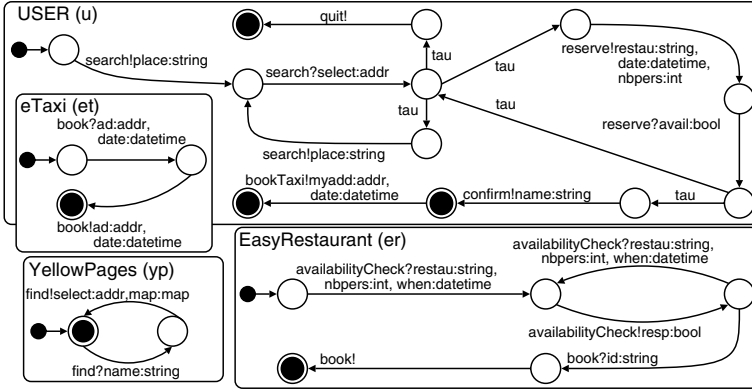


Fig. 2. Example – service protocols

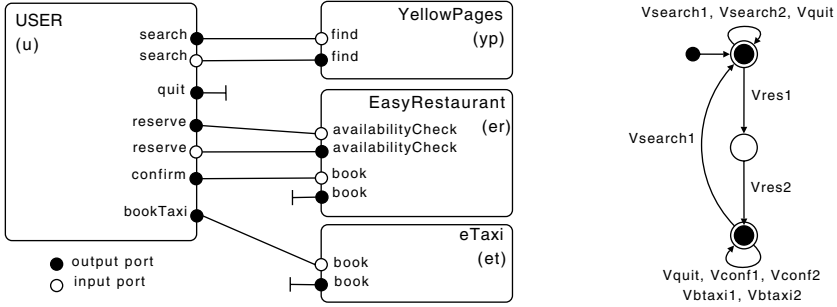
were used for verification, composition or adaptation purposes. For space reasons, in the rest of the paper, we will describe service interfaces only with their STSs. Signatures will be left implicit, yet they can be inferred from the typing of arguments (made explicit here) in STS labels.

**Example.** We will use throughout this paper an on-line restaurant booking system as a running example. First of all, let us present the three existing services we reuse to build this new system (Fig. 2). Service **YellowPages** can receive a search request, and returns an address and a map. Service **EasyRestaurant** can receive and answer availability requests to check if a restaurant has room for a given date and number of people. After these interactions, this service can receive a booking message and send an acknowledgement back. Service **eTaxi** receives booking requests with address and date. In addition, we give the system end-user requirements (**USER**). The user can first look for a place. Then, (s)he can search again, quit, or reserve a restaurant found in the former step. If reservation is possible, the user can accept and book a taxi if necessary. The tau transitions in the user protocol stand for internal decisions taken by her/him.

### 3 Adaptation Contracts

In this section, we present the adaptation contract notation that allows us to specify interactions and to work mismatch situations out. We rely on *synchronization vectors* [1] (or vectors for short). They express correspondences between messages, like bindings between ports or connectors in architectural descriptions. Each event appearing in a vector is executed by one service and the overall result corresponds to an interaction between all the involved services. A vector may involve any number of services and does not require interactions occurring on the same names of events. Furthermore, variables are used in events as placeholders for message parameters. The same variable name appearing in different events (possibly in different vectors) enables one to relate sent and received message





**Fig. 3.** Example – (left) system architecture, (right) vector LTS

parameters. Vectors can be either written by hand or obtained from a graphical description of the architecture built by the designer (Fig. 3).

However, vectors are not sufficient to support more advanced adaptation scenarios such as contextual rules, choice between vectors or, more generally, ordering (*e.g.*, when one message in some service corresponds to several in another service, which requires to apply several vectors in sequence). The ordering in which vectors have to be applied can be specified using different notations such as regular expressions, LTSs, or (Hierarchical) Message Sequence Charts. Due to their readability and user-friendliness, we chose to specify adaptation contracts using *vector LTSs*, that is, LTSs whose labels are vectors. In addition, vector LTSs ease the development of adaptation algorithms since they provide an explicit description of the adaptation contract set of states. An adaptation contract for a set of service STSs is a couple  $(V, L)$  where  $V$  is a set of vectors, and  $L$  is a vector LTS built over  $V$ . If only message name correspondences are necessary to solve mismatches between services, the vector LTS may leave the vector application order unconstrained using a single state and all vector transitions looping on it. In particular, this pattern may be used on specific parts of the contract for which the designer does not want to impose any ordering.

**Example.** The very first step in the construction of an adaptation contract is to relate messages, and then building the architecture of the system-to-be. The graphical architecture of our booking system is shown in Figure 3 (left) where for instance the `search` messages in the user requirements correspond to the `find` ones in the `YellowPages` service. A specific notation is used to denote an unsynchronized message, *i.e.*, a message with no correspondence (see `quit` in the user requirements for example).

However, such an architecture is not sufficient because we are considering value passing too, and data exchanged through messages (Fig. 2) have to be matched as well. We give below the vectors that are first deduced automatically from the architecture and complemented with data mappings. As an example the `search` request emitted by the user comes with a parameter (`place`) whose counterpart is the argument coming with the reception of `find` in the `YellowPages` service

(vector  $V_{search1}$ ). Next, in vector  $V_{search2}$ , we can see that answer sent by the **YellowPages** service comes with two parameters, one of which is matched (**select**) but the other one (**map**) is received by the adaptor yet never used afterwards in any other vector. An example of data reordering exists in vector  $V_{res1}$ . Note that the variable scope is not limited to one vector, and a data received in a vector can be used (sent) in another. We have implemented analysis techniques to check possible scope inconsistencies (see Section 4.3).

```

Vsearch1 = ⟨u : search!place; yp : find?place⟩
Vsearch2 = ⟨u : search?select; yp : find!select, map⟩
Vquit    = ⟨u : quit!⟩
Vres1    = ⟨u : reserve!restau, date, nbpers; er : availabilityCheck?restau, nbpers, date⟩
Vres2    = ⟨u : reserve?resp; er : availabilityCheck!resp⟩
Vconf1   = ⟨u : confirm!name; er : book?name⟩
Vconf2   = ⟨er : book!⟩
Vbtaxi1  = ⟨u : bookTaxi!address, date; et : book?address, date⟩
Vbtaxi2  = ⟨et : book!address, date⟩

```

Being given this set of interactions, the user would be able to submit infinitely availability requests to **EasyRestaurant** for the same restaurant, which is useless. Accordingly, a vector  $LTS$  is defined (Fig. 3, right) in order to impose a single interaction between the user and the **EasyRestaurant** service every time the user is eager to check for place availability at some restaurant.

## 4 Adaptor Generation and Verification

An adaptor model for a set of services is an STS running in parallel with the service STSs and guiding their execution (all exchanged messages pass through the adaptor) in such a way that mismatches are avoided and the ordering of messages imposed by the adaptation contract is guaranteed. Generating adaptor protocols is a complicated task since the adaptor has to respect the adaptation contract taking into consideration behavioral constraints of services formalised into their interfaces (STSS). In addition, protocols may generate many interleaved interactions that we want to preserve to accept all the possible message execution orders.

In this work, we chose to encode the adaptation constraints (service interfaces and adaptation contract) into the LOTOS process algebra [14]. LOTOS relies on a rich notation that allows to specify complex concurrent systems possibly involving data types. Our goal is first to generate LOTOS code for service interfaces and their interaction constraints as specified in the contract. In a second step, the LOTOS encoding allows the automatic generation of adaptor protocols whose traces represent all possible (correct) interactions between services. To do so, we rely on CADP [13] a toolbox for LOTOS which implements optimised state space exploration techniques. In particular, we employ *on-the-fly* algorithms to increase, *w.r.t.* existing approaches, the efficiency of the adaptor generation and reduction process by avoiding the generation of the full state space. The LOTOS encoding also enables the adaptor protocol verification by using model

checking tools available in CADP. Techniques and tools presented in this section have been validated on more than 200 examples.

### 4.1 Principles of the Encoding into LOTOS

This approach aims at successively encoding: the services’ STSs, the abstract requirements for composition and adaptation (*i.e.*, the adaptation contract), and the desired system architecture that formalises how the services interact guided by the contract.

**Service STS encoding.** Each service STS  $sv$  is encoded using several LOTOS processes. Each LOTOS process corresponds to one state  $s$  of the STS, and its behavior is a choice containing as many branches as there are transitions outgoing from  $s$ . Each branch encodes the label associated to the transition, and is followed by a call to the LOTOS process that encodes the target state of the transition being translated. An additional branch, using a specific **FINAL** action, models termination when  $s$  is final. STS labels are encoded into LOTOS following patterns presented in Figure 4. Sent (resp. received) messages are represented with a “**\_EM**” (resp. “**\_REC**”) suffix. In addition, LOTOS symbols **!** and **?** are used to support data transfer (resp. emission and reception). In our context, the correct distribution will be ensured by the encoding of the adaptation constraints (see the next step in this section), therefore all service STS labels that involve value passing (emission or reception of parameters) are translated into LOTOS with a question mark followed by as many fresh variables as there are parameters coming with the message. Since these variables are placeholders, their LOTOS type can simply be an arbitrary one that we call **PH**. This type is defined beforehand using the LOTOS abstract datatype facilities with all the placeholder names appearing in the contract defined as type constructors.

**Adaptation contract encoding.** An adaptation contract is encoded by generating (i) a process for the vector **LTS**, (ii) a process for each vector defined in the contract, and (iii) the interleaving of all these vector processes. The correct ordering of vectors is ensured by the vector **LTS** process thanks to two actions for each vector  $v$ . A first one (**run\_v**) activates the corresponding vector process. A second one (**rel\_v**) releases the vector **LTS** and enables it to overlap vector

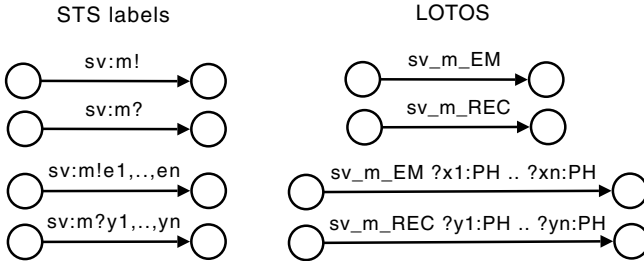


Fig. 4. Encoding patterns for STS labels

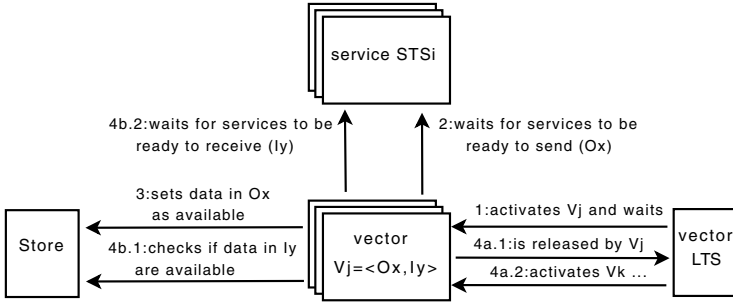


Fig. 5. Encoding pattern for vectors

applications. The vector LTS process (i) is encoded using the same pattern as service STSs, that is every state is encoded as a LOTOS process. For each transition with label  $v$  in the vector LTS, two LOTOS actions are generated in a sequence:  $\text{run}_v$ ;  $\text{rel}_v$ .

Vector processes (ii) are first launched through a “ $\text{run}_v$ ” interaction with the vector LTS (Fig. 5, 1). Next, they communicate with services on all actions appearing in their vector definition. They have to receive all sent messages (Fig. 5, 2) before beginning to emit some (Fig. 5, 4b.2). There is no specific ordering between receptions (*resp.* between emissions) in a vector process. When a vector process executes a vector, it must be ready to interact with the service STSs on their emissions ( $Ox$  in Fig. 5). Then, several strategies are possible to release the vector ( $\text{rel}_v$ ), and therefore to execute the services’ receptions. A first option is to wait the complete processing of a vector before firing a new one (4a.2 done after 4b.1 and 4b.2). Another strategy is to execute the release action once all the emissions executed, that means that the execution of the receptions by the services ( $Iy$  in Fig. 5) can be postponed, and the vector LTS can launch another vector. This behavior makes the reordering of messages possible, a typical case of mismatch between services.

As regards value passing, an auxiliary LOTOS process **Store** is generated to store information about the availability of received values. Every time some values are sent by a service, they are received by one of the vector processes and stored by using the (global) process **Store**, which makes them available at the level of the adaptor. This availability is essential, because when service receptions in a vector are being run (emissions at the level of the adaptor), this firing is conditioned by the availability of the values to be emitted. Thus, every service emission in a vector is followed by an interaction with the process **Store** to set to *true* the availability of the received values (Fig. 5, 3), and every service reception in a vector is preceded by some interactions with the **Store** process to check that the required values have been received (Fig. 5, 4b.1). In the latter case, the vector process may have to wait the availability of the needed resources. Such an active waiting is encoded using a looping process that terminates once the data are available. If they are never available, this will generate a deadlock

in the underlying state space that will be cut away in a second step by our reduction techniques (see Section 4.2).

Finally, vector processes (iii) are interleaved since they do not communicate together. All the vector processes may synchronize with the `Store` process to store new available values, or check the availability of some values to be sent. The collaboration diagram in Figure 5 summarizes the pattern for encoding vectors into LOTOS when vector overlapping is enabled.

**System encoding.** In this step, we generate a LOTOS expression corresponding to the whole system constraints from the LOTOS processes encoding the service STSs, the ones encoding the adaptation contract, and respecting the desired system architecture (adaptor in-the-middle, intercepting all messages). This means that the service STSs only interact together on `FINAL` (correct termination is when all services terminate) while they interact with vectors on actions used in their alphabets. The synchronizing between vector processes and vector LTS has been described earlier on (using “`run_`” and “`rel_`” actions). In addition, all actions that are not messages of the system, *i.e.*, messages appearing in the involved services, are hidden as they represent internal actions of the adaptor we are building (*e.g.*, “`run_`” and “`rel_`” actions, or all interactions with the `Store` process). They are the “mechanics” of adaptation and are not relevant for implementation. They will be removed by reduction steps of the adaptor generation process (see Section 4.2).

**Tool support: Compositor.** The LOTOS encoding is fully automated by `Compositor`, a tool we have implemented. Supported inputs are XML STSs and the `aut` LTS textual format extended with value passing for service interfaces, and XML for contract specifications. Strategies to implement the different ways of releasing vectors have been implemented as an option.

## 4.2 On-the-Fly Adaptor Generation

An adaptor can be obtained from the state space of the whole system (services and adaptation contract) by keeping only the correct behaviors, which amounts to cut the execution sequences leading to deadlock states. In the adaptation techniques that support deadlock elimination [62], the computation of the deadlock-free behaviors is done by performing a backward exploration of the explicit, *entirely constructed*, state space by starting at the deadlock states and cutting all the transitions whose target state leads to a deadlock. To increase efficiency, we avoid the entire construction of the state space and instead we explore it forwards in order to generate the adaptor *on-the-fly* by carrying out deadlock elimination and behavioral reduction simultaneously.

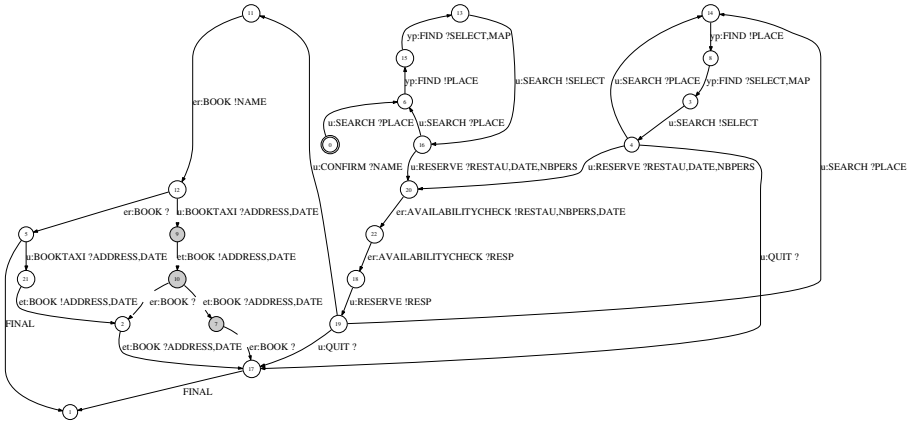
**Deadlock elimination.** First, the execution sequences leading to deadlocks must be pruned. We do this by keeping, for each state encountered, only its successor states that potentially reach a successful termination state, which is source of a transition labeled with the action `FINAL`. Besides avoiding deadlocks (sink states reached by actions other than `FINAL`), this also avoids livelocks,

*i.e.*, portions of the state space where some services get “trapped” and cannot reach their final states anymore. The desired successor states satisfy the PDL [8] formula  $\langle true^*.FINAL \rangle true$ , which can be checked on-the-fly using the Evaluator [18] model checker. However, this scheme is not efficient since each invocation of Evaluator has a linear complexity *w.r.t.* the size of the state space and therefore a sequence of invocations (in the worst case, one for each state) may have a quadratic complexity. An efficient solution is to translate the evaluation of the formula into the resolution of the boolean equation system (BES)  $\{X_s = \mu \bigvee_{s \xrightarrow{FINAL} s'} true \vee \bigvee_{s \rightarrow s''} X_{s''}\}$ , where a boolean variable  $X_s$  is true iff state  $s$  satisfies the propositional variable  $X$  corresponding to the PDL formula. A state  $s$  potentially leading to a successful termination is detected by solving on-the-fly the variable  $X_s$  of this BES using the algorithm A3 of the Caesar\_Solve library [16]. In this way, a sequence of resolutions performed during a forward exploration of the state space has a linear-time overall complexity and does not store transitions, but only states in memory.

**Behavioral reduction.** Second, the adaptor STS obtained by pruning can be reduced on-the-fly modulo an appropriate equivalence relation in order to get rid of the internal actions and obtain an adaptor as small as possible. These internal actions correspond here to the encoding of the system adaptation constraints, *e.g.*, “run\_” and “rel\_” actions. Such internal actions are not relevant for adaptor implementation but are usually inherent to adaptation processes, as they model internal computations done by adaptors [6]. The algorithms presented in [15] can be used to implement on-the-fly reductions modulo tau-confluence (a form of partial order reduction preserving branching bisimulation) and the tau\*.a and weak trace equivalences, both of which eliminate internal transitions and (for weak trace) determinize adaptor STSs.

**Tool support: Scrutator and CADP.** The on-the-fly adaptor generation is implemented by the Scrutator tool that we have developed using the Open/Caesar [11] environment for graph manipulation provided by the CADP verification toolbox. Two kinds of pruning are implemented by the tool: the first one deletes the states leading eventually to deadlocks and the second one keeps only the states leading (potentially or eventually) to transitions labeled by a given action (here, FINAL). Besides the on-the-fly reductions currently offered by Scrutator (tau-confluence, tau\*.a, and weak trace equivalence), we plan to implement reductions modulo other equivalences, such as branching bisimulation; for the time being, the adaptors generated by Scrutator can be reduced off-line modulo strong or branching bisimulation using the Bcg\_Min tool of CADP.

To automate the whole adaptation process, Compositor generates an SVL script [12] in charge of the following activities: building and reducing the adaptor on-the-fly by invoking Scrutator on the LOTOS specification of the system; “mirroring” of the adaptor actions (reversing emissions and receptions, “EM” and “REC”) as the adaptor acts as an orchestrator in-the-middle of the services; and pretty-printing of the adaptor STS by translating its actions from a LOTOS-like to a more user-friendly syntax.



**Fig. 6.** Example – adaptor protocol

The reduced adaptor protocol for our running example is shown in Figure 6. The initial state is identified by 0. In this state, the adaptor can interact with the user (message `SEARCH`) and receives as parameter the place (s)he is looking for. Next, the adaptor sends this place to the `YellowPages` service with the `FIND` message, etc.

### 4.3 Adaptor Verification

In our approach, contracts are built by the designer. Therefore, they can contain errors that will also appear at the level of the adaptor. As a first step in the verification of the adaptor, we have implemented several static analysis checks to verify that the contract is correctly written (labels defined in interfaces correctly used in vectors, vectors and vector LTS structurally consistent, scope and type of placeholders, etc). These static analysis features are very useful for detecting the simple errors that one can make while writing out a contract manually. Nonetheless, this is not enough since protocols of interfaces and contracts (vector LTS) are not considered. Therefore, to complement static analysis checks, we propose more powerful verification techniques based on model checking tools (`Evaluator`). Two kinds of temporal properties are suitable for checking the behavior of adaptors: (i) *general properties* (placeholder occurrence, service action preserving, etc) related to the adaptor structure, which should be satisfied by any adaptor generated using our approach, (ii) *specific properties* (safety and liveness) related to the adaptor protocol, which differ from one adaptor to another.

## 5 Adaptor Implementation

In this section we present the final step of our approach, namely adaptor implementation. Due to lack of space, the initial step, generating STS models from

(A)BPEL (using the rules defined in [21]) is not presented here. To generate a BPEL orchestrator from an adaptor model we proceed in two steps: (i) filtering the model, and (ii) encoding the filtered model into BPEL.

**Adaptor filtering.** The adaptor generation algorithm is a implementation independent model-based one whose objective is to be applied to different implementation platforms (BPEL, Windows Workflows, SCA components, etc.). To support implementation using the BPEL constructs, we have to apply first three simplification rules:

- whenever a state has both emission and reception outgoing transitions, we remove the reception transitions;
- whenever a state has more than one emission outgoing transition, we keep a single one;
- let  $o$  be a two-way operation, *i.e.*, a receive-reply operation of a service to be invoked by the adaptor in a synchronous way; for every transition  $t$  with an emission corresponding to such an  $o$ , and targeting state  $s$ , we remove all transitions outgoing from  $s$  but for the transition with the corresponding reception (*i.e.*, we impose atomicity of the two events corresponding to invocations in the adaptor). In a case where such a second transition is not available, we also remove  $t$ .

We end by cleaning the adaptor model, *i.e.*, we remove states (and accordingly transitions) which are not reachable (from the initial state) or not coreachable (from a final state). Filtering is demonstrated on Figure 6 where the grey states and related transitions are removed. Filtering is compatible with adaptation; it just removes some of the interactions between the services which are not possible from a BPEL point of view. Verification techniques presented for adaptor models apply to filtered models too. Currently, we have been able to show that the important safety and liveness properties that yield for the Figure 6 adaptor (*e.g.*, that the client cannot be asked to confirm the reservation before the YellowPages service has found an appropriate place, or that the client cannot be asked to confirm the reservation before the YellowPages service has found an appropriate place) yield also after filtering.

**BPEL implementation.** Once models have been cleaned up as presented above, we automatically implement them in BPEL as follows.

*Partnerlinks and variables.* A partnerlink is created for each service, plus one for the composite itself (USER). Global variables are created for the vector variables and for each part of received or emitted message. Moreover, a STATE integer variable is used to represent the current state and a FINAL boolean variable to represent the termination of the adaptor.

*Communication.* A  $c:\text{msg!}x_1, \dots, x_n$  transition ( $c$  not being USER) followed by a  $c:\text{msg?}y_1, \dots, y_n$  transition is encoded as a synchronous *invoke* activity with message  $\text{msg}$  and partnerlink  $c$ . A  $\text{USER}:\text{msg?}x_1, \dots, x_n$  transition corresponds to the interaction with the environment, it is encoded as a *receive* activity with message  $\text{msg}$  and partnerlink USER. Finally, a  $\text{USER}:\text{msg!}x_1, \dots, x_n$  transition corresponds



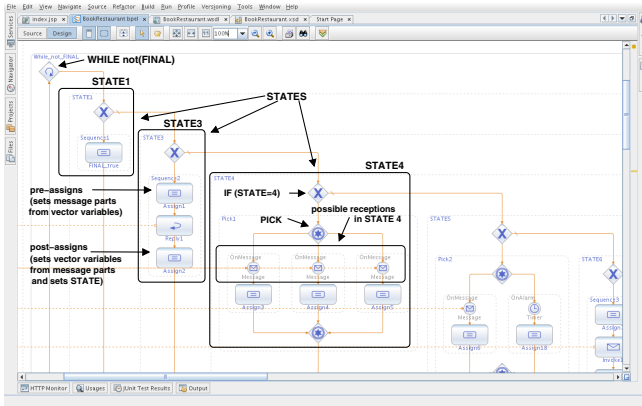


Fig. 7. BPEL Adaptor (part of) in the NetBeans IDE 6.0.1

to an interaction with the environment, it is encoded as a *reply* activity with message `msg` and partnerlink `USER`. All communication activities related to `USER` are linked using a correlation set named `USER_CS` with a property `USER_PROP`. Moreover, each of the operations in the `USER` interface has an additional part with a string identifier and a corresponding property alias making the link with `USER_PROP`. This machinery is required to ensure the correctness of the adaptor protocol *w.r.t.* its environment, *e.g.*, the user.

*Assignments.* Some adaptor variables (`xi` and `yj` above, *e.g.*, `name` in our example) come from vectors, while message parts in the communication activities (`invoke`, `receive`, `reply`) correspond to variables in service protocols (*e.g.*, `id` for message `book` in service `EasyRestaurant` in our example). To link them, before each `invoke` or `reply` activity, we add an `assign` activity assigning adaptor variables to message parts; accordingly, after each `invoke` or `receive` activity we add an `assign` activity assigning message parts to adaptor variables.

*Process.* We rely on the state machine pattern. Initially the `STATE` variable is set to the target state of the first transition in the adaptor. The main body of the process then corresponds to a `while (not FINAL)` activity. Cascaded `if` statements are used inside it to encode the adaptor states. The `if` body of a state `i` encodes its outgoing transition(s). For a single one we use communication encodings presented above. When there are several possible receptions we use a `pick` activity with an `onMessage` branch for each. If there is also a termination transition, we add an `onEvent` branch in the `pick` with a timer. In all cases, we terminate by updating the `STATE` variable accordingly to the transition(s) taken into account. For the final state we only set `FINAL` to true.

**Tool support: BPEL2STS and STS2BPEL.** The obtaining of STS from BPEL, the filtering of adaptor models, and the generation of BPEL adaptors from STS models, presented above, are automated by `BPEL2STS` and

STS2BPEL, tools we have implemented. A part of our adaptor in BPEL is presented in Figure 7. Service deployment has been achieved using the NetBeans 6.0.1 IDE with the GlassFish BPEL Engine.

## 6 Related Work

Several adaptation proposals [4,6,2] focus on solving behavioral mismatch between abstract descriptions of components. Brogi et al. (BBC) [4] present a methodology for generative behavioral adaptation where component behaviors are specified with a subset of the  $\pi$ -calculus and composition specifications with name correspondences. An adaptor generation algorithm is used to refine the given specification into a concrete adaptor which is able to accommodate both message name and protocol mismatch. This approach has recently been used to obtain adaptor implementations for services [5] (see below). Autili et al. (IT) [2] address the enforcement of behavioral properties out of a set of components. Starting from the specification with MSCs of the components to be assembled and of LTL properties (liveness or safety) that the resulting system should verify, they automatically derive the adaptor glue code for the set of components in order to obtain a property-satisfying system. They follow a restrictive adaptation approach, hence they are not able for example to reorder messages when required. More recently, in [6], we have proposed an automated adaptation approach that is generative and supports adaptation policies and system properties described by means of regular expressions of vectors. It superseded both IT (as it supported message reordering) and BBC (which could generate dumb adaptors [4] and has no tool-support), yet it built on algorithms based on synchronous products and Petri nets encodings with a resulting exponential complexity for the computation of adaptors. Here, this is avoided thanks to process algebra encodings and on-the-fly generation techniques.

In their paper *Adapt or Perish* [7], Dumas and collaborators presented an approach to behavioral interface adaptation based on the definition of a set of adaptation operations for establishing the basic relation patterns between the messages names used in the components being adapted, and they defined a trace-based algebra for describing the transformations required to solve the adaptation problem. They also present a visual notation for describing a mapping between the behavioral interfaces of the components. However, their proposal does not present a solution for deriving an adaptor from the visual mappings, but just contains a preliminary (*i.e.*, non sufficient) condition for detecting deadlock scenarios in the behavioral interfaces.

Some recent approaches found in the literature [5,20,19] focus on existing programming languages and platforms, such as BPEL or SCA components, and suggest manual or at most semi-automated techniques for solving behavioral mismatch. In the context of Web services and BPEL, [5] outlines a methodology for the generation of adaptors capable of solving behavioral mismatches between BPEL processes. In their adaptation methodology, the authors use an intermediate workflow language for describing component behavioral interfaces, and they

use lock analysis techniques to detect behavioral mismatch. Similarly, [20] provides automated support for the identification of protocol-level mismatches, but is able to generate an adaptor only in the absence of deadlock. If deadlock may arise from the combination of the components, the authors propose a way to handle the situation by generating a tree for all mismatches that result in a deadlock, and suggesting some hints for assisting the designer in the manual implementation of the actual adaptor. In [19], the authors deal with the monitoring and adaptation of BPEL services at run-time according to Quality of Services attributes (different focus than us). Their approach also proposes replacement of partner services based on various strategies either syntactic or semantic.

Finally, compared to a preliminary version of this work [17], in the current paper, we have first extended the model of services with value passing. Consequently, the contract notation was enhanced as well to consider not only message matching but also correspondences between message arguments. New adaptation and verification techniques have been proposed to deal with these new models, and tool support extended in consequence. Last but not least, we have addressed adaptor implementation in BPEL.

## 7 Concluding Remarks

Software adaptation is a promising solution to compose in a non-intrusive way black-box services that contain incompatibilities in their interfaces. In this paper, we have presented our tool-supported techniques to generate adaptor protocols from interfaces of services described by signatures and protocols with value-passing, and an adaptation contract. Adaptor generation is completely automated and the resulting adaptor makes the whole system work correctly by solving protocol mismatches as well as value passing issues. Since our mechanisms are based on an encoding into the LOTOS process algebra, we take advantage of the existing CADP toolbox for LOTOS to verify the correctness of the contract. We have also shown with BPEL how our adaptors can be implemented. The main perspective of this work is to propose an assisted design approach to help and guide the architect in the construction of adaptation contracts.

**Acknowledgements.** This work has been partially supported by project “Pervasive Service cOmposition” (PERSO) funded by the French National Agency for Research (ANR-07-JCJC-0155-01), project TIN2008-05932 funded by the Spanish Ministry of Innovation and Science, and project P06-TIC2250 funded by the Andalusian local Government.

## References

1. Arnold, A.: Finite Transition Systems. International Series in Computer Science. Prentice-Hall, Englewood Cliffs (1994)
2. Autili, M., Inverardi, P., Navarra, A., Tivoli, M.: SYNTHESIS: A Tool for Automatically Assembling Correct and Distributed Component-based Systems. In: Proc. of ICSE 2007, pp. 784–787. IEEE Computer Society, Los Alamitos (2007)

3. Becker, S., Brogi, A., Gorton, I., Overhage, S., Romanovsky, A., Tivoli, M.: Towards an Engineering Approach to Component Adaptation. In: Reussner, R., Stafford, J.A., Szyperski, C. (eds.) *Architecting Systems with Trustworthy Components*. LNCS, vol. 3938, pp. 193–215. Springer, Heidelberg (2006)
4. Bracciali, A., Brogi, A., Canal, C.: A Formal Approach to Component Adaptation. *Journal of Systems and Software* 74(1), 45–54 (2005)
5. Brogi, A., Popescu, R.: Automated Generation of BPEL Adapters. In: Dan, A., Lamersdorf, W. (eds.) *ICSOC 2006*. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
6. Canal, C., Poizat, P., Salaün, G.: Model-Based Adaptation of Behavioural Mismatching Components. *IEEE Transactions on Software Engineering* 34(4), 546–563 (2008)
7. Dumas, M., Wang, K.W.S., Spork, M.L.: Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) *BPM 2006*. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
8. Fischer, M.J., Ladner, R.E.: Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences* 18(2), 194–211 (1979)
9. Foster, H., Uchitel, S., Kramer, J.: LTSA-WS: A Tool for Model-based Verification of Web Service Compositions and Choreography. In: *Proc. of ICSE 2006*, pp. 771–774. ACM Press, New York (2006)
10. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: *Proc. of WWW 2004*, pp. 621–630. ACM Press, New York (2004)
11. Garavel, H.: Open/Cæsar: An Open Software Architecture for Verification, Simulation, and Testing. In: Steffen, B. (ed.) *TACAS 1998*. LNCS, vol. 1384, pp. 68–84. Springer, Heidelberg (1998)
12. Garavel, H., Lang, F.: SVL: a Scripting Language for Compositional Verification. In: *Proc. of FORTE 2001, IFIP*, pp. 377–392. Kluwer Academic, Dordrecht (2001)
13. Garavel, H., Mateescu, R., Lang, F., Serwe, W.: CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In: Damm, W., Hermanns, H. (eds.) *CAV 2007*. LNCS, vol. 4590, pp. 158–163. Springer, Heidelberg (2007)
14. ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, ISO (1989)
15. Mateescu, R.: On-the-fly State Space Reductions for Weak Equivalences. In: *Proc. of FMICS 2005*, pp. 80–89. ACM Computer Society Press, New York (2005)
16. Mateescu, R.: CAESAR\_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *STTT Journal* 8(1), 37–56 (2006)
17. Mateescu, R., Poizat, P., Salaün, G.: Behavioral Adaptation of Component Compositions based on Process Algebra Encodings. In: *Proc. of ASE 2007*, pp. 385–388. IEEE Computer Society, Los Alamitos (2007)
18. Mateescu, R., Sighireanu, M.: Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Science of Computer Programming* 46(3), 255–281 (2003)
19. Moser, O., Rosenberg, F., Dustdar, S.: Non-Intrusive Monitoring and Adaptation for WS-BPEL. In: *Proc. of WWW 2008*, pp. 815–824 (2008)
20. Motahari-Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-Automated Adaptation of Service Interactions. In: *Proc. of WWW 2007*, pp. 993–1002 (2007)
21. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and Reasoning on Web Services using Process Algebra. *International Journal of Business Process Integration and Management* 1(2), 116–128 (2006)

# Automatic Workflow Graph Refactoring and Completion

Jussi Vanhatalo<sup>1,2</sup>, Hagen Völzer<sup>1</sup>, Frank Leymann<sup>2</sup>, and Simon Moser<sup>3</sup>

<sup>1</sup> IBM Zurich Research Laboratory, Switzerland  
{juv,hvo}@zurich.ibm.com

<sup>2</sup> Institute of Architecture of Application Systems, University of Stuttgart, Germany  
frank.leymann@iaas.uni-stuttgart.de

<sup>3</sup> IBM Böblingen Software Laboratory, Germany  
smoser@de.ibm.com

**Abstract.** Workflow graphs are used to model the control flow of business processes in various languages, e.g., BPMN, EPCs and UML activity diagrams. We present techniques for automatic workflow graph refactoring and completion. These techniques enable various use cases in modeling and runtime optimization. For example they allow us to complete a partial workflow graph, they provide local termination detection for workflow graphs with multiple ends, and they allow us to execute models containing OR-joins faster. Some of our techniques are based on workflow graph parsing and the Refined Process Structure Tree [10].

## 1 Introduction

A *workflow graph* shows the control flow of a business process similar to a flow chart as a directed graph. Figure 1(a) shows an example. Workflow graphs form the core of many specification languages, e.g., BPMN, EPCs and UML activity diagrams.

Different workflow graphs can model the same behavior, i.e., the same control flow. For example, the two workflow graphs in Figs. 1(a) and 1(b) model the same behavior. The workflow graph in Fig. 1(b) is *well-structured* in the sense that it consists of matching pairs of a node that splits the flow and a node that joins the flow. Well-structured workflow graphs are often preferred because they are easier to comprehend [9,2] and analyze [11], and can be represented as a regular expression.

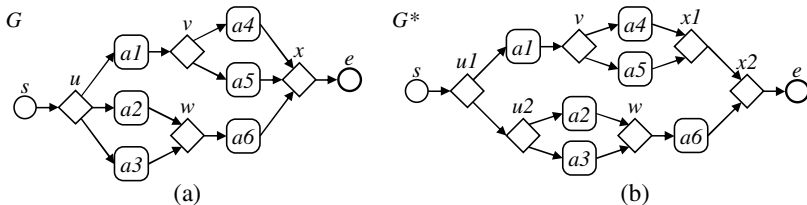


Fig. 1. (a) A workflow graph  $G$ , (b) A well-structured workflow graph  $G^*$

The workflow graph  $G$  can be transformed into the well-structured workflow graph  $G^*$  using local transformation rules that preserve the execution semantics. While those rules are known [7,8], it is not clear how to apply them to obtain  $G^*$  from  $G$  automatically. This is because a transformation rule can be applied to different parts of the workflow graph, which can lead to different refactoring results. For example, applying the same rule that transforms  $G$  into  $G^*$ , we can transform  $G$  also into  $G'$  in Fig. 2. The transformation rule is given in Sect. 3.2. We are not aware of any work that specifies a desired refactoring result and proposes a concrete algorithm that computes it.

In this paper, we propose such a definition and such an algorithm. Given a workflow graph  $G$ , the algorithm computes a *normal form*  $G^*$  of  $G$  that makes the structure of  $G$  more explicit—and, as a side-product, making it more well-structured. Our approach is based on the *normal* [11] and the *refined process structure trees* [10] of a workflow graph.

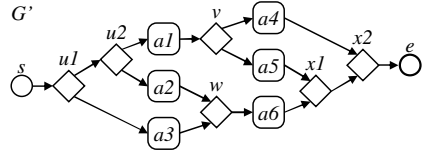


Fig. 2. Alternative transformation of  $G$

It is important that a workflow graph can be represented in an easily comprehensible form, as this makes it better accessible for users that may not be experienced in business process modeling. After all, workflow graphs are used for communicating business processes among different stakeholders, and not only among modeling experts. We hope that a workflow graph is, in general, easier to comprehend in our normal form than in its original form, as it is more well-structured.

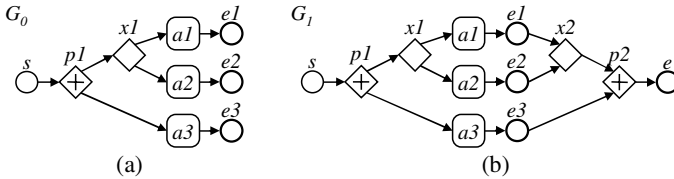


Fig. 3. (a) A workflow graph with multiple end nodes. (b) Its completion.

In the second part of the paper, we show how this refactoring technique can be used to compute a *completion* of a workflow graph. Figure 3 shows (a) a workflow graph with multiple end nodes and (b) its completion, which by definition has a unique end node. This has multiple use cases, which we discuss in Sect. 4. For example, it can be used to complete a partial workflow graph at modeling time [4], to provide local termination detection for workflow graphs with multiple ends, and to execute models containing OR-joins faster.

The refactoring technique efficiently computes a completion for many workflow graphs, but not for all. Sometimes a completion does not exist. We characterize these cases and also provide an algorithm that computes a completion in the general case when it exists. This algorithm is less efficient than the refactoring-based completion.

We review preliminary notions in Sect. 2. Then, we present our contributions: the refactoring technique in Sect. 3 and the completion technique in Sect. 4. Section 5

concludes this paper. We include short proofs of some theorems in this paper, whereas longer proofs of the other theorems can be found in a technical report [12].

## 2 Preliminaries

This section defines the basic preliminary notions of this paper, which include workflow graphs and their semantics, their extension by inclusive OR-gateways, and the soundness property for workflow graphs.

### 2.1 Workflow Graphs

A *workflow graph*  $G$  consists of a directed graph  $(V, E)$ , consisting of a set  $V$  of nodes, a set  $E \subseteq V \times V$  of edges, and a partial mapping  $\ell : V \rightarrow \{AND, XOR\}$  such that

1.  $\ell(x)$  is defined if and only if  $x$  has more than one incoming edge or more than one outgoing edge,
2. there is exactly one source and at least one sink,
3. the source has exactly one outgoing edge and each sink has exactly one incoming edge, and
4. every node is on a path from the source to some sink.

The source is also called the *start node*, a sink is called an *end node*,  $\ell(x)$  is called the *logic* of  $x$ . If the logic is AND or XOR, we call  $x$  a *gateway*; if  $x$  has no logic, then we call  $x$  a *task*. We use BPMN to depict workflow graphs, i.e., gateways are drawn as diamonds, where the symbol “+” inside stands for AND, whereas no decoration stands for XOR. Tasks are drawn as rectangles or circles. In particular, here start and end nodes are considered special tasks, which are always drawn as circles. A gateway that has more than one incoming edge and only one outgoing edge is also called a *join*, a gateway with more than one outgoing but only one incoming edge is also called a *split*. We say that an edge  $e$  is *incident* to a node  $n$  if  $e$  is incoming to  $n$  or outgoing from  $n$ .

The semantics of a workflow graph is, similarly to Petri nets, defined as a token game. A state of a workflow graph is represented by tokens on the edges of the graph. Let  $G = (V, E, \ell)$  be a workflow graph. A *state* of  $G$  is a mapping  $s : E \rightarrow \mathbb{N}$ , which assigns a natural number to each edge. When  $s(e) = k$ , we say that edge  $e$  carries  $k$  *tokens* in state  $s$ . The semantics of the various nodes is defined as usual. An AND-gateway removes one token from each of its ingoing edges and adds one token to each of its outgoing edges. An XOR-gateway nondeterministically chooses one of its incoming edges on which there is at least one token, removes one token from that edge, then nondeterministically chooses one of its outgoing edges, and adds one token to that outgoing edge. As usual, we abstract from the data that controls the flow in XOR-gateways, hence the nondeterministic choice.

To be more precise, let  $s$  and  $s'$  be two states and  $x$  a node that is neither a start nor an end node. We write  $s \xrightarrow{x} s'$  when  $s$  changes to  $s'$  by executing  $x$ . We have  $s \xrightarrow{x} s'$  if

1.  $\ell(x) = AND$  or the logic of  $x$  is undefined, and
 
$$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } x, \\ s(e) + 1 & e \text{ is an outgoing edge of } x, \\ s(e) & \text{otherwise.} \end{cases}$$

2.  $\ell(x) = XOR$ , and there exists an incoming edge  $e'$  and an outgoing edge  $e''$  of  $x$  such that

$$s'(e) = \begin{cases} s(e) - 1 & e = e', \\ s(e) + 1 & e = e'', \\ s(e) & \text{otherwise.} \end{cases}$$

The *initial state* of  $G$  is the state where there is exactly one token on the unique outgoing edge of the start node and no token anywhere else. Node  $x$  is said to be *activated* in a state  $s$  if there exists a state  $s'$  such that  $s \xrightarrow{x} s'$ . A state  $s'$  is *reachable from* a state  $s$ , denoted  $s \xrightarrow{*} s'$ , if there exists a (possibly empty) finite sequence  $s_0 \xrightarrow{x_1} s_1 \dots s_{k-1} \xrightarrow{x_k} s_k$  such that  $s_0 = s$  and  $s_k = s'$ . A state is a *reachable state* of  $G$  if it is reachable from the initial state of  $G$ .

## 2.2 Inclusive OR-Gateways

A *generalized workflow graph* is a workflow graph in which a gateway  $x$  may also have OR-logic (*inclusive OR*), i.e.,  $\ell(x) = OR$ . OR-gateways are drawn as diamonds with a circle inside. An OR-gateway has non-local join behavior, which is difficult to define if there is a cycle in the graph that contains an OR-join. As the semantics for the OR-join is not settled in that case, we do not consider that case. So, in the following we assume that a generalized workflow graph does not contain a cycle that contains an OR-gateway that has more than one incoming edge.

The OR-gateway behaves as follows. It is activated if for each incoming edge  $e'$  of the gateway that carries no token, and for each edge  $e''$  of the graph that carries a token, there is no directed path from  $e''$  to  $e'$ . When it executes, it consumes a token from each incoming edge that carries a token and produces a token for each edge of a nonempty subset of its outgoing edges. That subset is chosen nondeterministically. More precisely, we also have  $s \xrightarrow{x} s'$  if

3. -  $\ell(x) = OR$ ,  
 - for each edge  $e'' \in E$  and each incoming edge  $e'$  of  $x$  such that,  $s(e'') \geq 1$  and  $s(e') = 0$ , there is no path from  $e''$  to  $e'$  in the graph, and  
 - there exists a nonempty set  $F$  of outgoing edges of  $x$  such that
- $$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } x \text{ such that } s(e) \geq 1, \\ s(e) + 1 & e \in F, \\ s(e) & \text{otherwise.} \end{cases}$$

## 2.3 The Soundness Property

We now define what we understand by a “well-behaved” (generalized) workflow graph. A *final state* is a reachable state  $s$  of  $G$  that has no successor state, i.e.,  $s$  activates no node. A final state is a *deadlock* if it contains a token on some edge that is not an incoming edge of an end node. (It follows that it must then be an incoming edge of a join.) A reachable state  $s$  contains a *lack of synchronization* if there is an edge  $e$  such that  $s(e) > 1$ . A (generalized) workflow graph is *sound* if it contains neither a deadlock nor a lack of synchronization.



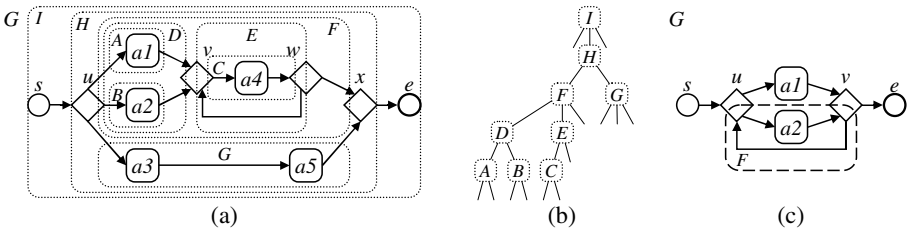
A deadlock is clearly undesired. Absence of lack of synchronization is a valuable design principle as it rules out that a task is executed concurrently to itself (“uncontrolled auto-concurrency”). Multiple concurrent instances of tasks can be modeled explicitly with dedicated constructs (e.g. “multiple instance activities” in BPEL and BPMN). Soundness is necessary for translating a (generalized) workflow graph to BPEL in a structured way, but has also an independent motivation: For workflow graphs with a unique end node and without OR-gateways, it is equivalent with the traditional definition of soundness (cf. [9][11]).

### 3 Automatic Refactoring of Workflow Graphs

In this section, we present our technique that automatically refactors a workflow graph into a normal form that makes the structure of the workflow graph explicit. By structure we mean the decomposition of the workflow graph into logically atomic parts, which we call *fragments*. We use two alternative ways to do this decomposition, the *normal* [11] and the *refined process structure tree* [10]. We review the definitions of these process structure trees in Sect. 3.1. In Sect. 3.2 we present our novel refactoring technique.

#### 3.1 The Refined and the Normal Process Structure Trees

Let  $G = (V, E, \ell)$  be a (generalized) workflow graph. We assume that  $G$  has a unique end node. A detailed discussion on how to extend a workflow graph with multiple end nodes to a workflow graph with a unique end node is given in Sect. 4

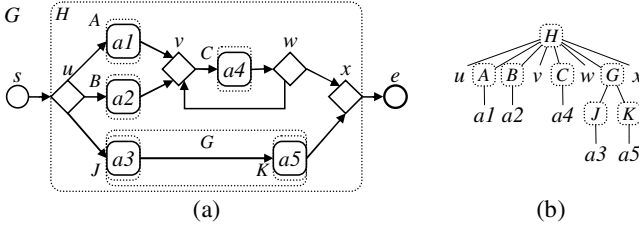


**Fig. 4.** (a) The canonical R-fragments of a workflow graph  $G$ . (b) The refined process structure tree of  $G$ . (c)  $F$  is not an R-fragment of  $G$ .

Figure 4(a) shows a workflow graph  $G$  and its decomposition into *canonical R-fragments*. An R-fragment is a connected subgraph with a unique entry and a unique exit node. The canonical R-fragments form a hierarchy, which can be represented as a tree. This tree, called the *refined process structure tree* (RPST), is shown in Fig. 4(b). These notions can be formally be defined as follows.

For a set  $F \subseteq E$  of edges, let  $V_F$  denote the set of nodes that are incident to some edge in  $F$  and let  $G_F = (V_F, F)$ . We say  $G_F$  is *formed by*  $F$ .

Let  $F \subseteq E$  be a set of edges such that  $G_F$  is a connected subgraph of  $G$ . A node  $v \in V_F$  is a *boundary node* of  $F$  if it is the source or sink node of  $G$ , or if there exist edges  $e \in F$  and  $e' \in E \setminus F$  such that  $v$  is incident to  $e$  and  $e'$ . A boundary node  $v$  is



**Fig. 5.** (a) The canonical N-fragments of a workflow graph  $G$ . (b) The NPST of  $G$ .

an *entry* of  $F$  if no incoming edge of  $v$  is in  $F$  or if all outgoing edges of  $v$  are in  $F$ . A boundary node  $v$  is an *exit* of  $F$  if all incoming edges of  $v$  are in  $F$  or if no outgoing edge of  $v$  is in  $F$ .  $F$  is called an *R-fragment* of  $G$  if it has exactly two boundary nodes, an entry and an exit.

Figure 4(a) shows examples of R-fragments, which are indicated dotted boxes. They contain all those edges that are either inside the box or cross the boundary of the box. Thus, the box  $D$  denotes the R-fragment  $D = \{(u, a1), (a1, v), (u, a2), (a2, v)\}$ . Node  $u$  is the entry and  $v$  is the exit of  $D$ .  $E = \{(v, a4), (a4, w), (w, v)\}$  is an R-fragment with entry  $v$  and exit  $w$ . In Fig. 4(c),  $F = \{(u, a2), (a2, u), (v, u)\}$  has two boundary nodes,  $u$  and  $v$ , but neither of them is an entry or an exit of  $F$ . Thus,  $F$  is not an R-fragment.

An R-fragment  $F$  is *canonical* if it does not overlap with any other R-fragment, that is, for each R-fragment  $F'$  of  $G$ , we have  $F \subseteq F'$  or  $F' \subseteq F$  or  $F \cap F' = \emptyset$ . It follows that canonical R-fragments do not overlap with each other and hence form a hierarchy, which is represented as the *refined process structure tree* (RPST) of  $G$ . Note that each leaf node of the RPST represents an edge of the workflow graph, as each edge forms an R-fragment. The boundary nodes of this R-fragment are the two nodes that this edge connects. Vanhatalo, Völzer and Koehler [10] show how the RPST can be computed in linear time.

Figure 5(a) shows an alternative decomposition of the same workflow graph into *N-fragments*. An N-fragment has unique entry and exit *edges* (as opposed to nodes). The corresponding tree, shown in Fig. 5(b), is called the *normal process structure tree* (NPST).

Let  $G = (V, E, \ell)$  be a workflow graph. An *N-fragment*  $G_F = (V', E')$  is a nonempty connected subgraph of  $G$  such that there exist edges  $e, e' \in E$  with  $E \cap ((V \setminus V') \times V') = \{e\}$  and  $E \cap (V' \times (V \setminus V')) = \{e'\}$ ;  $e$  and  $e'$  are called the *entry* and the *exit* edge of  $G_F$ , respectively. An N-fragment is *canonical* if it does not overlap with any other N-fragment. The tree representing the hierarchy of canonical N-fragments is called the *normal process structure tree* (NPST). It can also be computed in linear time [5111].

We define special kinds of R-fragments and N-fragments as follows. An R-fragment  $F$  is *trivial* if  $F$  has exactly one edge. The union of two R-fragments  $F, F'$  is an *R-sequence* if the exit node  $u$  of  $F$  is the entry node of  $F'$ , and each edge incident to  $u$  is in  $F \cup F'$ . An N-fragment  $F$  is *trivial* if the entry and the exit edge of  $F$  are incident to the same node of  $F$ . An N-fragment  $F$  is an *N-sequence* if  $F$  is the union of two N-fragments  $F', F''$  such that the exit edge of  $F'$  is the entry edge of  $F''$ . An R-fragment (N-fragment) is *proper* if it is neither trivial nor an R-sequence (N-sequence). A node

$u$  is a *child* of a canonical N-fragment  $F$  if  $F$  is the smallest canonical N-fragment that contains  $u$ .

While the RPST exhibits more structure than the NPST, the NPST shows the structure more explicitly, in a less dense, more readable, form. The NPST produces a decomposition of the edges and nodes, defining a home fragment for each node and each edge—whereas the RPST produces a decomposition of edges only, while nodes may be shared between adjacent fragments. In the following, we show how to compute the normal form of a workflow graph  $G$ , which has the best of both worlds: It has all the structure of the RPST of  $G$ , but shows it explicitly in the more readable form of the NPST. The normal form is also more well-structured than the original workflow graph.

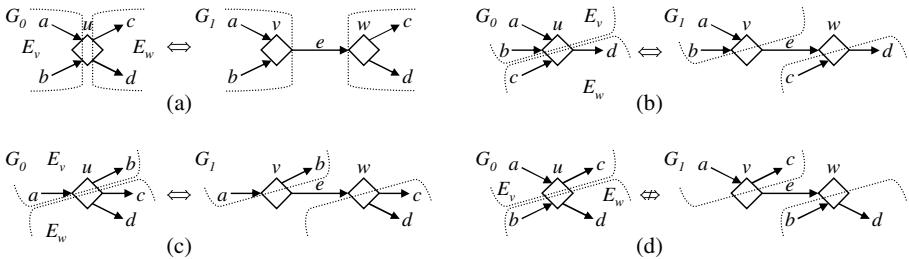
### 3.2 Refactoring Based on the RPST

Let  $G$  be a (generalized) workflow graph with a unique end node. We want to transform  $G$ , maintaining its structure given by its RPST but showing it more explicitly in form of N-fragments. Some R-fragments can be considered as N-fragments. We call them *normal*:

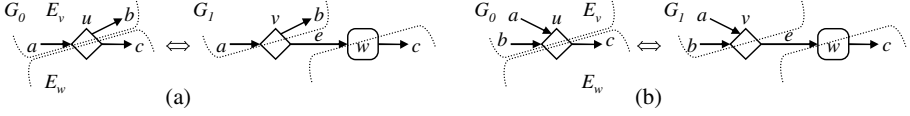
**Definition 1 (Normal R-fragment, normal (generalized) workflow graph).** A proper canonical R-fragment  $F$  is normal if exactly one edge outside  $F$  is incident to the entry of  $F$  and exactly one edge outside  $F$  is incident to the exit of  $F$ . A (generalized) workflow graph  $G$  is normal if every proper canonical R-fragment  $F$  is normal.

Normal R-fragments can be obtained by splitting nodes. Figures 6(a)-(c) show three examples of a valid expansion  $(G_0, G_1)$  that splits a node  $u$  into two nodes  $v$  and  $w$  in a way that it preserves the execution semantics of the workflow graph. Subfigure (a) shows the splitting of a node into a join and a split. Subfigures (b) and (c) show a splitting that separates different inputs and outputs of the node respectively. The splitting shown in subfigure (d) is invalid as the simultaneous separation of inputs and outputs does not preserve execution semantics. (The original path from  $b$  to  $c$  gets lost.) The three valid cases are instances of a single rule that splits a node into two nodes  $v, w$ :

**Definition 2 (Valid expansion).** Let  $G_i = (V_i, E_i, \ell_i), i = 0, 1$  be two (generalized) workflow graphs. The pair  $(G_0, G_1)$  is a valid expansion if there exists two nodes  $v, w \in V_1$  and a surjective mapping  $\phi : V_1 \rightarrow V_0$  such that



**Fig. 6.** (a), (b), (c) Valid expansions that split a node  $u$  into nodes  $v, w$ . (d) An invalid expansion.



**Fig. 7.** Two examples of undesired expansions that split node  $u$  into nodes  $v$  and  $w$

- $(v, w) \in E_1$  and  $(w, v) \notin E_1$ ,
- $(v, x) \in E_1$  and  $(y, w) \in E_1 \Rightarrow x = w$  or  $y = v$ ,
- $\phi(x) = \phi(y) \Leftrightarrow x = y$  or  $\{x, y\} = \{v, w\}$ ,
- $(\phi(x), \phi(y)) \in E_0 \Leftrightarrow (x, y) \in E_1$  and  $(x, y) \neq (v, w)$ ,
- $\ell_0(\phi(x)) = \ell_1(x)$ , or  $\ell_1(x)$  is undefined.

We define two parameters for a valid expansion: 1. the node  $u = \phi(v) = \phi(w)$  and 2. a partition of the edges  $E_u$  that are incident to  $u$  into two sets  $E_u = E_v \cup E_w$ . We define

$$E_v = \{(u, \phi(y)) \mid (v, y) \in E_1, y \neq w\} \cup \{(\phi(x), u) \mid (x, v) \in E_1, x \neq w\} \quad (1)$$

$E_w$  can be defined similarly or as  $E_w = E_u \setminus E_v$ .

Sadiq and Orłowska [8] have shown for workflow graphs that if  $(G_0, G_1)$  is a valid expansion that splits a node  $u$  into nodes  $v$  and  $w$ , and the logic of  $u$  is AND or XOR, then  $G_0$  and  $G_1$  have the same behavior. An analogous result is also known from Petri nets [7]. The behavior is also the same if the logic of  $u$  is OR and  $u$  is not in a cycle.

Some valid expansions create redundant nodes that we want to exclude from our refactoring technique. We call these *undesired expansions*. A *desired expansion* splits a gateway into two gateways, whereas an *undesired expansion* creates at least one task. Figure 7 shows two examples of undesired expansions. Each of these valid expansions is undesired, because node  $w$  has only one incoming and only one outgoing edge.

**Definition 3 (Desired expansion, undesired expansion).** Let  $G_i = (V_i, E_i, \ell_i), i = 0, 1$  be two (generalized) workflow graphs and  $v, w \in V_1$  be distinct nodes such that  $(G_0, G_1)$  is a valid expansion and  $\phi(v) = \phi(w)$ . The pair  $(G_0, G_1)$  is a desired expansion if

- $v$  has at least two incoming edges or at least two outgoing edges, and
- $w$  has at least two incoming edges or at least two outgoing edges.

Otherwise  $(G_0, G_1)$  is an undesired expansion.

The valid expansions in Fig. 6 are also desired expansions. Desired expansions restrict the application of node splitting. However, application of the desired expansion can still lead to different results, as shown by the example in Sect. 4. As we want to maintain the structure of the graph, we apply the expansion only based on the R-fragments. This will lead to a unique result.

**Definition 4 (F-based expansion of node  $u$ ).** Let  $(G_0, G_1)$  be a desired expansion with parameters  $u, E_v$  and  $E_w$  as in Def. 2 and let  $F$  be an R-fragment of  $G_0$ . Furthermore, let  $E_u$  denote the set of edges that are incident to  $u$ . We say that the expansion is F-based if either  $u$  is the entry of  $F$  and  $E_w = F \cap E_u$ , or  $u$  is the exit of  $F$  and  $E_v = F \cap E_u$ .

The fragment-based expansions can be applied repeatedly until we obtain a normal generalized workflow graph.

**Definition 5 (Expansion).** Let  $G_0$  and  $G_n$  be (generalized) workflow graphs.  $G_n$  is an expansion of  $G_0$  if there is a sequence  $G_0, \dots, G_n$  of (generalized) workflow graphs such that  $(G_i, G_{i+1})$  is a fragment-based expansion, when  $0 \leq i < n$ .

Although a given workflow graph  $G$  allows different sequences of fragment-based expansions, we nevertheless obtain a unique result, which we call the *expanded normal form* of  $G$ , denoted as  $G^*$ .

**Theorem 1.** For every (generalized) workflow graph  $G$ , there exists a unique (generalized) workflow graph  $G^*$  such that  $G^*$  is an expansion of  $G$  and  $G^*$  is normal.

*Proof.* Each  $F$ -based expansion replaces a fragment that is not normal by a sequence of a trivial fragment and  $F$ . It is a local change to the original generalized workflow graph  $G$  that preserves the structure of the RPST. This feature, which was named *modularity* of the RPST, was proved earlier [10]. In fact, if we do not consider trivial fragments and sequences, the RPST stays the same after the expansion. Because expansions can be considered as modular replacements that do not change the RPST up to trivial fragments and sequences, and because fragments are either nested or disjoint, then all fragment-based expansions that can be applied to the original graph  $G$  are mutually independent, i.e. they can be applied in any order to obtain the same result. (Note however, that a particular expansion can be based on different fragments.)

Furthermore, an  $F$ -based expansion can only be applied if either the entry or the exit of  $F$  violates the normality constraint and if  $F$  is neither trivial nor a sequence. An  $F$ -based expansion removes such a violation of a normality constraint and cannot introduce a new one. It follows that we arrive at a unique normal generalized workflow graph after any sequence of all the expansions that are possible in the original graph  $G$ .  $\square$

It is clear from the proof of Thm. 1 that the normal form  $G^*$  arises as the maximal expansion of  $G$ . Therefore, Alg. 1 computes  $G^*$ . It can be implemented in linear time to the number of edges of  $G$ .

**Theorem 2.** Let  $G$  be a (generalized) workflow graph. Algorithm 1 computes the expanded normal form of  $G$ .

---

**Algorithm 1.** Computes the expanded normal form of a generalized workflow graph  $G$

---

**computeExpandedNormalForm( $G$ )**

  Compute the RPST  $T$  of  $G$ , and a list  $L$  of proper canonical  $R$ -fragments that are not normal.

**while**  $L$  has a proper canonical  $R$ -fragment  $F$  such that  $F$  is not normal **do**

    Transform  $G$  into  $G'$  based on an  $F$ -based expansion  $(G, G')$ .

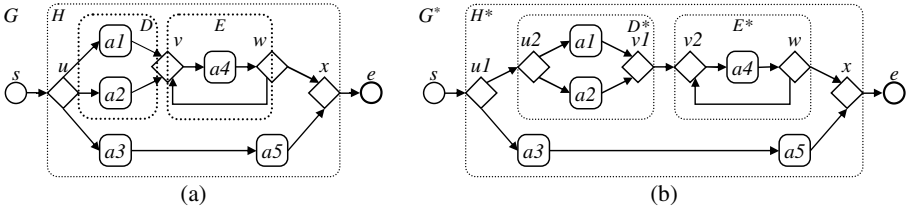
    Locally update  $T$  to correspond to the RPST of this updated  $G$ .

**if**  $F$  is normal, **then** remove  $F$  from  $L$ .

**return**  $G$

---

Next we show that the NPST and the RPST coincide if  $G$  is normal. As  $R$ -fragments and  $N$ -fragments are different objects, there are subtle differences in the trivial fragments. However, the main structure that is represented by the proper fragments coincides:



**Fig. 8.** (a) A workflow graph  $G$ . (b) The expanded normal form  $G^*$  of  $G$ .

**Theorem 3.** Let  $G$  be a normal (generalized) workflow graph,  $F$  a set of edges and  $G_F$  the subgraph formed by  $F$ .  $F$  is a proper canonical R-fragment of  $G$  if and only if  $G_F$  is a proper canonical N-fragment of  $G$ .

*Proof.* Presented in a technical report [12].

Figure 8(a) shows a workflow graph  $G$  and its proper canonical R-fragments. The R-fragment  $H$  is normal, whereas  $D$  and  $E$  are not. Thus, we transform  $G$  into its extended normal form  $G^*$  shown in Fig. 8(b) through  $D$ -based and  $E$ -based expansions. It is possible to transform  $G$  with  $D$ -based expansion of  $u$  and  $v$ , and an  $E$ -based expansion of  $v$ . The  $D$ -based and  $E$ -based expansions of  $v$  split  $v$  into nodes  $v1$  and  $v2$  the same way. Thus, we can obtain  $G^*$  from  $G$  through two  $D$ -based expansions, or through one  $E$ -based and one  $D$ -based expansion. The proper canonical R-fragments of  $G^*$  are also the proper canonical N-fragments of  $G^*$ .  $G^*$  has two more proper canonical N-fragments ( $D^*$  and  $E^*$ ) than  $G$ .

An N-fragment  $F$  is *well-structured* (c.f. [6][1]) if  $F$  is trivial, an N-sequence, or if  $F$  has exactly two gateways as children, a split and a join  $j$ , that have the same logic and if the entry edge of  $F$  is incoming to  $j$ , then  $j$  is an XOR-join. Note that a well-structured fragment cannot be further refactored. However, if a fragment is not well-structured it may become well-structured as shown in the examples of Figs. 1 and 8. In this sense, the normal form of  $G$  is “more well-structured” than  $G$ . Note that we only use local refactorings. It is known [6] that some workflow graphs can only be transformed into well-structured graphs through non-local transformations, whereas there are also workflow graphs that have no well-structured equivalent.

## 4 Automatic Completion of Workflow Graphs

In this section, we show how the refactoring technique of Sect. 3 can be used to compute a *completion* of a workflow graph.

**Definition 6 (Completion).** Let  $G = (V, E, \ell)$  be a sound workflow graph (sound generalized workflow graph) and  $G' = (V', E', \ell')$  a workflow graph (generalized workflow graph).  $G'$  is called a completion of  $G$  if

1.  $G$  is a subgraph of  $G'$  such that if an edge  $e \in E' \setminus E$  is incident to some node in  $x \in V$ , then  $x$  is an end node of  $G$  and  $e$  is outgoing from  $x$ ,
2.  $G'$  has a unique end node,
3.  $G'$  is sound.

A completion of a generalized workflow graph is easy to construct. We just add an OR-join  $j$  and an end node  $e$ , and connect each original end node to  $j$  and  $j$  to  $e$ . Figure 9 illustrates this.

**Proposition 1.** *The construction shown in Fig. 9 defines a completion of a generalized workflow graph.*

*Proof.* We have to prove soundness of  $G'$ . We claim that the final OR-join can fire only if  $G$  has no more tokens. Suppose the contrary. Then there is a state  $s$  that activates the OR-join and there is some token inside  $G$ , say on edge  $e$ . As  $G$  has no deadlock, we can move the token from  $e$  to some end node of  $G$ . As  $G$  has no lack of synchronization, this end node was unmarked in  $s$ . It follows that in state  $s$ , there is a path from some token to an unmarked incoming edge of the OR-join. It follows that the OR-join is not activated in  $s$ , contradicting our supposition. The claim of the proposition follows now directly.  $\square$

This simple completion can be used to compute a translation from generalized workflow graphs (e.g. BPMN diagrams) with multiple end nodes to BPEL using the refined process structure tree [10], which needs a generalized workflow graph with a unique start<sup>1</sup> and a unique end node as input. However, there are various use cases where we want a completion without using OR-gateways:

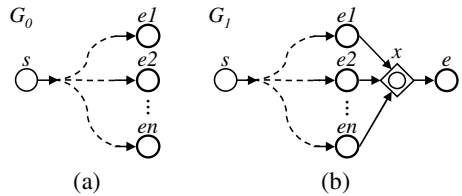


Fig. 9.  $G_1$  is the simple completion of  $G_0$

- A user modeling a process has drawn a part of a diagram where she opened a number of parallel and alternative paths, i.e., she used a combination of AND-splits and XOR-splits, possibly also some AND-joins or XOR-joins. Now the user does not know what logic, i.e., combination of AND-joins and XOR-joins to use to close a given set of paths correctly. In that situation, an OR-join could also be used to close these paths. However, the OR-join may not be available in the language chosen or may be considered too expensive to execute (see next use case). Gschwind et al. [4] present this use case described above, but do not provide a technical solution.
- An OR-join was used to close a particular set of paths. In our restricted setting, the evaluation of the OR-join at runtime requires that the whole graph preceding the OR-join be checked for tokens. If we compute a completion of the graph preceding the OR-join, we can replace the OR-join with a combination of merges and joins, all of which can be executed locally. Removing OR-joins also allows us to translate the generalized workflow graph to a Petri net in a simple way [9]. (Petri net transitions have a local semantics.) A translation to Petri nets may not only be useful as an intermediate language between two different business process specification languages, but also to apply some of the various analysis techniques and tools available for Petri nets.

<sup>1</sup> In most languages, multiple start nodes stand for either an implicit AND-split or an implicit XOR-split. Therefore, we restrict to workflow graphs having exactly one start node.

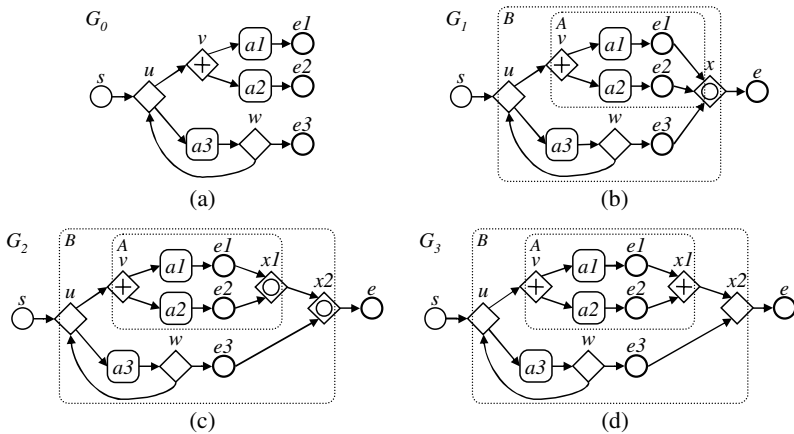
- Dead Path Elimination of BPEL [3] is a way to simulate the OR-join by gateways that can be executed locally. There, *dead tokens* are sent along those paths that are not taken. An OR-join can then wait for a token, dead or alive, on each incoming edge before it executes. If we can replace OR-joins by merges and joins, dead path elimination can be switched off, saving the overhead of sending, propagating and synchronizing dead tokens along potentially long paths.
- Checking whether a process has terminated also requires checking the entire generalized workflow graph for tokens. A sound generalized workflow graph with a unique end node will, however, signal termination through a single token on the unique end node. Thus, a completion at compile time provides a local termination detection at runtime.

In the general case, a completion may be difficult to compute or not even exist. In the following section, we show how to efficiently compute a completion based on the refactoring technique from Sect. 3 for many cases. In Sect. 4.2 we characterize for which workflow graphs a completion exists. Finally, in Sect. 4.3, we discuss how to compute a completion in the general case where it exists.

#### 4.1 Completion by Refactoring

Let  $G$  be a sound generalized workflow graph with multiple end nodes. We first complete  $G$  using the simple OR-join completion described above. This adds a unique end node, which allows us to compute the RPST for the completed graph. We obtain one or more proper canonical R-fragments that contain the added OR-join of  $G$  as exit. Such an R-fragment is called an *end fragment* of  $G$ . Figures 10(a) & (b) show an example of a workflow graph  $G_0$  and its simple completion  $G_1$  that has two end fragments  $A$  and  $B$ .

We then split the final OR-join into several OR-joins, one per end fragment according to the refactoring technique presented in Sect. 3.2. This preserves behavior as stated in



**Fig. 10.** Example of a completion by refactoring. The refactoring step produces  $G_2$  from  $G_1$ .



Sect. 3.2 and therefore also soundness. We refactor the end fragments until they are normal, and thus also N-fragments. Figure 10(c) shows an example of the resulting expansion  $G_2$  of  $G_1$ . Now we can replace an OR-join  $j$  by an XOR-join if the fragment  $F$  of  $j$  is *sequential*, that is, if  $F$  has no AND-split and no OR-split as a child. On the other hand, an OR-join  $j$  can be replaced by an AND-join if  $F$  of  $j$  is *deterministic*, that is,  $F$  has no XOR-split and no OR-split as a child. This replacement preserves soundness.

**Theorem 4.** *Let  $F$  be an N-fragment of a sound generalized workflow graph  $G$ , and an OR-join  $j$  be a child of  $F$ .*

1. *If  $F$  has neither an XOR-split nor an OR-split as children and  $j$  is replaced with an AND-join, then the resulting generalized workflow graph  $G'$  is sound.*
2. *If  $F$  has neither an AND-split nor an OR-split as children and  $j$  is replaced with an XOR-join then, the resulting generalized workflow graph  $G'$  is sound.*

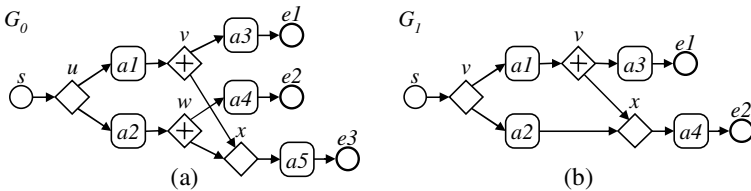
*Proof.* Presented in a technical report [12].

Applying this theorem repeatedly allows us to replace all OR-joins in such a fragment  $F$  with either 1) AND-joins or 2) XOR-joins. Figure 10(d) shows an example of a completion  $G_3$  of  $G_0$  obtained with our technique. Note that this completion technique requires only linear time. After a review of more than 150 sound workflow graphs created from realistic business processes, we believe that in practice most workflow graphs can be completed using this fast technique. The power of this technique stems from the fact that it abstracts from the interior of subfragments. For example, fragment  $B$  in Fig. 10(c) is sequential, although it contains a subfragment with concurrency.

Figure 11(a) shows a workflow graph that can be completed, but not with the fast technique presented here. We discuss these cases in the next section. It is clear that this technique can also replace OR-joins in the middle of a generalized workflow graph, provided that we restrict its application to settings where an OR-join is not in a cycle.

### 4.2 Existence of Completions

In this section, we present a technique that computes a completion for each workflow graph that has a completion, and we characterize for which workflow graphs a completion exists. Let  $G$  be a sound workflow graph. As  $G$  is sound, we can represent a final state as a set of end nodes. Let  $\mathcal{F}$  be the set of final states of  $G$ , presented in this way.



**Fig. 11.** Two workflow graphs. (a)  $G_0$  has a completion and (b)  $G_1$  has no completion.

**Definition 7.** Two distinct end nodes  $x, y$  are mutually exclusive if there is no final state  $M \in \mathcal{F}$  such that  $x \in M$  and  $y \in M$ . A test is a set  $T$  of pairwise mutually exclusive end nodes such that for each final state  $M$ ,  $M \cap T \neq \emptyset$ .

Figure 11(a) shows a workflow graph with final states  $\mathcal{F} = \{\{e1, e3\}, \{e2, e3\}\}$ . There are two tests,  $T_1 = \{e1, e2\}$  and  $T_2 = \{e3\}$ . Figure 11(b) shows a workflow graph with final states  $\mathcal{F} = \{\{e1, e2\}, \{e2\}\}$ . There is only one test  $T_3 = \{e2\}$ .

**Theorem 5.** A sound workflow graph has a completion if and only if each end node belongs to a test.

*Proof.* Presented in a technical report [12].

This completion can be constructed as shown in Fig. 12. We extend a workflow graph  $G$  by creating an XOR-join for each test of  $G$  and one final AND-join that is connected to the unique end node. Each end node  $x$  is connected to those XOR-joins that correspond to a test  $T$  such that  $x \in T$ . If an end node has to be connected to more than one XOR-join, an AND-split is inserted after the end node. All XOR-joins are connected to the final AND-join.

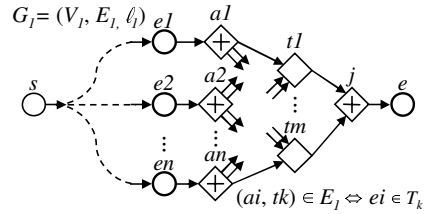


Fig. 12. General completion

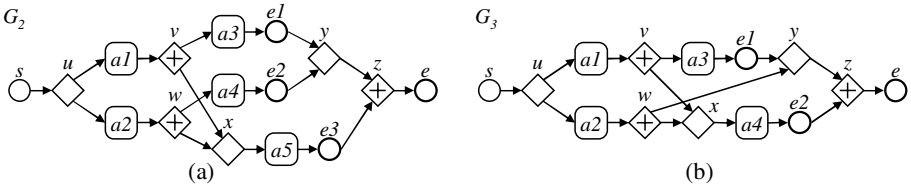


Fig. 13. (a) Completion of  $G_0$  from Fig. 11(a). (b) Completion of  $G_1$  from Fig. 11(b) after adding AND-split  $w$  in the middle of the graph.

Figure 13(a) shows the completion of  $G_0$  from Fig. 11. Note that gateways of completion that would have only a single incoming and a single outgoing edge can be omitted. Node  $y$  corresponds to the test  $\{e1, e2\}$  whereas the node created by the test  $\{e3\}$  was omitted. Also no AND-split was necessary.

$G_1$  in Fig. 11(b) does not have a completion because  $e1$  is not in any test. Note that it is impossible to obtain a completion if there exist two final states  $M$  and  $M'$  such that  $M \subseteq M'$ . Figure 13(b) shows that it is nevertheless possible to “complete”  $G_1$  in a more liberal sense; AND-split  $w$  added in the middle of  $G_1$  allows us to complete  $G_1$ .

### 4.3 Computation of the Completion in the General Case

The completion technique above requires computing the final states of the workflow graph, of which there can be exponentially many. The computation can be done by a

simple state-space exploration. However, using the refactoring in Sect. 3.2 first, allows us to restrict the completion to single end fragments, that is, we also have to compute the state space only for individual end fragments, which are typically much smaller. We transformed our library of more than 150 industrial process models into workflow graphs having, on average, 57 edges. The largest has 203 edges and more than 100,000 states. However, those fragments that are neither sequential nor deterministic have at most 38 states.

Once the final states have been computed, a suitable set of tests can be computed. Sometimes a simple set of tests exists that can be computed in a simple way: An end node  $x$  is called an *identifier* of a final state  $M$  if  $x$  is contained in  $M$  but not in any other final state. Assume every state  $M$  has an identifier  $x_M$ . Let  $y$  be an end node. Then  $T_y = \{x_M \mid M \in \mathcal{F}, y \notin M\} \cup \{y\}$  is a test containing  $y$ . So, in that special case, which is easy to check, we get a simple set of tests that suffices.

## 5 Conclusion

This paper presents two new techniques for generalized workflow graphs—an RPST-based refactoring technique that renders their structure more explicit, and a completion technique that builds on this refactoring technique. We also provide a characterization of workflow graphs that have a completion.

Only few related papers on workflow graph refactoring and completion exist and these papers, having different focus, are only loosely related. Sadiq and Orłowska [8] transform a workflow graph preserving its behavior to analyze its soundness. Eder et al. [2] define equivalence of workflow graphs through rewriting rules. Zhang and D’Hollander [13] use hammocks to structure flow graphs of sequential programs, which can all be made well-structured in the absence of concurrency. A hammock is a special case of an R-fragment, but their technique to restructure hammocks differs from ours. Well-structuredness makes workflow graphs more readable [9,2]. Gschwind et al. [4] identify a use case for a completion technique, but do not provide a solution to solve it.

We believe that our refactoring-based completion technique is also useful for removing OR-joins that occur in cycles. Figure 14 shows an example. Note that the refactoring produces a well-structured workflow graph. A formal treatment of these examples would, however, exceed the scope of this paper as it requires a semantics for OR-joins in cycles. In EPCs, our refactoring-based technique can also provide a completion in the beginning of a graph, which describes the start node combinations that lead into a sound execution.

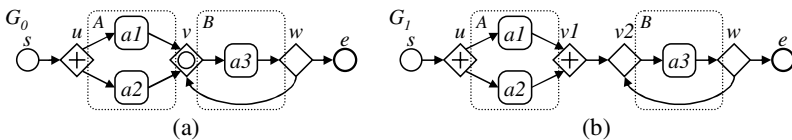


Fig. 14. Separating sequential cyclic fragments from concurrent fragments

**Acknowledgments.** The work published in this article was partially supported by the SUPER project (<http://www.ip-super.org/>) under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

## References

1. Ananian, C.S.: The static single information form. Master's thesis, Massachusetts Institute of Technology (September 1999)
2. Eder, J., Gruber, W., Pichler, H.: Transforming workflow graphs. In: INTEROP-ESA 2005, pp. 203–214 (2005)
3. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Org. (2007)
4. Gschwind, T., Koehler, J., Wong, J.: Applying patterns during business process modeling. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 4–19. Springer, Heidelberg (2008)
5. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: PLDI 1994, pp. 171–185. ACM, New York (1994)
6. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On structured workflow modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
7. Murata, T.: Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4), 541–580 (1989)
8. Sadiq, W., Orłowska, M.E.: Analyzing process models using graph reduction techniques. *Inf. Syst.* 25(2), 117–134 (2000)
9. van der Aalst, W.M.P., Hirschsall, A., Verbeek, H.M.W.: An alternative way to analyze workflow graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)
10. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008)
11. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
12. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic workflow graph refactoring and completion. IBM Research Report RZ 3715 (2008)
13. Zhang, F., D'Hollander, E.H.: Using hammock graphs to structure programs. *IEEE Trans. Softw. Eng.* 30(4), 231–245 (2004)

# Authorization and User Failure Resiliency for WS-BPEL Business Processes

Federica Paci<sup>1</sup>, Rodolfo Ferrini<sup>2</sup>, Yuqing Sun<sup>3</sup>, and Elisa Bertino<sup>1</sup>

<sup>1</sup> Cerias and Computer Science Department, Purdue University  
`{paci,bertino}@cs.purdue.edu`

<sup>2</sup> Department of Computer Science, University of Bologna  
`ferrini@csr.unibo.it`

<sup>3</sup> School of Computer Science and Technology (SCST), Shandong University  
`sun_yuqing@sdu.edu.cn`

**Abstract.** We investigate the problem of WS-BPEL processes *resiliency* in RBAC-WS-BPEL, an authorization model for WS-BPEL that supports the specification of authorizations for the execution of WS-BPEL process activities by roles and users and authorization constraints, such as separation and binding of duty. The goal of resiliency is to guarantee that even if some users becomes unavailable during the execution of a WS-BPEL process, the remaining users can still complete the execution of the process. We extend RBAC-WS-BPEL with a new type of constraints called *resiliency constraints* and the notion of *user failure resiliency* for WS-BPEL processes and propose an algorithm to determine if a WS-BPEL process is user failure resilient.

## 1 Introduction

Several XML-based languages have been proposed for specifying and orchestrating business processes, resulting in the WS-BPEL standard language. WS-BPEL has been developed to specify automated business processes that orchestrate activities of multiple Web services. There are, however, cases in which people must be considered as additional participants to the execution of a process. Therefore, it is important to extend WS-BPEL to include the specification of activities that must be fully or partially performed by humans. The inclusion of humans, in turn, requires an access control model to support the specification and enforcement of authorizations to users for the execution of human activities while enforcing constraints, such as separation of duty, on the execution of those activities. One such model is RBAC-WS-BPEL, a role based access control model for WS-BPEL, that supports the specification of authorization information stating which role or user is allowed to execute which human activities in a process [6]. The authorization information comprises a role hierarchy reflecting the organizational structure, a permission-role assignment relation, and a set of permissions which represent the ability to execute activities. Authorization constraints place restrictions on the roles and users that can perform the activities in the business process. RBAC-WS-BPEL includes also a mechanism to

determine if user requests to perform an activity in a WS-BPEL process can be granted or not; requests are granted if the execution of a WS-BPEL process will complete without violation to the authorization constraints.

In many situations, it is however necessary to make sure that a process can complete even if certain users become unavailable to execute critical activities in the process. The set of available users may change during the execution of a WS-BPEL process for a large variety of reasons. Therefore, resiliency to user unavailability is an important requirement for WS-BPEL processes.

In this paper, we investigate the problem of *resiliency* for WS-BPEL processes. The goal of resiliency is to guarantee that even if some users become unavailable, the remaining users can still complete the activities according to the stated authorization constraints. To address such goal, we extend RBAC-WS-BPEL with a new type of constraints, referred to as *resiliency constraints*, that specify the minimum number of users that must be associated with the execution of the activities in order to give some assurance that even if some users are not available, the WS-BPEL process can terminate. We also define an algorithm that generates assignments (if such assignments exist) of users to activities that satisfy both the authorization constraints and the resiliency constraints.

The remainder of the paper is organized as follows. Section 2 introduces a running example. Section 3 presents the main components of RBAC-WS-BPEL authorization model. Section 4 investigates the problem of resiliency for WS-BPEL processes. Section 4 describes the approach to check if a WS-BPEL process is user failure resilient and presents some complexity results. Sections 6 and 7 discuss the system architecture and report experimental results respectively. Section 8 outlines related work. Section 9 concludes the paper and outlines future research directions.

## 2 Running Example

In this section we show an example of WS-BPEL process that implements the submission of a research project in an academic institution (see Figure [1](#)). The process orchestrates the following operations:

- the **submit** operation, by the **Submission** service, to submit a project proposal and check if the proposal satisfies various regulations;
- the **review** operation, by the **Review** service, that allows one to review the project proposal;
- the **approve** operation, by the **Approval** service, that allows a faculty member to check the reviews and decide if the project must be supported or not;
- the **assign\_funds** operation, by the **Fund Assignment** service, that allows one to revise the funds available and to determine the amount of funds that can be assigned to the project.

The project submission process is organized as follows. First, a faculty member or a phd student submits a project proposal to the academic institution by invoking operation **submit** (the `<receive>` **submit** activity). Then, the two **review**

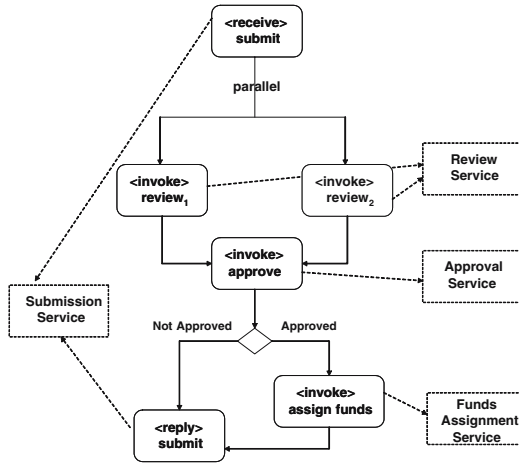


Fig. 1. Project Submission process specification

operations (`<invoke> review` activity) are executed in parallel. After the review process is completed, the `approve` operation is executed (`<invoke> approve` activity): if the project is approved, the operation `assign_funds` (`<invoke> assign_funds` activity) is performed and a notification is sent back to the project investigator (`<reply> submit` activity).

### 3 RBAC-WS-BPEL Authorization Model

RBAC-WS-BPEL applies to WS-BPEL business processes deployed in a single organization composed of different organizational units. RBAC-WS-BPEL inherits all the components of traditional RBAC models: users, roles, permissions, role hierarchies, user-role assignment and role-permission assignment relations. Moreover, RBAC-WS-BPEL supports the specification of authorization constraints such as separation of duty and binding of duty that restrict the set of users that can perform a given activity (see Figure 2). In particular, RBAC-WS-BPEL associates with a WS-BPEL business process a set of roles  $R$ . Each role is associated with a set of conditions on users' properties that users must satisfy in order to be assigned to that role. Examples of such properties are "social security number", "birth-date" and "employment". Users' properties, that are referred to as *identity attributes*, are conveyed in digital credential issued by trusted third parties called Certification Authorities. Therefore, the assignment of a user to a role is executed by evaluating the user's attributes against the conditions associated with the role. If the user attributes satisfy such conditions, the user is assigned to the role.

In RBAC-WS-BPEL, permissions represent the ability to execute an activity of a WS-BPEL business process and are specified as tuples of the form  $(A_i, Action)$  where  $A_i$  identifies an activity and  $Action$  identifies the execution

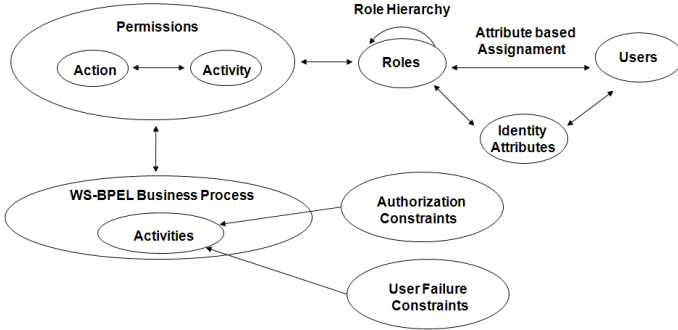


Fig. 2. RBAC-WS-BPEL main components

of the activity. Permissions are assigned to roles that are structured in a *role hierarchy* that defines a permission inheritance relation among the roles. RBAC-WS-BPEL supports separation of duty and binding of duty constraints and any authorization constraint that can be expressed as a binary relation on the set of users or roles. An authorization constraint is represented by a tuple  $\langle D, (A_1, A_2), \rho \rangle$ , where  $D$  is the user or role who has executed activity  $A_1$ , called the *antecedent activity*,  $A_2$  is the *consequent activity* to which the constraints is applied and  $\rho$  is a relation on  $U$ , the set of users, or on  $R$ , the set of roles. A constraint  $\langle D, (A_1, A_2), \rho \rangle$  is *satisfied* if, whenever  $x \in D$  performs  $A_1$  and  $y$  performs  $A_2$ , then  $(x, y) \in \rho$ . Authorization information is encoded in RBAC-XACML [3] while authorization constraints are represented in BPCL, a special purpose XML language for specifying authorization constraints [6]. Finally, RBAC-WS-BPEL supports an algorithm to evaluate at runtime whether a request to execute an activity by a user can be granted or not on the basis of the authorizations the user has and on the basis of authorization constraints defined for the activity.

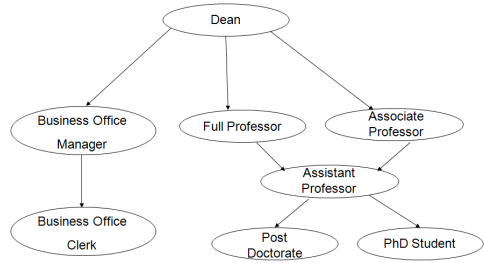
*Example 1.* Figures 3 and 4 show the RBAC-WS-BPEL components defined for our running example. Figure 3 (b) shows the role hierarchy that specifies the different positions in an academic institution. Figure 3 (a) lists for each role the users explicitly assigned to each role. Figure 4 (a) illustrates a typical role-permission assignment relation. For example, activity `<invoke> review1` can be performed both by an `Assistant professor` and by an `Associate professor`. Figure 4 (b) reports the authorization constraints defined for the project submission process.  $C_1$  is a binding of duty constraint, requiring that the same user that assigns the funds to the project must notify if the project proposal has been approved or not.  $C_2, C_3, C_4, C_5$  and  $C_6$  are separation of duty constraints. For example,  $C_2$  states that the users who perform `<invoke> review1` and `<invoke> review2` must be different.

<sup>1</sup> A user is *explicitly* assigned to a role if the user's attributes satisfy the conditions associated with the role. A user is *implicitly* assigned to the roles that are dominated in the role hierarchy by the role to which the user is explicitly assigned.



Roles	Users
Dean	{John }
Full professor	{Mary, Jane}
Associate professor	{Chris, Irini }
Assistant professor	{Anna, Dan }
Post Doctorate	{Ellen, Doug}
PhD Student	{Ashish, Melanie, Kara}
Business Office Manager	{Tammy }
Business Office Clerk	{ Robynne, Leslie}

(a) Roles



(b) The role hierarchy

**Fig. 3.** RBAC-WS-BPEL role hierarchy and role-permission assignment relation for the project submission process

Roles	Permission
Post Doctorate, PhD Student	(<receive> submit, execute)
Assistant professor, Associate professor	(<invoke> review <sub>1</sub> , execute)
Assistant professor, Associate professor	(<invoke> review <sub>2</sub> , execute)
Full professor	(<invoke> approve, execute)
Business Office Manager	(<invoke> assign_funds, execute)
Business Office Clerk	(<reply> submit, execute)

(a) Role-permission assignment relation

Authorization Constraint
C <sub>1</sub> <U, (<invoke> assign_funds, <reply> submit), = >
C <sub>2</sub> <U, (<invoke> review <sub>1</sub> , <invoke> review <sub>2</sub> ), ≠ >
C <sub>3</sub> <U, (<receive> submit, <invoke> review <sub>1</sub> ), ≠ >
C <sub>4</sub> <U, (<receive> submit, <invoke> review <sub>2</sub> ), ≠ >
C <sub>5</sub> <U, (<receive> submit, <invoke> approve), ≠ >
C <sub>6</sub> <U, (<invoke> review <sub>1</sub> , <invoke> approve), ≠ >
C <sub>7</sub> <U, (<invoke> review <sub>2</sub> , <invoke> approve), ≠ >

(b) Authorization constraints

**Fig. 4.** RBAC-WS-BPEL role-permission assignment relation and authorization constraints for the project submission process

## 4 Process User Failure Resiliency

In this section we introduce the key notions of our resiliency model.

**Definition 1 (Resiliency constraint).** *Let  $U$  be the set of users and let  $BP$  a WS-BPEL process. A resiliency constraint is a tuple  $\langle A_i, n_i \rangle$ , where  $A_i \in BP$  and,  $n_i \in \mathbb{N}$ ,  $n_i$  denotes the minimum number of users that must have the authorization<sup>2</sup> to perform  $A_i$ .*

We now introduce the notion of *user failure resiliency* for a WS-BPEL business process.

**Definition 2 (User Failure Resiliency).** *Let  $U$  be the set of users,  $BP$  be a WS-BPEL business process,  $U_{A_i}$  be the set of users authorized to perform activity  $A_i \in BP$ , and  $\langle A_i, n_i \rangle$  be a resiliency constraint for activity  $A_i$ . We say that  $BP$  is user failure resilient if for each  $A_i \in BP$  such that a resiliency constraint  $\langle A_i, n_i \rangle$  exists, then  $|U_{A_i}| \geq n_i$ . Moreover, the maximum resiliency of a WS-BPEL business  $BP$ , denoted as  $MaxRes$ , is defined as the maximum over the set  $\{n_i \mid \exists \langle A_i, n_i \rangle \text{ such that } A_i \in BP\}$ .*

If a WS-BPEL process is user failure resilient, there is a sufficient number of authorized users to perform the process so that authorization constraints are satisfied and the process terminates even if some users become unavailable.

A relevant concept to determine whether a WS-BPEL process is user failure resistant is the concept of *configurations*.

**Definition 3 (Configuration).** *Let  $U$  be the set of users,  $BP$  be a WS-BPEL business process and  $U_{A_i}$  be the set of users authorized to perform activity  $A_i \in BP$ . Let  $C$  be the set  $\{\langle A, u \rangle \mid A \in BP \wedge u \in U_A\}$ . We say that  $c, c \subseteq C$ , is a configuration for  $BP$  if  $\forall A_i \in BP, \exists$  one and only one tuple  $\langle A, u \rangle \in c$  such that  $A = A_i$ .*

The above definition states basically that a configuration must specify a user assignment for each activity in the process.

To determine if a WS-BPEL process is user failure resilient a possible approach is to compute the set  $S$  of all possible configurations and then evaluate if the resiliency constraints are satisfied. All such configurations would be then stored to decide which user has to substitute another user if the latter becomes unavailable at run-time. However, rather than computing and storing all the possible configurations, it is sufficient to compute only a subset  $S_c$  of  $S$  that satisfies the following property:

---

<sup>2</sup> To determine if a WS-BPEL business process is user failure resilient, we assume that a user has the authorization to execute an activity  $A_i$  if he/she is assigned to a role which has the permission to perform  $A_i$ . The authorization to execute the activity  $A_i$  is effectively granted to the user only at runtime when he/she claims the execution of  $A_i$ .

for each activity  $A_i \in BP$  such that a resiliency constraint  $\langle A_i, n_i \rangle$  exists,  $|\bigcup_{c \in S_c} \{ \langle A_i, u \rangle \mid \langle A_i, u \rangle \in c \}| = n_i$ .

A resiliency constraint  $\langle A_i, n_i \rangle$  for an activity  $A_i$  is satisfied if it is possible to find at least  $n_i$  users authorized to perform  $A_i$  and therefore  $n_i$  configurations. It is trivial to prove that if  $S_c$  exists, the cardinality of  $S_c$  is equal to  $MaxRes$ .

In the next section we evaluate the complexity of computing the configurations in  $S_c$ .

#### 4.1 Computational Complexity of Checking User Failure Resiliency

The complexity of checking whether a WS-BPEL process is user failure resilient is given by the following lemmas.

**Lemma 1.** *Checking whether a WS-BPEL process is user failure resistant, which is called the user failure resiliency checking problem (RCP for short), is NP-Complete in RBAC-WS-BPEL.*

**Lemma 2.** *RCP is P in RBAC-WS-BPEL if only binding of duty constraints are specified on the process activities.*

**Lemma 3.** *RCP is NP-Complete in RBAC-WS-BPEL if only separation of duty constraints are specified on the process activities.*

See [7] for the proofs.

## 5 Constraints Evaluation and Planning

As we proved in the previous section, the complexity of computing the configurations to check whether a WS-BPEL process is user failure resilient is NP-Complete. In fact, in the worst case, the complexity is  $O(|U|^{|A|} * MaxRes)$ , where  $|U|$  is the number of potential users and  $|A|$  is the number of activities in the process, because to compute a configuration, all the possible assignments of users to activities are tried for all the activities in the process and this step is iterated a number of times equal to  $MaxRes$ .

To reduce the complexity of computing configurations, we thus introduce two heuristics that reduce the number of assignments of users to activities. First, for all the activities that are linked by a binding of duty constraint, the set of users that are authorized to perform these activities is set to the intersection of the sets of users who are authorized to perform each single activity. For example, if the binding of duty constraints  $\langle U, (A_1, A_2), = \rangle$  and  $\langle U, (A_1, A_3), = \rangle$  are specified for activities  $A_1, A_2$  and  $A_3$ , the sets of users  $U_{A_1}, U_{A_2}$  and  $U_{A_3}$  that are authorized to perform  $A_1, A_2$  and  $A_3$  are equal to the intersection  $V_{A_1} \cap V_{A_2} \cap V_{A_3}$ .  $V_{A_1}, V_{A_2}$  and  $V_{A_3}$  are, respectively, the set of users that are authorized to execute  $A_1, A_2$  and  $A_3$  because they are assigned to a role that has the permission to execute  $A_1, A_2$  and  $A_3$ . The adoption of this heuristic increases the success rate of assignment of users to activities and therefore minimizes the number of user assignments.

**Algorithm 1.** Process User Failure resiliency satisfaction

**Require:**  $AC$  set of authorization constraints,

$RC$  set of resiliency constraints,

$Activities$  set of activities ordered according to the business process specification

**Ensure:** BP is user failure resistant

```

1.  $MaxRes = \text{Max}(RC)$ 
2. for  $A_i \in Activities$  do
3.    $V_{A_i} = \text{getAuthorizedUsers}(A_i)$ 
4.   if  $|V_{A_i}| < RC_{A_i}$  then
5.     exit
6.   else
7.      $V. \text{add}(V_{A_i})$ 
8.   end if
9. end for
10.  $LinkedActivities = \text{getSubActivities}(Activities)$ 
11.  $satisfiable = \text{true}$ 
12. while ( $Num < MaxRes$  AND  $satisfiable$ ) do
13.   for  $LinkedActivities_i \in LinkedActivities$  do
14.      $Current\_Activity = LinkedActivities_i.\text{head}()$ 
15.      $SubConfig_i = \text{build\_config}(Current\_Activity, V, AC, RC, Conf\_Set,$ 
16.        $SubConfig_i, LinkedActivities_i)$ 
17.      $SubConfig.\text{add}(SubConfig_i)$ 
18.   end for
19.    $Current\_Config = \text{merge}(SubConfig)$ 
20.   if  $Current\_Config.\text{size}() == Activities.\text{size}()$  then
21.      $satisfiable = \text{true}$ 
22.      $Conf\_Set.\text{add}(Current\_Config)$ 
23.      $Num = Num + 1$ 
24.   else
25.      $satisfiable = \text{false}$ 
26.   end if
27. end while
28. if  $|Conf\_Set| < MaxRes$  then
29.   for  $A_i \in Activities$  do
30.     if  $|Conf\_Set| < RC_{A_i}$  then
31.        $Missing\_Users = RC_{A_i} - |Conf\_Set|$ 
32.        $Roles = \text{ua-update}(Missing\_Users)$ 
33.     end if
34.   end for

```

The second heuristic groups the activities that are linked by authorization constraints. For example, activities  $A_4$  and  $A_5$  are in the same subset of activities if there is a separation of duty constraint  $\langle U, (A_4, A_5), \neq \rangle$  between  $A_4$  and  $A_5$ . For each subset of activities, a partial configuration is computed and then a complete configuration for the process is generated by merging the partial configurations. This optimization reduces the number of user assignments to activities because, when the assignment of a user to an activity fails, the

reassignment of a user is performed only for the antecedent activities that are in the same subset of the activity for which the assignment fails and not for all the other antecedent activities. The computation of the set of users authorized to perform the activities linked by a binding of duty constraint and of the subsets of activities has complexity  $|AC|^2$ , where  $|AC|$  is the number of authorization constraints. The computation of the sub-configurations for each subset of activities has complexity  $O(|U_{subset}|^{|A_{subset}|})$ , where  $|U_{subset}|$  and  $|A_{subset}|$  are respectively the number of candidate users and the number of activities in each subset while the complexity of combining the sub-configurations together to obtain a configuration for the whole business process is  $|A|$ . Therefore, by adopting these heuristics, the complexity of calculating the configurations necessary to assure that a WS-BPEL process is user failure resilient becomes  $O(|AC|^2 + MaxRes * \{|U_{subset}|^{|A_{subset}|} + |A|\})$ .

Algorithm 1 adopts the heuristics we have described to compute a number of users-to-activities assignment configurations equal to  $MaxRes$ . First of all the algorithm, computes  $MaxRes$  (line 1). Then, for each activity  $A_i$ , the procedure `getAuthorizedUsers` returns the set of users  $V_{A_i}$  that are authorized to perform activity  $A_i$  because they are assigned to a role that has the permission to execute  $A_i$  (lines 2-3). If the cardinality of  $V_{A_i}$  is lower than the resiliency value specified for  $A_i$ , the algorithm terminates, otherwise  $V_{A_i}$  is added to  $V$ , that is a vector containing for each activity  $A_i$  the set  $V_{A_i}$  (line 7). Then, the procedure `getSubSetActivities` calculates the subsets of activities  $SubSetActivities$  on the basis of the authorization constraints that are applied to them. Each  $SubSetActivities$  is saved in the  $LinkedActivities$  set (line 10). Then, the algorithm iterates till a number of user configurations equal to  $MaxRes$  is not found (line 12) or it is not possible to find such a number of configurations because all the possible combinations of users-to-activities assignment have been tried. The configurations are computed by the recursive procedure `build_config`. `build_config` is executed for each subset  $LinkedActivities_i$  and returns a partial configuration  $SubConfig_i$ . Once a partial configuration  $SubConfig_i$  is computed for each subset  $LinkedActivities_i$ , the procedure `merge` combines all  $SubConfig_i$  in one configuration  $Current_Config$  that is added to the configurations set  $Config_Set$ . If Algorithm 1 is not able to compute a number of configurations equal to  $MaxRes$ , it determines the activities for which the resiliency constraint is not satisfied. These activities are the activities whose resiliency value is lower than the number of configurations in  $Config_Set$ . For each of these activities, Algorithm 1 calculates the number of additional potential users should be associated with the execution of the activities. Then, `ua-update` checks the logs associated with the activities and returns the roles for which the user failure assignment has more frequently failed. The additional potential users needed must be added to these roles.

*Example 2.* Assume that for the activities `<invoke> review1`, `<invoke> review2` and `<invoke> approve` the following resiliency constraints are specified: (`<invoke> review1`, 3), (`<invoke> review2`, 3) and (`<invoke> approve`, 2). Since  $MaxRes$  is equal to three, to prove that the project submission process is user failure resilient,

we need to find three different users-to-activities assignment configurations. An example of such configurations is:

1. (<receive> submit, Irini), (<invoke> review<sub>1</sub>, Mary), (<invoke> review<sub>2</sub>, Anna), (<invoke> approve, Jane), (<invoke> assign\_funds, John), (<reply> submit, John)
2. (<receive> submit, Kara), (<invoke> review<sub>1</sub>, Chris), (<invoke> review<sub>2</sub>, Mary), (<invoke> approve, John), (<invoke> assign\_funds, Tammy), (<reply> submit, Tammy)
3. (<receive> submit, Irini), (<invoke> review<sub>1</sub>, Jane), (<invoke> review<sub>2</sub>, John), (<invoke> approve, Mary), (<invoke> assign\_funds, Tammy), (<reply> submit, Tammy).

Therefore, the project submission process is user failure resilient. Consider a different scenario, in which the resiliency constraints (<invoke> review<sub>1</sub>, 4), (<invoke> review<sub>2</sub>, 4) and (<invoke> approve, 3) are applied to activities <invoke> review<sub>1</sub>, <invoke> review<sub>2</sub> and <invoke> approve. Now, we have to find four configurations to prove that the process is resilient since *MaxRes* is equal to four. In this case, the following configurations are generated:

1. (<receive> submit, Jane), (<invoke> review<sub>1</sub>, Dan), (<invoke> review<sub>2</sub>, Chris), (<invoke> approve, Mary), (<invoke> assign\_funds, Tammy), (<reply> submit, Tammy)
2. (<receive> submit, Anna), (<invoke> review<sub>1</sub>, Irini), (<invoke> review<sub>2</sub>, John), (<invoke> approve, Jane), (<invoke> assign\_funds, John), (<reply> submit, John)
3. (<receive> submit, Kara), (<invoke> review<sub>1</sub>, Jane), (<invoke> review<sub>2</sub>, Mary), (<invoke> approve, John), (<invoke> assign\_funds, null), (<reply> submit, null)
4. <receive> submit, Ashish), (<invoke> review<sub>1</sub>, Chris), (<invoke> review<sub>2</sub>, Anna), (<invoke> approve, null), (<invoke> assign\_funds, null), (<reply> submit, null).

It's easy to see that the process is not user failure resilient because the first two configurations are complete but for the other ones the assignment of a user to activities <invoke> approve, <invoke> assign\_funds and <reply> submit fails.

## 6 System Architecture

The main components of the RBAC-WS-BPEL architecture (see Figure 5) are the *WS-BPEL engine*, the *XACML Policy Store*, *BPCL Constraints Store* repositories, the *History Store* and the *RBAC-WS-BPEL Enforcement Service*. The WS-BPEL engine is responsible for scheduling and synchronizing the various activities within the business process according to the specified activity dependencies, and for invoking Web services operations associated with activities. The XACML Policy Store records the RBAC-WS-BPEL authorization schema associated with the business process, whereas the BPCL Constraints Store records

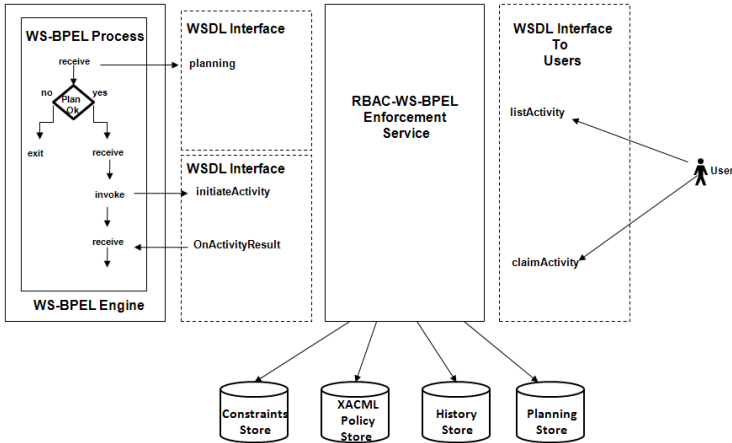


Fig. 5. RBAC-WS-BPEL architecture

the authorization constraints. The History Store records the users who have performed an activity and whether the execution of the activity has been successful or not. The RBAC-WS-BPEL Enforcement Service supports the WS-BPEL process administrators both at deployment time and at runtime. When the process is deployed, the RBAC-WS-BPEL Enforcement Service checks if the process is user failure resilient and, hence, if there is a number of users sufficient to start the execution of the process, while during the execution of the process the RBAC-WS-BPEL Enforcement Service checks whether the execution of an activity by a user violates authorization constraints. The RBAC-WS-BPEL Enforcement Service offers three WSDL interfaces. The first interface provides the operations for starting and completing the execution of a WS-BPEL activity that must be performed by a user. The second interface allows users to visualize the activities they can claim, and to claim and execute them [6]. The third interface provides functions for determining if a WS-BPEL process is user failure resilient.

In what follows, we focus on the description of the third interface, because it is the most relevant for the discussion in the paper.

Such interface provides a single operation, called **planning**, that implements Algorithm 1. The **planning** operation notifies the WS-BPEL process administrator if the process is user failure resilient and, if this is not the case, displays the activities for which the resiliency constraints are not satisfied and the roles authorized to perform the activities that should be populated with additional users. The WS-BPEL process administrator can decide to proceed with the execution of the process or to halt the execution and modify the set of potential users associated with the execution of the process. To enable the execution of the **planning** operation, the WS-BPEL process designer has to include in the `<partnerLinks>` list, the RBAC-WS-BPEL Enforcement Service. Moreover, the WS-BPEL specification must be such that the start activity of the process is a `<receive>` activity that executes the **planning** operation. The `<receive>` **planning** activity is followed by an `<if>` activity that performs the subsequent

**Table 1.** Test cases parameters

Test Case	Business Process	Num of BoD	Num of SoD	MaxRes	Num of Users
1	21 activities	4	4	6	50..140
2	21 activities	4	4	[3,9]	50
3	21 activities	[0,5]	4	6	50
4	21 activities	0	[3,6]	6	50

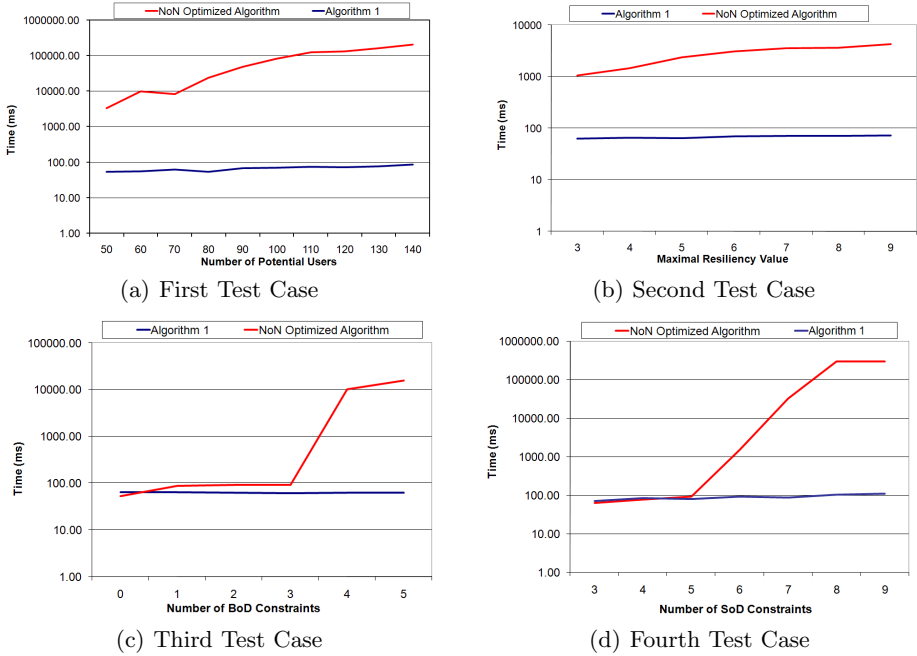
activities that implement the business process only if the execution of **planning** is successful. The **planning** operation retrieves from the XACML Policy Store the hierarchy of roles and the list of users assigned to the roles and selects from the BPCL Constraints Store the set of authorization and resiliency constraints that are necessary for executing Algorithm 1. All configurations computed by the **planning** operation are stored in an additional repository, referred to as *Planning Store*, while the logs of unsuccessful assignments of users to activities are recorded in the History Store.

## 7 Experimental Evaluation

We have carried out several experiments to evaluate the impact of the heuristics for reducing the cost of the configuration computation and to prove the effectiveness of our approach. To execute the tests we have implemented Algorithm 1 and, in addition, an algorithm, referred to as *NoNOptimized*, which computes a number of configurations equal to *MaxRes* as Algorithms 1 but without adopting the heuristics to reduce the complexity. We have also generated a WS-BPEL process composed by 21 activities, a set of 50 potential users, a role hierarchy of 7 roles, 4 separation of duty and 4 binding of duty constraints. Such process has a *MaxRes* value equal to 6. We have considered four test cases that are summarized in Table 1. In particular, we have measured in CPU time (milliseconds) the execution time of Algorithm 1 and of the *NoNOptimized* algorithm in the following cases:

1. we varied the number of potential user from 50 to 140 and we kept the number of separation of duty and binding of duty constraints equal to 4, and the value of *MaxRes* equal to 6.
2. we varied the value of *MaxRes* from 3 to 6 and we kept the number of separation of duty and binding of duty constraints equal to 4 and the number of potential users equal to 50.
3. we varied the number of binding of duty constraints defined for the process from 0 to 5 and we set the number of separation of duty constraints to 4, the value of *MaxRes* to 6 and the number of potential users to 50.
4. we varied the number of separation of duty constraints defined for the process from 3 to 6 and we set the number of binding of duty constraints to 4, *MaxRes* value to 6 and the number of of potential users to 50.





**Fig. 6.** Experimental results

The experiments have been run on a PC with operating system WINDOWS XP SP2, a 2Gz T7200 processor and 2GB RAM. Moreover, for each test case we have executed twenty trials, and the average over all the trial execution times has been computed.

Figures 6 (a) and (b) report the execution times measured for test cases 1 and 2. The execution times of Algorithm 1 are almost constant for increasing values of the number of potential users and  $MaxRes$  value, while the execution time of the *NoNOptimized* algorithm increases. The reason is that in Algorithm 1 the first heuristic reduces the number of unsuccessful users assignments. Moreover, the second heuristic reduces the number of activities for which we try to reassign a user in case of failure. Instead, for the *NoNOptimized* algorithm, the time increases because in case of user assignment failure for an activity  $A_i$ , a re-assignment of a user is tried for all the antecedent activities  $A_{i-1}, A_{i-2} \dots$  rather than only for the antecedent activities that are linked to  $A_i$  by an authorization constraint. The experimental results reported in Figure 6 (c) show the advantage of adopting the first heuristic about the activities that are linked by a binding of duty constraint. When the number of binding of duty constraints is equal to 0, the execution times of Algorithm 1 and of the *NoNOptimized* algorithm are the same; while when the number of binding of duty constraints increases, the execution times of the *NoNOptimized* algorithm become greater than Algorithm 1. When increasing the number of binding of duty constraints, the user assignment

success rate for Algorithm 1 increases and, as a consequence, the number of user assignments is minimized. This is the reason why Algorithm 1's execution time is almost constant, while the execution time of the *NoNOptimized* algorithm increases. Finally, Figure 6 (d) shows the impact of generating the subsets of activities. The increase of the number of separation of duty constraints restricts the number of users authorized to perform the activities and, as a consequence, the probability that a user assignment fails is very high. Therefore also the number of reassignments of users to activities is high. The execution time of Algorithm 1 is lower than the time of the *NoNOptimized* algorithm because the number of activities for which the user reassignments is performed is minimized; the user reassignment is tried only for the antecedent activities in the same subset of the activity for which the user assignment fails. Note that the execution time of Algorithm 1, regardless of the test cases we have performed, is under 100 ms. Such results show that our approach to check whether a WS-BPEL process is user failure resilient is applicable to real case scenarios.

## 8 Related Work

With the widespread adoption of Web services composition to implement complex business processes and of WS-BPEL as the standard language to specify business processes based on Web services, the problem of how to associate authorized users with the activities of a WS-BPEL process is gaining attention. Koshutanski et al. [5] propose an authorization model for business processes based on Web services. Both the model of Koshutanski et al. and RBAC-WS-BPEL assume an RBAC model and support authorizations constraints on the set of users and roles. They also consider the problem of taking authorization decision on the execution of business process's activities. The main difference with RBAC-WS-BPEL is in the approach to take authorization decision. In the model by Koshutanski et al., an authorization decision is taken by orchestrating the authorization processes of each Web service, the activities of which are orchestrated in the business process, while in RBAC-WS-BPEL an authorization decision is taken independently for each activity in the process.

Xiangpeng et al. [8] propose an RBAC access control model for WS-BPEL business process. Roles correspond to `<partnerRole>` elements in the WS-BPEL specification and are organized in a hierarchy. Permissions correspond to the execution of the basic activities in the process specification. In addition, separation of duty constraints can be specified. A main difference with respect to our approach is that RBAC-WS-BPEL's BCPL constraints language supports the specification of a broader range of authorizations constraints than the model by Xiangpeng et al.

BPEL4People [1] is a recent proposal to handle person-to-person WS-BPEL business process. With respect to RBAC-WS-BPEL, in BPEL4People users that have to perform the activities of a WS-BPEL business process are directly specified in the process by user identifier(s) or by groups of people's names. No assumption is made on how the assignment is done or on how it is possible to enforce constraints like separation of duties.

The workflow authorization model proposed by Wang et al. [9] is probably the one that is most closely related to RBAC-WS-BPEL. Wang et al. propose the role-and-relation-based access control ( $R^2BAC$ ) model for workflow systems. In  $R^2BAC$ , in addition to a users role memberships, the users relationships with other users help determine whether the user is allowed to perform a certain step in a workflow. Wang et al. investigate the workflow satisfiability problem, which asks whether a set of users can complete a workflow. They also investigate the resiliency problem in workflow systems, which asks whether a workflow can be completed even if a number of users may be absent. The notion of resiliency supported by RBAC-WB-BPEL is slightly different from the one proposed by Wang et al. They propose a notion of resiliency parametrized in the number of absent users. A workflow is resilient, if it is satisfiable in any configuration where any set of users of cardinality equal to the parameter is not available. Instead, in RBAC-WS-BPEL, a WS-BPEL process is resilient if it is possible to find a number of configurations that satisfy both resiliency constraints and authorization constraints.

## 9 Conclusions

In this paper, we have investigated the resiliency problem for WS-BPEL business processes. Resiliency in the context of business process means that even if some users become unavailable, the remaining users can still complete the execution of the process according to the stated authorizations and authorization constraints. To address such problem, we have extended RBAC-WS-BPEL, which is an authorization model for WS-BPEL business processes, with the notions of *resiliency constraints* for activities and *user failure resiliency* for a business process. We have proposed an algorithm that allows to statically determine if a WS-BPEL is user failure resilient. The algorithm verifies there is a number of users-to-activities assignment configurations equal to  $MaxRes$ . These configurations are computed by assuming that users are assigned to the execution of an activity because they cover a role that is granted the execution of the activity. The authorization to execute an activity is effectively granted to users only at runtime when they claim the execution of the activity. Though, the complexity of computing such configurations is NP-complete in the most general case, the proposed algorithm adopts some heuristics that reduce the computational complexity. The experimental results, we have performed, have shown that the algorithm is efficient in most practical cases.

We are planning to extend this work in several directions. We are currently investigating how the RBAC-WS-BPEL Enforcement Service can be implemented on top of ODE BPEL engine [4]. We also want to extend RBAC-WS-BPEL to support authorizations for cross-organizations business processes. Currently, RBAC-WS-BPEL is applied to inter-organization business processes. As we have been told by the main companies providing solutions for WS-BPEL processes, WS-BPEL is mainly used to specify inter-organization business processes rather than cross-organizations business processes. We are also planning to extend RBAC-WS-BPEL with more sophisticated authorization constraints.

## References

1. Agrawal, A., et al.: WS-BPEL Extension for People (BPEL4People), Version 1.0 (2007), <http://www.adobe.com/devnet/livecycle/pdfs/bpel4people.spec.pdf>
2. Alves, A. et al.: Web Services Business Process Execution Language, Version 2.0, OASIS Standard (April 2007), <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>
3. Anderson, A.: Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML, Version 2.0, OASIS Standard (2005), [http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-rbac-profile1-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf)
4. Apache ODE BPEL engine, <http://ode.apache.org/bpel-extensions.html>
5. Kostutanski, H., Massacci, F.: An Access Control Framework for Business Processes for Web Services. In: Proceedings of ACM Workshop on XML Security, George W. Johnson Center at George Mason University, Fairfax, Va, USA, October 2003, pp. 15–24 (2003)
6. Paci, F., Bertino, E., Crampton, J.: An Access Control Framework for WS-BPEL. *International Journal of Web service Research* 5(3), 20–43 (2008)
7. Paci, F., Ferrini, R., Sun, Y., Bertino, E.: Authorization and User Failure Resiliency for WS-BPEL business processes, Cerias Technical report (2008)
8. Xiangpeng, Z., Cerone, A., Krishnan, P.: Verifying BPEL Workflows Under Authorisation Constraints. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) *BPM 2006*. LNCS, vol. 4102. Springer, Heidelberg (2006)
9. Wang, Q., Li, N.: Satisfiability and Resiliency in Workflow Systems. In: Biskup, J., López, J. (eds.) *ESORICS 2007*. LNCS, vol. 4734, pp. 90–105. Springer, Heidelberg (2007)

# Reasoning on Semantically Annotated Processes

Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher,  
Luciano Serafini, and Paolo Tonella

FBK-irst, Via Sommarive 18 Povo, I-38100, Trento, Italy  
{dfmchiara,ghidini,rospocher,serafini,tonella}@fbk.eu

**Abstract.** Enriching business process models with semantic tags taken from an ontology has become a crucial necessity in service provisioning, integration and composition. In this paper we propose to represent semantically labelled business processes as part of a knowledge base that formalises: business process structure, business domains, and a set of criteria describing correct semantic labelling. Our approach allows (1) to impose domain dependent constraints during the phase of process design, and (2) to automatically verify, via logical reasoning, if business processes fulfill a set of given constraints, and to formulate queries that involve both knowledge about the domain and the process structure. Feasibility and usefulness of our approach will be shown by means of two use cases. The first one on domain specific constraints, and the second one on mining and evolution of crosscutting concerns.

## 1 Introduction

Semantic Business Process Management (SBPM) [16,12] has the main objective of improving the level of automation in the specification, implementation, execution, and monitoring of business processes by extending business process management tools with the most significant results from the area of semantic web. Focussing on process modeling, i.e. the activity of specification of business processes at an abstract level (descriptive and non executable), it has been argued that annotating process descriptions with a set of tags taken from a set of domain ontologies would provide an additional support to the business analysis in this phase. (see e.g. [20]).

However, semantic annotations will positively affect the creation of high quality process models only if they are correct. Though the notion of *correct semantic annotation* deserves a precise definition, we can intuitively say that a necessary condition for a correct annotation is that it respects types. E.g. activities in a business process should be labeled with some concepts denoting indeed an activity; similarly, conditional tests should be labeled with boolean conditions, and so on. Thus, for instance, an activity labeled “purchase order” or a gateway condition labeled “send a request” are intuitively not correct annotations. Additional requirements for a correct labeling could be imposed because of the specific application domain. For instance, in a domain in which simple actions only come from a fixed set, tasks should be only tagged with actions taken from

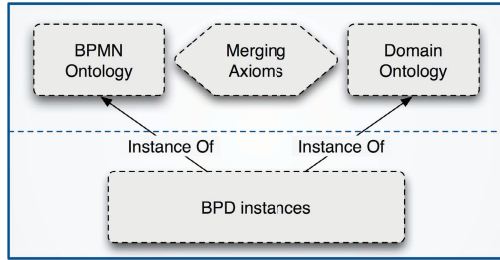
this set. Providing a way to specify the criteria for correct annotation is a critical and important issue in the development of business processes. It is even more important because, as shown in the examples above, while certain criteria for correct annotation can be valid for all (or a wide range of) business domains, others need to be specified for the specific business domain at hand.

Assuming that the process is correctly annotated, semantic tags can be helpful in several BPM activities. Semantic tags convey the necessary information to perform an early analysis of the process in order to find critical patterns, and can be used to guide the user to recognize problems at design time and features which can be useful in further refinements of the process specification. In addition the search for instances of these critical patterns can be automatised using queries. Another use is related to the presence and evolution of the so-called *crosscutting concerns* [23] of a business process. Crosscutting concerns are process features that cannot be modularized into a single unit (e.g., an activity or a subprocess), because they are intrinsically scattered across the process and tangled with the other concerns. For example, in a business process for an online shop there are usually several places in which the user makes choices based on her/his preferences. All related activities form a crosscutting concern, the *user preferences* concern. Knowledge about its presence is important to understand how such a feature is currently managed and how it can be evolved in the future (e.g., storing preferences and making suggestions to the user in a proactive way). Semantic annotations provide the basic information necessary for documentation and consistent management of crosscutting concerns in business processes.

In this paper we propose an approach for (1) the specification of constraints for correct annotations of business processes, (2) the automatic verification of the correctness of annotated processes, and (3) the provision of reasoning services on labelled processes via query answering. Our approach is based on two pillars: first, on the implementation of a Knowledge Base, called *Business Process Knowledge Base*, that contains: an ontology formalizing one of the most widely used language for describing business processes, i.e. BPMN [7], a (set of) domain ontology(es) that provide the tagging language and semantics, and a set of merging axioms that specify labeling restrictions and annotation criteria; second, on a tool that automatically translates a BPMN process labelled with semantic tags into a set of assertions of the knowledge base. Detection of correctness of semantic tagging, suggestion of possible correct tagging, and semantic query based analysis of a process in order to find criticalities and manage crosscutting concerns are implemented via logical reasoning.

## 2 The Approach

Business Process Modeling Notation (BPMN) [21] is a (graphical) language for the specification of Business Process Diagrams (BPD). Roughly speaking, a BPD is an annotated graph whose nodes represents activities, control flows, data, and auxiliary information about the process. In our semantic variant of BPMN we allow for tagging objects of a BPD with concept descriptions taken from a (set of) domain ontology(es), i.e. shared formalizations of a specific domain.



**Fig. 1.** Business Processes Knowledge Base

However, not every BPD object can be labelled with any concept. Meaningful semantic annotations should respect some restrictions. For instance, if the domain ontology is RosettaNet<sup>1</sup>, then a BPMN object of type activity can be correctly tagged by `rosetta:PIP`<sup>2</sup> (Partner Interface Process, i.e. a protocol of outgoing and incoming messages representing request and response dialogue), or some of its subclasses, but it would be a mistake to label it with `rosetta:TotalPrice` (i.e. Total Price for the entire business document, including freight, tax and special handling if applicable).

Criteria for correct/incorrect annotation are statements that bridge the semantics of BPMN and the semantics of the domain ontology. From the formal point of view, these criteria can be represented by inclusion axioms between the concepts of an ontology formalizing BPD and the domain ontology.

In order to express this kind of constraints and to support automated reasoning on them, we propose to encode all the information about semantically annotated processes into a logical knowledge base, called *Business Processes Knowledge Base* (BPKB) and schematized in Figure 1. A BPKB is composed of the following four modules:

**BPMN ontology** formalises the structure of a BPD. This ontology is a formalization of the BPMN standard [21] and consists of a set of axioms that describe the BPMN elements and the way in which they can be combined for the construction of BPDs.

**Domain Ontology** is a (set of) ontology(es) that describes a specific business domain. It can be an already existing business domain ontology (e.g. RosettaNet or similar standard business ontologies) or an ontology developed on purpose.

**Merging axioms** state the correspondence between the domain ontology and the BPMN ontology. They formalise the criteria for correct/incorrect semantic annotations.

**BPD instances** contain the description of a set of BPDs in terms of instances of the BPMN/domain ontology. Every element of the process is represented as an

<sup>1</sup> [www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta.owl](http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta.owl)

<sup>2</sup> The notation `<ontology>:<concept-name>` stands for the concept `<concept name>` of the ontology `<ontology>`.

individual of a class. The structure of the process (i.e. the connection between different elements) is represented by means of relations between instances.

A BPKB can be implemented in any knowledge representation language with minimum expressive power and a complete decision procedure. Using logical reasoning over BPKB we can implement the following services:

**Query answering on the BPD instances:** a number of queries that involve either the domain ontology, the BPMN ontology or both, can be formulated. An example of query would be to “provide all the actions that are associated with credit card data”. We can see that to formulate this query we need knowledge coming from the BPMN ontology (the concept “action” and the relation “associated with”) and knowledge coming from the domain ontology (“credit card data”).

**Verification of semantic labeling:** verifying whether the semantic labelling satisfies the constraints specified using the merging axioms.

**Suggestions for correct labelling of the process:** merging axioms can be used to suggest (sets of) labels on-the-fly during process annotation. Note that the usage of merging axioms ensures that the suggested labels are correct.

### 3 The Business Process Knowledge Base

We implemented BPKB using the standard semantic web language OWL (Web Ontology Language) based on Description Logics (DL). Description Logics (see [4]) are a family of knowledge representation formalisms which can be used to represent the terminological and assertional knowledge of an application domain in a structured and formally well-understood way. The terminological knowledge, contained in the so-called T-box, represents the background knowledge and the knowledge about the terminology (classes and properties) relevant for the described domain. The assertional part, the so-called A-box, contains knowledge about the individuals which populate the given domain in the form of membership statements. Roughly speaking, in our framework the terminological part - which is the stable description of a given domain - is provided by the upper level modules of Figure 1. Instead, the changeable part, which corresponds to a specific process description shown in the bottom part of Figure 1, is provided in the form of assertional knowledge.

#### 3.1 An Ontology for BPMN

The BPMN specification [21] aims at the definition of: the building blocks of BPDs, their graphical representation, their attributes and properties, and how they can be combined to build a BPD.

Examples of BPMN elements are: *Event*, *Activity Gateway* and *Sequence Flow*. Properties of basic elements concern both the usage of the BPMN elements to compose the business process diagrams, and the behavior of the elements during the execution of a process. An example of property of the first kind is the one used to specify a *Start Event*:



*“A Start Event MUST NOT be a target for Sequence Flow; it MUST NOT have incoming Sequence Flow.”*[An exception follows.] (1)

A different example of property, which specifies the behavioral nature of a graphical element, is the following one. This property contributes towards the specification of the exclusive nature of the Sequence Flows which originate from an Exclusive Data-Based Gateway, that is, a Gateway which is used to indicate the place where a Sequence Flow can take two or more alternative paths:

*“if there are multiple outgoing Sequence Flow then only one Gate (or the DefaultGate) SHALL be selected during performance of the Process.”* (2)

BPMNO<sup>3</sup> (namely our BPMN ontology) provides a formalization of the structural part of BPDs, i.e. which are the basic elements of a BPD and how they are (can be) connected. BPMNO is not intended to model the dynamic behaviour of BPDs (that is, how the flow proceeds within a process). Ontology languages are not particularly suited to specify behavioral semantics. This part can be better modelled using formal languages for Workflow or Business Process Specification based on Petri Nets, as proposed in [18].

BPMNO is based on the latest stable BPMN specifications from OMG [21]. The ontology is structured according to the description of the complete set of BPMN Element Attributes and Types contained in Annex B of [21]. The ontology currently consists of 95 Classes and 439 Class Axioms, 108 Object Properties and 18 Object Property Axioms, and 70 Data Properties; it has the expressiveness of *ALC<sub>HORN</sub>(D)* and a textual description of its Description Logic version is contained in [14].

In BPMNO, besides organizing the BPMN objects in an *is-a* taxonomy, we encoded the attributes and properties which describe how to use these elements to compose business process diagrams. As a consequence of our effort towards the modelling of properties, BPMNO contains, in its current state, more than 400 class axioms which describe a wide set of properties of the BPMN elements. Due to expressiveness limitation imposed by Description Logics and by the fact that we want to remain in a decidable version of OWL, there is a limited (and documented) number of properties listed in [21] which are not represented in BPMNO. These properties concern: (i) attributes’ default values, and (ii) all the properties that, once translated into first order logic, require more than two variables. A typical example of this kind of properties is *“two objects cannot have the same object identifier”* or that *“all outgoing sequence flows connected to an inclusive gateway must have the same conditional expression attached”*.

### 3.2 Representing Criteria for Correct Semantic Annotations

Let us indicate with BDO a specific domain ontology contained in BPKB. Though belonging to different ontologies, concepts from BPMNO and BDO are not totally

<sup>3</sup> Available for download at [http://dkm.fbk.eu/index.php/BPMN\\_Ontology](http://dkm.fbk.eu/index.php/BPMN_Ontology)

unrelated. Consider for instance the ontologies shown in Figure 2. A BPMN activity, for example, could correspond to BDO actions, but not to BDO objects. Such kind of relationships can be generic or domain specific constraints that a business designer decides to impose. For instance, one would like to impose that BPMN subprocesses can not be annotated by `to_add_product` and `to_remove_product`, as these two actions in BDO are considered to be atomic. Conversely, one would like to impose that certain complex BDO actions (e.g. `to_manage`) should correspond to subprocesses in a BPD because they must be specified in more details.

To allow the business designer to specify this kind of positive and negative constraints between pairs of concepts, each belonging to one of the two ontologies, we introduce two relations: “*annotatable only by*” ( $\xrightarrow{AB}$ ) and “*not annotatable by*” ( $\xrightarrow{nAB}$ ) from BPMNO concepts to BDO concepts. Moreover, to allow the binding of a specific BDO concept only to a certain BPMNO element, instead of defining the  $\xrightarrow{nAB}$  relationship between each BPMNO element and a specific BDO concept, we introduce the symmetrical relations: “*annotates only*” ( $\xrightarrow{A}$ ) and “*cannot annotate*” ( $\xrightarrow{nA}$ ). These four relationships represent constraints on the valid labelling of BPD objects. This informal description is then translated into a more formal set of DL axioms, denoted with `Merging_Axioms(BPMNO, BDO)`, that formalise these constraints within the logical formalism. In the following table we report the intuitive meaning, and the formalization as DL axioms, of the four annotation restrictions introduced above. We use  $x$  to denote a concept of BPMNO and  $y$  to denote a concept of BDO.

Restriction	Intuitive meaning	DL axiom <sup>4</sup>
$x \xrightarrow{AB} y$	a BPMN element of type $x$ can be annotated only with a concept equivalent or more specific than $y$	$x \sqsubseteq y$
$x \xrightarrow{nAB} y$	a BPMN element of type $x$ cannot be annotated with a concept equivalent or more specific than $y$	$x \sqsubseteq \neg y$
$y \xrightarrow{A} x$	any concept equivalent or more specific than $y$ can be used to denote BPMN elements of type $x$	$y \sqsubseteq x$
$y \xrightarrow{nA} x$	any concept equivalent or more specific than $y$ can not be used to denote BPMN elements of type $x$	$y \sqsubseteq \neg x$

Figure 2 shows examples of constraints. `BPMNO:data_object  $\xrightarrow{AB}$  BDO:object` states that a BPMN data object is indeed an object of the domain ontology. `BPMNO:action  $\xrightarrow{AB}$  BDO:activity` states that an activity can be any kind of action (either an action itself or a subclass of action). However, if an activity is a subprocess, then it cannot be labelled by `to_add_product` and `to_remove_product`, as stated by `BPMNO:sub_process  $\xrightarrow{nAB}$  BDO:to_add_product` and `BPMNO:sub_process  $\xrightarrow{nAB}$  BDO:to_remove_product`. Finally the concept `to_manage` can be used only for annotating subprocesses, as stated by `BDO:to_manage  $\xrightarrow{A}$  BPMNO:sub_process`.

<sup>4</sup> Though the meaning of  $x \xrightarrow{nAB} y$  and  $y \xrightarrow{nA} x$  coincide, we provide both primitives as, depending on the case to be modelled, one may result more intuitive than the other.

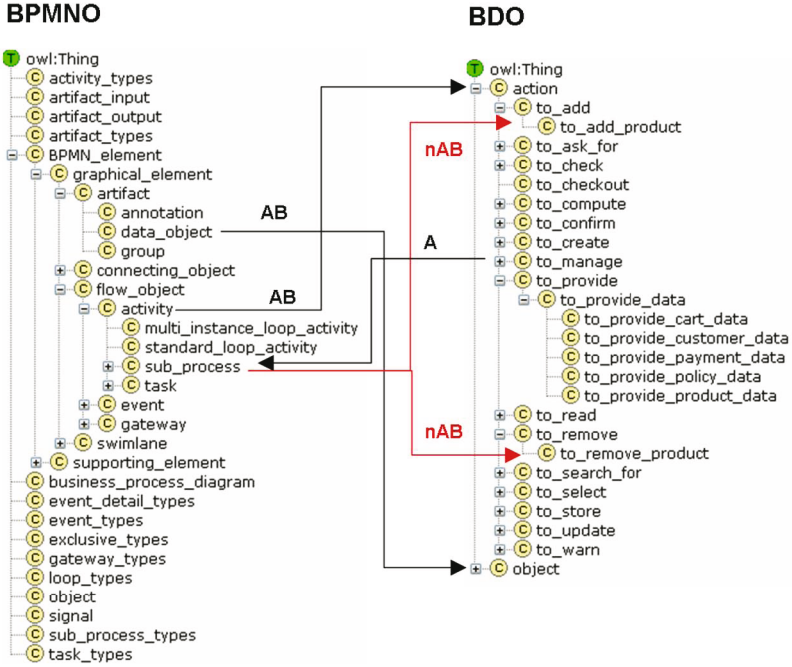


Fig. 2. Relationships between BPMNO and BDO

In the general case, some bindings specified by the user via the four primitives may generate inconsistencies in the integrated ontology. This situation can be automatically detected by verifying the consistency of  $BPMNO \cup BDO \cup \text{Merging\_Axioms}(BPMNO, BDO)$  via a DL reasoner. In these cases, suggestions can be given automatically for recovering from inconsistency.

### 3.3 Representing a Semantically Annotated BPD in an OWL A-Box

Given a semantically annotated BDO  $\beta$ , in this section we describe how it is possible to formalize it as an A-box  $A_\beta$  in the language of  $BPMNO \cup BDO$ . We explain it with the help of the sample process of Figure 3. This figure represents the sub-process that manages the addition/removal of items in a shopping cart of the on-line shopping process represented in full in [13] and not included here for lack of space. Semantic annotations are preceded by the “@” symbol. The main part of the A-box associated with this sub-process is shown in Figure 4.

The elements of the A-box  $A_\beta$  obtained from the annotated BPD  $\beta$ , the so-called BPD objects in Figure 4, are all the graphical objects of  $\beta$ . The assertions on these elements can be divided into three groups: *BPM-type assertions*, *BPM-structural assertions* and *BPM-semantic assertions*.

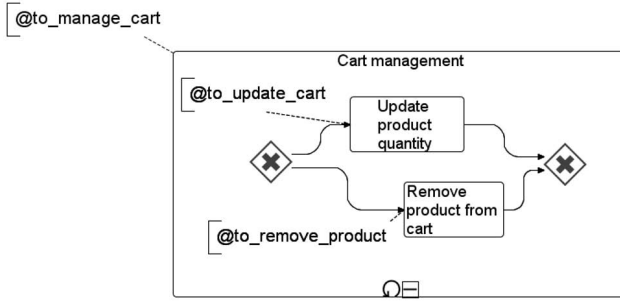


Fig. 3. A sub-process of the on-line purchase process

**BPM-type assertions:** for every graphical element  $g$  of type  $T$  occurring in  $\beta$ ,  $A_\beta$  contains the assertions  $T(g)$ , i.e.,  $g$  is an instance of concept  $T$ <sup>5</sup>. For instance the assertion `sequence_flow(s4)` in Figure 4 states that the BPM object  $s_4$  is of type `sequence_flow`.

**BPM-structural assertions** For every connecting object  $c$  of  $\beta$  that goes from  $a$  to  $b$ ,  $A_\beta$  will contain two structural assertions of the form `SourceRef(c, a)` and `TargetRef(c, b)`. For instance the assertion `has_sequence_flow_source_ref(s1, g1)` in Figure 4 states that the sequence flow  $s_1$  originates from gateway  $g_1$ .

**BPM-semantic assertions** For every graphical element  $g$  of the diagram which is annotated with a label  $C$  (where  $C$  is a complex concept expression of the domain ontology),  $A_\beta$  contains the assertion  $C(g)$ . For instance the assertion `to_update_cart(t1)` in Figure 4 states that task  $t_1$  is an instance of concept `to_update_cart` and is obtained from the semantic annotation `to_update_cart` of the BPD in Figure 3.

Given an OWL representation  $A_\beta$  we can reduce the problem of checking the correctness of the semantic annotation of the BPD  $\beta$  to a satisfiability problem in DL. In particular we can reformulate the fact that  $A_\beta$  represents a business process labelled correctly as the fact that  $\text{BPMNO} \cup \text{BDO} \cup \text{Merging\_Axioms}(\text{BPMNO}, \text{BDO}) \cup A_\beta$  is a consistent knowledge base.

The semantic annotation of the process in Figure 3 is consistent with the merging axioms `Merging_Axioms(BPMNO, BDO)` shown in the previous section. Assume, instead, to introduce a faulty annotation by replacing `@to_manage_cart` in Figure 3 with `@to_remove_product`. In this case the assertion `to_manage_cart(p1)` in Figure 4 is replaced by the assertion `to_remove_product(p1)`. This new assertion generates an inconsistent A-box. Indeed, `BPMNO` contains the axiom:

$$\text{embedded\_loop\_sub\_process} \sqsubseteq \text{sub\_process}$$

which allows to infer `sub_process(p1)` from the assertion `embedded_loop_sub_process(p1)`. Furthermore `Merging_Axioms(BPMNO, BDO)` contains the annotation restriction.

<sup>5</sup> For the sake of readability, we omit the `BPMNO` prefix in non ambiguous expressions.

<b>BPD objects</b>	
$p_1$	corresponds to the entire subprocess
$s_1, \dots, s_4$	correspond to the four sequence flow
$g_1$ and $g_2$	correspond to the left and the right gateways
$t_1$ and $t_2$	correspond to the top and bottom atomic task
<b>BPM-type assertions</b>	
<code>embedded_loop_sub_process(<math>p_1</math>)</code>	<i>/* <math>p_1</math> is an iterative subprocess */</i>
<code>data_based_exclusive_gateway(<math>g_1</math>)</code>	<i>/* <math>g_1</math> is a data base xor gateway */</i>
<code>data_based_exclusive_gateway(<math>g_2</math>)</code>	
<code>sequence_flow(<math>s_1</math>)</code>	<i>/* <math>s_1</math> is a sequence flow object */</i>
<code>sequence_flow(<math>s_2</math>)</code>	
<code>sequence_flow(<math>s_3</math>)</code>	
<code>sequence_flow(<math>s_4</math>)</code>	
<code>task(<math>t_1</math>)</code>	<i>/* <math>s_1</math> is an atomic task object */</i>
<code>task(<math>t_2</math>)</code>	
<b>BPM-structural assertions</b>	
<code>has_embedded_sub_process_sub_graphical_elements(<math>p_1, g_1</math>)</code>	
<code>:</code>	
<code>:</code>	<i>/* <math>p_1</math> contains <math>g_1, g_2, s_1 \dots s_4, t_1</math> and <math>t_2</math> */</i>
<code>has_embedded_sub_process_sub_graphical_elements(<math>p_1, t_2</math>)</code>	
<code>has_sequence_flow_source_ref(<math>s_1, g_1</math>)</code>	
<code>has_sequence_flow_target_ref(<math>s_1, t_1</math>)</code>	
<code>has_sequence_flow_source_ref(<math>s_2, g_1</math>)</code>	
<code>has_sequence_flow_target_ref(<math>s_2, t_2</math>)</code>	
<code>has_sequence_flow_source_ref(<math>s_3, t_1</math>)</code>	
<code>has_sequence_flow_target_ref(<math>s_3, g_2</math>)</code>	
<code>has_sequence_flow_source_ref(<math>s_4, t_2</math>)</code>	
<code>has_sequence_flow_target_ref(<math>s_4, g_2</math>)</code>	
<b>BPM-semantic assertions</b>	
<code>to_manage_cart(<math>p_1</math>)</code>	<i>/* <math>p_1</math> is an activity of managing of carts */</i>
<code>to_update_cart(<math>t_1</math>)</code>	
<code>to_remove_product(<math>t_2</math>)</code>	

**Fig. 4.** The encoding of the OnlineShop process in an OWL A-box

$$\text{BPMNO:sub\_process} \sqsubseteq \neg \text{BDO:to\_remove\_product}$$

corresponding to  $\text{BPMNO:sub\_process} \xrightarrow{\text{nAB}} \text{BDO:to\_remove\_product}$ , which implies  $\neg \text{BDO:to\_remove\_product}(p_1)$ . This last assertion is in contradiction with the assertion  $\text{BDO:to\_remove\_product}(p_1)$  generated by the incorrect labelling.

### 3.4 Automatically Encoding a BPD into an A-box

We developed a tool for the automated transformation of a BPD into an OWL A-box. Given  $\text{BPMNO}$ ,  $\text{BDO}$ ,  $\text{Merging\_Axioms}(\text{BPMNO}, \text{BDO})$  and a BPD  $\beta$  annotated with concepts taken from the domain ontology, the tool creates the A-box  $A_\beta$  and populates the ontology with instances of BPMN elements belonging to the specific process.

The input BPMN process is currently described in a `.bpnm` file, one of the files generated by both the Eclipse SOA Tools Platform and the Intalio Process Modeler tools. The `.bpnm` file is an XML file that contains just the structural description of the process, leaving out all the graphical details. The ontology population is realized by parsing the file and, taking advantage of a mapping file, instantiating the corresponding classes and properties in the BPKB T-Box. The mapping file associates XML tags/attributes used in the `.bpnm` file with BPMN

ontology concepts and properties. It is dependent on the particular process representation adopted by the tools generating the `.bpmn` file, hence it must be redefined or adjusted whenever a tool different from Eclipse SOA Tools Platform and Intalio Process Modeler are used for process creation and editing.

The BPMN process descriptions currently generated by the Eclipse tool or the Intalio tool do not exhaustively cover the features provided by the BPMN specification and, therefore, the full ontology potential. The mapping file is limited to the subset of the specification actually implemented by the considered tools. Sometimes the mapping is based on assumptions implicitly made by the tools (e.g. whenever a subprocess is created using the two considered tools, the concept to be instantiated is “embedded-subprocess”, not “subprocess”, since only embedded subprocesses can be created using the two tools).

Our transformation tool uses the `org.w3c.dom` XML parsing library to manage the `.bpmn` input file, Protégé libraries to populate the resulting OWL A-box, and Pellet for reasoning.

## 4 Use Cases

Some examples of the usefulness of semantic annotations associated with a Business Process Knowledge Base are shown in the following scenarios.

*Restricting to subclasses of semantically annotated BPD.* A first usage of the encoding of BPMN into an ontology is the possibility to impose additional restrictions on a BPD, or on how a BPD can be semantically labelled. These restrictions can be verified automatically via reasoning in the BPKM. We propose two main forms of constraints: (1) constraints on the BPD structure, independent from the labelling; and, (2) constraints on the structure that depend on the business domain.

In the first case, a new constraint on the BPD structure can be imposed by extending BPMNO with a new set of axioms that encode this restriction. For instance, suppose that a business designer, in order to facilitate the decisions that participants in the process have to make, wants to allow only binary exclusive decisions, hence rejecting both multiple and inclusive ones. This kind of constraint can be formalized by extending BPMNO with the following DL axioms:

$$\begin{aligned} \text{BPMNO:inclusive\_gateway} &\sqsubseteq \perp \quad /* \text{ no inclusive gateways } */ \\ \text{BPMNO:gateway} &\sqsubseteq (\leq 2)\text{BPMNO:has\_gateway\_gate} \quad /* \text{ gateway has at most 2 outgoing gates } */ \end{aligned}$$

In the second case (i.e. restrictions that involve the business domain), the new constraints deal with business domain specific rules. For instance, in an on-line shop process, the business expert may be interested, for security reasons, in guaranteeing that processing payment data is preceded by a customer checkout request. Such a constraint can be formalized in DL as follows:

$$\text{BDO:to\_provide\_payment\_data} \sqsubseteq \exists \text{BPMNO:connect} \bar{\sqsupset} . \text{BDO:to\_ask\_for\_checkout}$$

where `BPMNO:connect` is the transitive closure of the connections provided by the connecting elements<sup>6</sup>. In other words, the transitive `BPMNO:connect` relationship ensures that there exists at least a path in the model connecting two flow objects.

*Searching for Crosscutting Concerns.* Another possible application of semantic reasoning over process ontologies is related to the documentation, management, and evolution of the so-called *crosscutting concerns* (CC) in business processes [25]. In a business process, a crosscutting concern is any relevant feature that cannot be modularized in a single unit (i.e. activity, subprocess, etc.) of the process description. CCs are intrinsically scattered across the process and tangled with other concerns. For example, multiple process elements usually deal with user choices and there is no unique place where all such choices are made. Hence, comprehension and modification of user choices management requires a complex, time consuming understanding of the overall process and its parts. Ontologies can help us simplify this task.

Let us consider again the On-line Shop process from which we extracted the sub-process depicted in Figure 3. A critical design decision is how to manage the customer's preferences, for example in order to store and make them available in next accesses. As observed above, this is a quite typical example of crosscutting concern in a business process, since the customer usually expresses her preferences at different points in the workflow. We can mine for such points by formulating a query matching the "customer choice" concern. Figure 5 (top left) represents such a query in SPARQL [22]. The query matches all the places where the "customer choice" concern is dealt with in the BP, collecting the five tasks that represent a customer's choice (related to products, group products, quantity and shipping method).

In a second scenario, the business designer could explicitly look for the communication between the two pools in the On-line Shop process, so as to know all the on-line shop activities that follow an event generated by a customer activity (e.g. in order to apply a new event handler). Since the process as a whole is based on the messages exchanged between the two participants, this concern is scattered across the entire process (i.e. it is a CC). The query matching the "customer-shop communication" concern is formulated as the SPARQL query shown at the bottom of Figure 5. This query asks for all the on-line shop activities triggered by (i.e. immediately following) a customer event. Similarly to the "annotatable only by" constraints considered above, we can think to provide aid for an easier query formulation (e.g. by means of a visual query language [25]) and result representation (e.g. by highlighting the query results across the process), so as to avoid exposing SPARQL directly to the final user. Once a query capturing a CC of interest has been carefully formulated, it can be recorded as a form of process documentation. Whenever understanding such a CC will be important for some process design or evolution task, re-execution of the stored query will

<sup>6</sup> The relation `BPMNO:connect` can be formalized by the following axioms: `BPMNO:has_sequence_flow_source_ref`  $\sqsubseteq$  `BPMNO:connect`, `BPMNO:has_sequence_flow_target_ref`  $\sqsubseteq$  `BPMNO:connect`, and `Trans(connect)`.

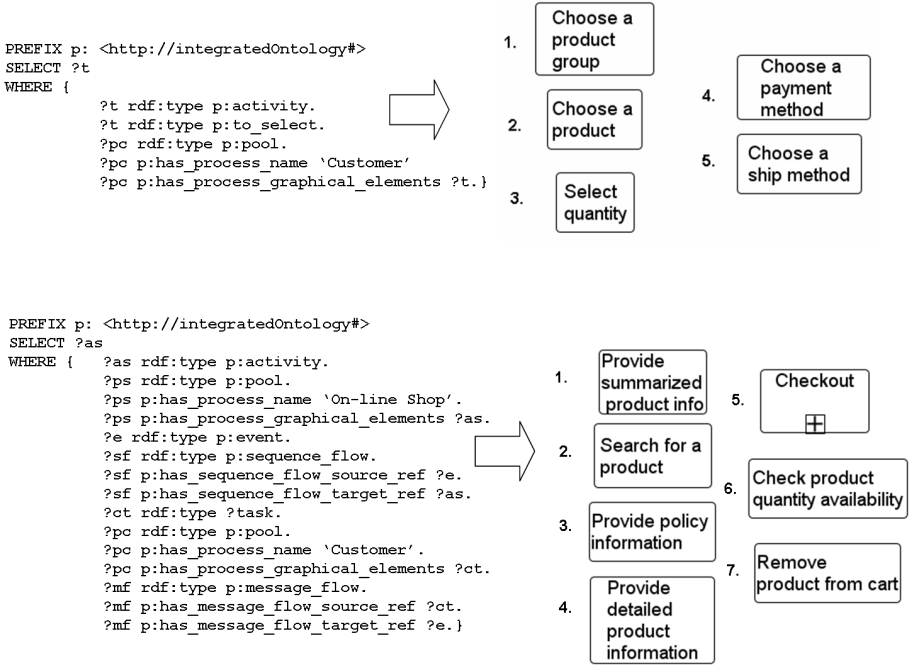


Fig. 5. Queries (on the left) and results (on the right)

immediately provide all involved process elements, even though they are possibly scattered and tangled with other process features.

## 5 Related Work

The idea of adding formal semantics to business processes is not new and a few approaches are already available for BPMN [26,10,27,18,5,11,24,19]. We can roughly divide the existing proposals into two groups: (1) those adding semantics to specify the dynamic behavior exhibited by a business process [26,27,18], and (2) those adding semantics to specify the meaning of the entities of a BPD in order to improve the automation of business process management [5,11,24,19]. We clearly belong to the second group.

Thomas and Fellmann [24] consider the problem of augmenting EPC process models with semantic annotations. They propose a framework which joins process model and ontology by means of properties (such as the “semantic type” of a process element). This differs substantially from our proposal, which establishes a set of subsumption (aka subclass or is-a) relations between the classes of the two ontologies being integrated (BPMN meta-model and domain ontology), instead of associating annotation properties to the process instances. This difference has a direct impact on the kind of constraints that can be automatically enforced (e.g.



BPMN elements annotatable by domain concepts). In particular, our approach supports automatic checking of the correctness of the semantic annotation. De Nicola *et al.* [19] propose an abstract language (BPAL) that bridges the gap between high-level process description (e.g. in BPMN) and executable specification (e.g. in BPEL). The formal semantics offered by BPAL refers to notions such as activity, decision, etc., while the problem of integrating process model and domain ontology is not their focus. In the SUPER project [11], the SUPER ontology is used for the creation of semantic annotations of both BPMN and EPC process models in order to support automated composition, mediation and execution. In [26], semantic annotations are introduced for validation purposes, i.e. to verify constraints about the process execution semantics. Our work represents an extension of the existing literature in that we provide an ontology integration scheme, based on hierarchical ontology merge, that supports automated verification of semantic constraints defining the correctness of semantic process annotations. Moreover, our proposal is compatible with top-level ontologies and allows rich and expressive queries, such as those necessary to identify, document and manage crosscutting concerns in business processes.

Other related works are in the area of crosscutting concerns in business processes, which has been mainly investigated with reference to executable process description languages. AO4BPEL [3] is a dynamic aspect oriented extension of BPEL [9], designed to be as close as possible to the aspect-oriented programming language AspectJ. Similar approaches, mainly differing from AO4BPEL for the choice of the advice language (Java), have been proposed by Courbis and Finkelstein [8] and by Verheecke, Cibràn and Jonckers [2], who defined WSML (Web Service Management Layer), based on a dynamic aspect oriented extension of JAsCo [11]. Padus [6] by Braem et al. is another aspect-oriented BPEL extension, without dynamic weaving and with some improvements that increase portability.

All these works mainly focus on the developers' perspective, without considering the business designers' one. Only a few attempts are available which try to address the problem of crosscutting concerns in business processes at an abstraction level similar to the business designers' one [17][15]. Though applied to a formal modeling language like Petri Nets, the approach proposed by Hornung et al. [17] supports the designer in the process modeling phase by suggesting a list of correct and fitting process fragments for completing the business process. It is based on business rules describing constraints on the specific business domain, similar to aspects, but expressed in a specific "if-then" syntax and defined before process modeling. Another similar, rule-based approach was proposed to support automated verification of compliance to constraints [15].

## 6 Conclusions

We have proposed a method to add semantic annotations to a business process, based on a set of merging axioms that connect BPMN ontology and domain ontology. Semantic annotations allow formal, automated reasoning on the elements and properties of a business process. Structural and domain specific constraints

can be expressed as axioms and can be verified as ontology consistency violations. Queries on the instances (i.e. actual process elements) can be defined to match relevant process features, such as crosscutting concerns.

In our future work, we will simplify the task of ontology merging for the final user by means of tools and algorithms that handle the typical inconsistencies generated in this step. We will also investigate user friendly notations for constraint and query specification. Another direction deals with moving from specification to executable process description languages, such as BPEL, for which the discovered CCs may be mapped to AO4BPEL aspects. Finally, we will validate the approach further, on larger case studies.

## References

1. Jasco: an aspect-oriented approach tailored for component based software development. In: AOSD, pp. 21–29 (2003)
2. Aspect-oriented programming for dynamic web service monitoring and selection. In: Zhang, L.-J. (ed.) ECOWS, LNCS. vol. 3250, pp. 15–29. Springer, Heidelberg (2004)
3. Mezini, M., Charfi, A.: Aspect-oriented web service composition with AO4BPEL. In (LJ) Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 168–182. Springer, Heidelberg (2004)
4. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, Cambridge (2003)
5. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: VLDB 2006, pp. 343–354 (2006)
6. Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W., Jonckers, V.: Isolating process-level concerns using padus. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 113–128. Springer, Heidelberg (2006)
7. Business Process Management Initiative (BPMI). Business process modeling notation: Specification (2006), <http://www.bpmn.org>
8. Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: ICSE 2005: Proc. of the 27th international conference on Software engineering, pp. 69–77. ACM, New York (2005)
9. Curbera, F., Goland, Y., Klein, Y., Leymann, F., Roller, D., Weerawarana, S.: Business process execution language for web services. Web page. Version 1.0 (July 31, 2002)
10. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of bpmn process models (2007), <http://eprints.qut.edu.au/archive/00005969/>
11. Dimitrov, M., Simov, A., Stein, S., Konstantinov, M.: A bpmo based semantic business process modelling environment. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management at the ESWC, CEUR-WS, vol. 251 (2007)
12. Wetzstein, B., et al.: Semantic business process management: A lifecycle based requirements analysis. In: Proc. of the Workshop on Semantic Business Process and Product Lifecycle Management, CEUR Workshop Proceedings, vol. 251 (2007)

13. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P.: Reasoning on semantically annotated processes. Technical report, FBK-irst (2008), <http://se.fbk.eu>
14. Ghidini, C., Rospocher, M., Serafini, L.: A formalisation of BPMN in description logics. Technical Report TR 2008-06-004, FBK-irst (2008)
15. Happel, H.-J., Stojanovic, L.: Ontoprocess – a prototype for semantic business process verification using swrl rules. In: Proc. of the 3rd European Semantic Web Conference (2006)
16. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In: ICEBE 2005: Proceedings of the IEEE International Conference on e-Business Engineering, pp. 535–540. IEEE Computer Society, Los Alamitos (2005)
17. Hornung, T., Koschmider, A., Oberweis, A.: A recommender system for business process models. In: 17th Annual Workshop on Information Technologies and Systems, December (2007)
18. Koschmider, A., Oberweis, A.: Ontology based business process description. In: Proceedings of the CAiSE 2005 Workshops. LNCS, pp. 321–333. Springer, Heidelberg (2005)
19. De Nicola, A., Lezoche, M., Missikoff, M.: An ontological approach to business process modeling. In: Proceedings of the 3rd Indian International Conference on Artificial Intelligence (IICAI), December 2007, pp. 1794–1813 (2007)
20. Fellmann, M., Thomas, O.: Semantic epc: Enhancing process modeling using ontology languages. In: Proc. of the Workshop on Semantic Business Process and Product Lifecycle Management at the ESWC, CEUR-WS, vol. 251 (2007)
21. OMG. Business process modeling notation, v1.1, <http://www.omg.org/spec/BPMN-/1.1/PDF>
22. Seaborne, A., Prud'hommeaux, E.: SPARQL query language for RDF. W3C recommendation, W3C (January 2008), <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>
23. Tarr, P.L., Ossher, H., Harrison, W.H., Sutton Jr., S.M.: N degrees of separation: Multi-dimensional separation of concerns. In: Proc. of the International Conference on Software Engineering (ICSE), Los Angeles, CA, USA, pp. 107–119. ACM press, New York (1999)
24. Thomas, O., Fellmann, M.: Semantic epc: Enhancing process modeling using ontology languages. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM), June 2007, pp. 64–75 (2007)
25. Tonella, P., Di Francescomarino, C.: Business process concern documentation and evolution. Technical report, FBK-irst (2008), <http://se.fbk.eu>
26. Weber, I., Hoffmann, J., Mendling, J.: Semantic business process validation. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM) (June 2008)
27. Wong, P., Gibbons, J.: A Relative Timed Semantics for BPMN (submitted, 2008), Extended version <http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmntime.pdf>

# Event-Driven Quality of Service Prediction

Liangzhao Zeng, Christoph Lingenfelder, Hui Lei, Henry Chang

IBM T.J. Watson Research Center Yorktown Heights, NY 10598  
{lzeng,hlei,hychang}@us.ibm.com, lin@de.ibm.com

**Abstract.** Quality of Service Management (QoSM) is a new task in IT-enabled enterprises that supports monitoring, collecting and predicting QoS data. QoS solutions must be able to efficiently process runtime events, compute and predict QoS metrics, and provide real-time visibility and prediction of key performance indicators (KPI). Currently, most QoS systems focus on monitoring of QoS constraints, i.e., they report what has been happened. In a way, this provides the awareness of past developments and sets the basis for decisions. However, this kind of knowledge is afterwit. For example, it cannot provide early warnings to prevent the QoS degradation or the violation of commitments. In this paper, we move one step forward to provide QoS prediction. We argue that performance metrics and KPIs can be predicted based on historical data. We present the design and implementation of a novel event-driven QoS prediction system. Integrated into the SOA infrastructure at large, the prediction system can process operational service events in a real-time fashion, in order to predict or refine the prediction of metrics and KPIs.

## 1 Introduction

In order to function effectively in today's business environment, organizations must have transparent business services and know the operation performance at all levels and at all times. This allows them to stay competitive and profitable. QoSM is a new generation of business event processing systems that focuses on monitoring service operations. It provides a comprehensive view of service operations in organizations. Adopting a QoS solution provides the following benefits:

- Increased revenue by speeding up response time, actions and regulatory changes;
- Effective risk management by providing information in the right context to facilitate decision making;
- Improved customer satisfaction by facilitating continuous improvement of business services.

Currently, most QoS solutions focus on QoS monitoring and reporting the past and the present state of the services. In general, QoS information can be classified into two categories, metrics and Key Performance Indicators (KPIs). Metrics are used to measure properties of individual service instances, while KPIs are used to aggregate the metrics. An example of a KPI can be the average turnaround time for service instances in the second quarter. Both types of QoS information help to understand the

current status of service operations. However, such retrospection does not provide enough information to understand what might happen in the future. Indeed, if a preview were available, then actions to improve business performance can be performed in advance to prevent undesired situations.

The rapidly increasing popularity of a SOA infrastructure in enterprises gives us an incentive to integrate QoS prediction as part of the SOA infrastructure. There are two main issues in designing and implementing such a prediction system:

- *Automated data collection.* The prediction system needs to collect not only historic metric and KPI values, but also related events and environment data. It should be noted that these events and environment data are snapshots of other variables, which are important for prediction, as they might influence the performance result at that timestamp or later. In most of the enterprise business intelligence projects, about 80% of the development effort is spent on data collection and transformation. It is critical to offer a systematic approach to collect and transform the data.
- *Real-time QoS prediction.* It is important to provide early prediction when the situation changes. However, prediction is not a simple computation. The complexity of prediction stems from two aspects: large amounts of data need to be processed and the nature of the dependency is unknown. For example, for time series prediction, the more relevant historic data is used, the better the quality of prediction can be. In most business intelligence systems, manual programming effort is performed to prepare the data. And usually the data are processed in a batch model, which indicates that the system may not support prompt response to changing situation. As the metric and KPI to be predicted can be very dynamic, it is a challenge to provide real-time prediction on QoS.

In order to tackle the above challenges, we design and implement an event-driven QoS prediction system. It enables declarative QoS prediction in the SOA infrastructure. It employs a collection of event-analysis techniques to improve the accuracy of predictions. In a nutshell, the main contributions of this paper are:

- *QoS Prediction-enabled SOA Infrastructure.* Building upon our previous work on QoS monitoring system that report past performance results, we further enrich the SOA infrastructure to enable QoS prediction. Such an extension enables metric and KPI predictions without programming effort. To detect operational service events, QoS management needs to be integrated into the SOA infrastructure at large. It is important to leverage existing components in the SOA infrastructure, and to enable detection and routing of the events systematically. Further, the prediction system needs to collect historic QoS data in order to use up-to-date data to make predictions. Therefore, it is also important to integrate the prediction function as part of the monitoring infrastructure.
- *Event-Driven QoS Prediction.* We design a novel event-driven QoS prediction mechanism. In order to support real-time prediction, our solution is equipped with an event pattern processing module. First, the event-driven mechanism facilitates the collection of data whenever they become available. The freshness of data is critical for the precision of prediction models. Second, the event-driven mechanism is adopted to trigger the prediction of metrics and KPIs when it is necessary, which enables real-time or on demand prediction with minimal overhead.

The rest of this paper is organized as follows. Section 2 presents the service QoS management programming model. Section 3 presents the overall system architecture. Section 4 gives details on metric prediction. Section 5 discusses KPI prediction. Following the discussion on related work in Section 6, Section 7 contains concluding remarks.

## 2 QoS Management Metamodel

In this section, we discuss the proposed programming model dubbed *QoS Management Metamodel* (see Figure 1). In the metamodel, a *ServiceMonitorContext* contains a collection of *BusinessEvents*, *PerformanceMetrics* and *PerformanceKPIs*. A *BusinessEvent* describes status changes in a service. A *BusinessEvent* can be raised when a service initiates an execution, wherein the event may contains the service ID, time stamp, value of input parameters, and etc. A *PerformanceMetric* describes quality of a service execution. For example, an execution duration of a service can be a *PerformanceMetric*. A *PerformanceKPI* is the aggregation of a collection of *PerformanceMetrics*. For an example, a *PerformanceKPI* can be the average execution for a service in a given period. Basically, there are two aspects of the proposed QoS management metamodel: monitoring and prediction. In the following subsections, these two aspects are presented.

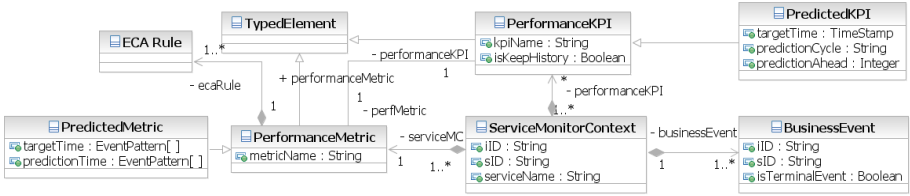


Fig. 1. Simplified Class Diagram of the QoS Management Metamodel

### 2.1 QoS Monitoring Aspects

QoS monitoring is the cornerstone for enabling predictions as it records what is happening. To describe the metric computation logic, we adopt Event–Condition–Action (ECA) rules (cf. Expression 1) to describe when and how the metric values are computed. Such a rule-based programming model frees users from the low-level details of procedural logic.

$$\text{Event}(\text{eventPattern})[\text{condition}] \mid \text{expression} \quad (1)$$

In an ECA rule, the event pattern component indicates business events. The condition component in a rule is a Boolean expression specifying the circumstances to perform the action described in the expression component. The expression consists of an association predicate and a value assignment expression. The association predicate specifies which monitor context instance should receive the event. The operators allowed

in the predicate expressions include relational operators, event operators, scalar, vector and set operators, Boolean operators and mathematical operators. An example ECA rule for metric computation is given in equation (2).

$$Event(E_1 :: e)[e.a_2 > 12] \mid (MC_1.iID == e.iID) \quad MC_1.m_2 := f_1(e) \tag{2}$$

In the above example, when an instance of event  $E_1$ , denoted as  $e$ , occurs, if  $e.a_2 > 12$ , then the event is delivered to the instance of  $MC_1$  whose  $iID$  matches the  $iID$  field (ID for service instance) of event instance  $e$ , and the metric value of  $m_2$  is computed by function  $f_1(e)$ . When there is no matching context instance, a new monitor context instance is created. It should be noted that the monitor context represents the process that is being monitored. Another example ECA rule is given in equation (3). In this example, when the value of metric  $MC_1.m_2$  changes, the value of metric  $MC_1.m_3$  is updated by function  $f_2(MC_1.m_1, MC_1.m_2)$ .

$$Event(changeValue(MC_1.m_2)) \mid MC_1.m_3 := f_2(MC_1.m_1, MC_1.m_2) \tag{3}$$

Usually KPI values are computed by aggregating metric values within specific time windows. Equation (4) shows a KPI expression:

$$MC_1.k_3 := sum(MC_1.m_2, repeat[2 Week]) \tag{4}$$

In this example, the KPI  $k_3$  is defined as the sum of metric  $m_2$  in two calendar weeks. Currently, two kinds of time windows are supported: (i) calendar time window, for example, week, month, year, (ii) sliding window, for example the last ten days, and (iii) expanding window or running total, i.e., from a past timestamp, for example from the beginning of the year until now. For the first kind of KPI, the computation is either triggered periodically, or computed on demand by a user’s request. The second and third kinds of KPI usually need to be computed whenever the base metric values are created or updated.

### 2.2 QoS Prediction Aspects

The prediction aspect is defined based on the monitoring model. We discuss the prediction of metrics first. By default, we predict the final value of a metric, i.e., the value the metric has at the time when the service instance is completed. Further, a customized *prediction target time* can be identified as an event pattern, for example, “ $E_1/E_2/E_4$ ”, which indicates to predict the metric value at the time when events  $E_1, E_2$ , and  $E_4$  have occurred in that order.

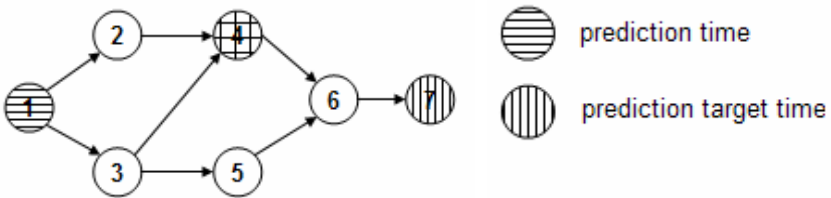


Fig. 2. Prediction Time and Prediction Target Time

Another notion in metric prediction is the *prediction time*, when the system makes the prediction. A prediction can be triggered whenever a new event occurs in a process instance. In this case, more information has become known, and it will usually be possible to make a more precise prediction of the metric value at prediction target time (node 7 or node 4 in Figure 2). Usually, the prediction time will be customized using event patterns. In the example of Figure 2, two prediction times have been defined, one after the initialization event  $E_1$  and the other after the event sequence “ $E_1/E_2/E_4$ ”. For these two times, predictive models  $M_1$  and  $M_{124}$  are computed respectively, based on historic cases of instances that went through these sequences of events in that order. Note that a less accurate model  $M_4$  could be built using all instances that reached node 4 regardless of their paths.

If a prediction for some instance is made, the choice of predictive model depends on the event sequence through which the instance went. If the instance is in node 1  $M_1$  is used. In node 2 or 3  $M_1$  must still be used, because no additional prediction time was defined, and hence no predictive model computed. For instances in nodes 4  $M_{124}$  can be used, if the instance went through node 2, otherwise  $M_1$  must still be used.

As KPIs are defined based on aggregation of metrics across a time window, the target time of the prediction can be either a future timestamp, or a collection of future and possibly past timestamps. Usually, when a KPI is defined based on a calendar time window, the target prediction time is defined based on the calendar. For example, if the KPI computes a monthly running total, the user may be most interested in the value of the KPI at the end of the current or some future month. For the specification of when the system makes the prediction in case of a KPI, it is usually performed on demand, or periodically.

### 3 System Architecture

In this section, we present the overall system architecture (see Figure 3) that realizes event-driven QoS management in a SOA infrastructure. Basing on the generic SOA infrastructure, four specific components that enable QoS monitoring and prediction are introduced.

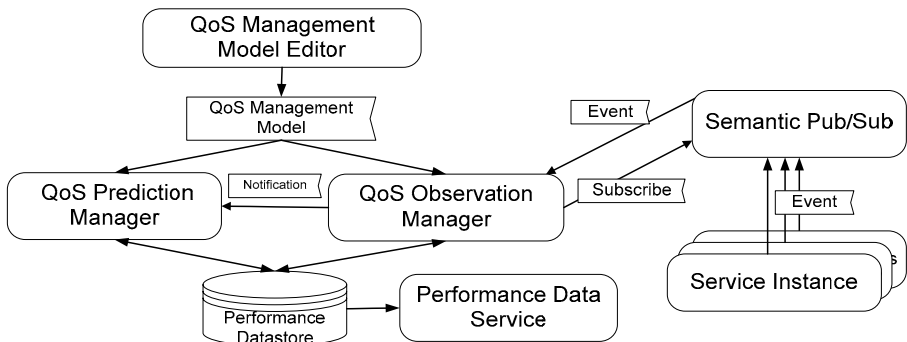


Fig. 3. Simplified QoS Prediction-enabled SOA infrastructure



The *QoS Management Model Editor* provides interfaces that allow users to create a monitor model. The *QoS Observation Manager* subscribes events and computes metric and KPI values. The *QoS Prediction Manager* receives notifications and predicts metrics and KPIs. It should be noted that we utilized *Semantic Pub/Sub*[9], for routing the business events from service instances to the QoS Observation Manager semantically. In this section, we only briefly discuss the QoS monitoring. Detailed techniques for high performance event processing for metric and KPI value computation are given in [11]. In the next two sections, the discussion of QoS prediction is presented.

As discussed in section 2, the programming model for metric and KPI computation is event-driven, i.e., the values are computed according to the occurrence of events. Given the high volume of events and the complexity of the computations, the system performance on event throughput is critical. In our design we advocate model-analysis techniques to improve event throughput. In the build time, a series of model analyses of the application logic are conducted to understand such factors as runtime data-access path, data flow, and control flow. Such analyses can be used to improve throughput in three ways: at build time it can be used to facilitate the generation of customized code to optimize I/O and CPU usage; information about the control flow and data flow can be used to ensure that CPU resources are used effectively by distributing event-processing computation logic evenly over time; and at runtime, knowledge gained from the model can be used to plan multithreaded parallel event-processing execution to reduce wait states by maximizing parallelization and reducing the planning overhead.

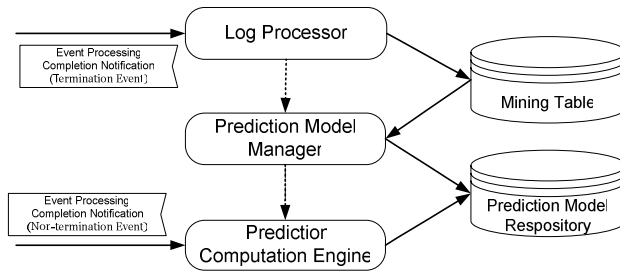
It should be noted that, from a monitoring perspective, the system is only concerned with the final values of metrics. However, in order to predict QoS, persistence of both final values and the changes of values in the history of metrics are required. Therefore, in our design, whenever a metric value is initiated or updated, a metric change log is appended. The format of the log entries is `<iID, timestamp, eventID, metricID, newValue>`, where `iID` identifies a service instance (or context instance), `timestamp` indicates the time when the event was received, `eventID` is the ID of the event, `metricID` refers to the changed object, `newValue` represent the metric's new value in a lexical format (the actual data type is the metric's data type and can be one of Boolean, integer, decimal, string, duration, dateTime, date, or time).

As KPI values are aggregated metric values associated with time windows, real time computation and persisting are usually required. For example, a user may request to keep the history of a monthly (sliding-window) KPI on a daily basis, which requires the system to compute the KPI values daily and persist them. In our system, we develop a history management module to manage historical KPI values. It should be noted that the historic value of a KPI can be used for time series KPI prediction, which is presented in section 5.

## 4 Event-Driven Metric Prediction

As discussed earlier, metric values are related to the service instance. When making the predictions of metrics, understanding the execution progress of the service is critical. And usually, as the execution is going on, more and more accurate prediction should become possible. One of the naive solutions for understanding the execution

progress is to access the process schemas and map the execution status to their control flow structure. However, process schemas are not always available, especially in heterogeneous environments. In order to overcome this limitation, we propose a novel event-driven approach. Instead of understanding the process schema, the system understands the service instance execution progress by the event sequence that occurs in the service instance. In fact, what is happening in a service instance can be identified as an event sequence. For example, the execution progress given by event sequence “ $E_1/E_2/E_3$ ” must be following the one represented by “ $E_1/E_2$ ”. Such a design allows the system to manage a larger spectrum of services, for example legacy services that do not have formal process schema definitions or services that do not expose their process schemas.



**Fig. 4.** Metric Prediction in QoS Prediction Manager

With the event-driven mechanism, we adopt an approach based on data mining to predict metric values for services. Our metric prediction consists of three phases: mining data preparation, prediction model creation and prediction result scoring. Accordingly, there are three components (see Figure 4) to realize these three functions: The *Log Processor* processes the metric change logs and monitor data to populate the mining data. It initiates processing when it receives a processing completion notification of a service termination event from the QoS monitor manager. Once the mining data is ready, it notifies the *Prediction Model Manager*. The *Prediction Model Manager* determines whether the mining data is rich enough to create a prediction model, or whether a refinement of a prediction model is required when a prediction model already exists. When prediction models are created or refined, the *Prediction Model Manager* notifies the *Prediction Computation Engine* that prediction models are ready for prediction. When the *Prediction Computation Engine* receives a processing completion notification of a service termination event, it identifies the prediction model and uses the latest independent metric values to compute the predicted metric value. In this section, we present the details of each phase.

#### 4.1 Mining Data Preparation

In this subsection, we first illustrate the data schema (see Figure 5) of mining data and then present the details of data population. The data schema consists of tables *ServiceInstanceTable\_T*, *ServicePerformance\_T*, *EventSequence\_T*, *IndependMetricSnapshotValue\_T* and a collection of views *MiningData\_V\_i*.

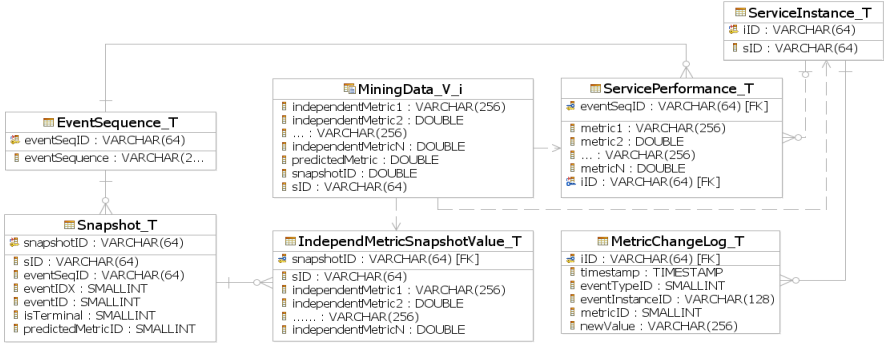


Fig. 5. Simplified Mining Data Schema

- **ServiceInstanceTable\_T.** This is the service instance and service type lookup table that is managed by the QoS monitor manager.
- **ServicePerformance\_T.** This table schema is generated, in particular, the column  $metric_i$  is generated according to a performance metric that is associated with the service context. In the table,  $iID$  identifies a service instance;  $eventSeqID$  is a foreign key for table **EventSeq\_T**, which holds an identifier for the sequence of all events that leads to the termination of service instance, ending in a terminal event;  $metric_i$  holds the final values of metrics (values after completion of processing the instance’s terminal event). In our design, **ServicePerformance\_T** is used to save final values of metrics for each service instance, by the QoS monitor manager.

**EventSeq\_T.** In this table, each entry indicates an event sequence that leads to the termination of service instance execution for service type  $sID$ . This table is populated using table **MetricChangeLog\_T**. The detailed algorithm (see Algorithm 1) is given as followed. For the sake of simplified presentation, we use procedural SQL type of pseudo code to illustrate the algorithm 1. Lines 1-6 search for distinct event instances that lead to the termination of a service instance. Lines 7-9 construct the event sequence pattern. An example of an event pattern can be “ $E_1/E_2/E_3/E_2/E_5$ ”. It should be noted that the invocation of the algorithm occurs when processing of a termination event is completed by the QoS monitor manager.

- **Snapshot\_T.** In this table, event sequences are used to identify the snapshot of independent metric values at prediction time. Instead of directly storing the event sequence, two attributes  $eventSeqID$  and  $eventIDX$  are used. For example, an event sequence with  $eventSeqID$  **ES\_0247** might consist of an event sequence “ $E_1/E_2/E_4/E_5$ ”. Then, for each predicted metric, there may be up to four entries in the table, as the event sequence has four subsequences, including itself, which are “ $E_1$ ”, “ $E_1/E_2$ ”, “ $E_1/E_2/E_4$ ” and “ $E_1/E_2/E_4/E_5$ ”. The entry  $\langle ES\_0247, 3, E_4, M\_17, TRUE \rangle$ , for example, will represent the partial event sequence “ $E_1/E_2/E_4$ ”.  $isTerminal$  indicates whether the event instance terminates the service instances.

predictedMetricID identifies the metric to be predicted. The population of the table is based on the new entry in table EventSeq\_T, wherein each subsequence of event sequence can create an entry for the table Snapshot\_T. To be more specific, when the algorithm 1 returns a new event sequence record, then N records are inserted into table Snapshot\_T, where N is the length of the event sequence.

---

INPUT: Event e, a service termination event  
 OUTPUT: a record in table EventSequence\_T

1. EventSequence eventSeq = null;
2. **Set** logCollection = null
3. **Select** distinct m.eventTypeID, m.eventInstanceID **Into** logCollection
4. **From** MetricChangeLog\_T m
5. **Where** m.sID = e.sID
6. **Order by** m.timeStamp
7. **For each** record in logCollection **Do** {
8. eventSeq = eventSeq.append(record.eventTypeID+ '/')
9. }
10. **If not** (eventSeq **Existing In** EventSeq\_T) **Then** {
11. **Insert into** EventSeq\_T **Value** (newID, eventSeq)
12. **Return** new (newID, eventSeq) } **Else** {
13. **Return** (eventSeqID, eventSeq)
14. }

---

**Algorithm 1.** Populating table EventSeq\_T

- IndependMetricSnapshotValue\_T. In this table, iID is used to join the table ServicePerformance\_T for creating the mining data view Mining-Data\_V\_i. The attribute snapshotID identifies the prediction point. independentMetric<sub>i</sub> hold the values of independent metrics at the time of the given event sequence. This table is populated from metric table MetricChangeLog\_T, using the algorithm (see Algorithm 2). In algorithm 2, lines 1-5 collect all the log entries in table MetricChangeLog\_T; lines 6-22 form a loop, based on the number of subsequences in eventSeq. For each subsequence, the related metric value needs to be collected as snapshot data. In lines 7 through 11, all the entries associated with the final event of the subsequence are collected. It should be noted that we allow repeating events in the event sequence of a service instance. For example, in “E<sub>1</sub>/E<sub>2</sub>/E<sub>3</sub>/E<sub>2</sub>/E<sub>5</sub>”, E<sub>2</sub> occurs twice. Then it is possible that two different event instances of E<sub>2</sub> appear in the metric change log, i.e., count(distinct(logCollection.eventInstanceID))>1. In such cases we need to locate the earlier event entry first, which is done in lines 13-16. Function **InsetRecordToMetricSnapshotTable()** is used to insert an entry to IndependMetricSnapshotValue\_T, using the metric values in the metric change log.

---

INPUT: Event  $e$ , a service instance termination event; and associated record event sequence  $seq$  in table `EventSequence_T`

```

1.   Set MetricChangeLogCollection = null
2.   Select * Into MetricChangeLogCollection
3.   From MetricChangeLog_T m
4.   Where M.sID = e.sID
5.   Order by m.timeStamp
6.   For (i = 1; i < eventSeq.length; i++) {
7.     Set logCollection = null;
8.     eventTypeID = eventSeq.elementAt(i);
9.     Select * into logCollection
10.    From MetricChangeLogCollection M
11.    Where M.eventTypeID = eventTypeID
12.    IF count(distinct(logCollection.eventInstanceID))>1 Then {
13.      Select min(timeStamp) As earliestTimeStamp From logCollection
14.      Select * into earliestLogCollection From logCollection
15.      Where timeStamp = earliestTimeStamp
16.      MetricChangeLogCollection -= earliestLogCollection
17.      InsetRecordToMetricSnapshotTable(eventSeq.subseq(i),
        earliestLogCollection)
18.    } else {
19.      MetricChangeLogCollection -= logCollection
20.      InsetRecordToMetricSnapshotTable(eventSeq.subseq(i), logCollection)
21.    }
22.  }

```

---

**Algorithm 2.** Populating table `IndependMetricSnapshotValue_T`

- `MiningData_V_i`. These views are generated, one for each prediction point and predicted metric. The views join the table `Snapshot_T` and `ServicePerformance_T`. In this view, `predictedMetric` is the metric to be predicted, while `independentMetrici` are the independent variables for creating the predictions (see Algorithm 3)

---

INPUT:  $sID$ , a service type ID; and associated predicted metric ID `MetricX`

```

1.   Select snapshotID into snapshotCollection
2.   From Snapshot_T
3.   Where Snapshot_T.sID = sID
4.   For each snapshotID in snapshotCollection {
5.     Create View MiningData_V_j As
6.     Select sID, snapshotID, independentMetric1, ..., independentMetricn, S.MetricX as
        predictedMetric
7.     From IndependMetricSnapshotValue_T I, ServicePerformance_T S
8.     Where I.sID = sID and I.snapshotID = snapshotID and I.sID = S.sID
9.   }

```

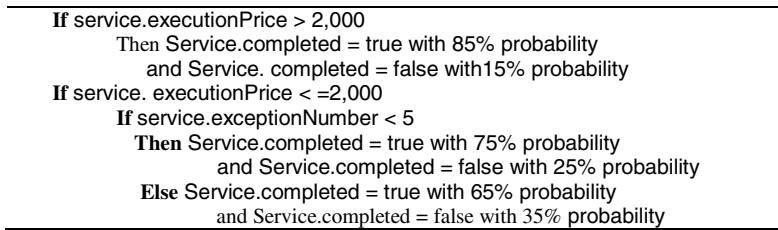
---

**Algorithm 3.** Creating view `MiningData_V_i`

In algorithm 3, lines 1-3 identify all the snapshots for service type sID; lines 4-9 create a view for each snapshot. This algorithm is involved when a metric is identified for predictions or a new event sequence is discovered for the service type sID.

## 4.2 Prediction Model Creation

Based on the predicted metric's data type, different prediction models are generated. For decimal, integer, duration, time, datetime, and date type of metrics, regression is used to create a prediction model, while for Boolean and string types of metrics, a classification models such as decision trees are used to create predictive models. When creating the models, the data in `MiningData_V_i` are split randomly for training and testing. If there are not enough data for this approach standard statistical techniques such as cross-validation are used. Once the Prediction Model Manager receives the mining and testing tasks, it creates prediction models and saves them in a Prediction Model Repository. An example model (decision tree) is shown in Figure 6.



**Fig. 6.** An Example of a Decision Tree

In this example, the first leaf indicates when the metric `executionPrice` is greater than 2,000, then with 85% probability that execution service instance will be completed while with probability of 15% that execution service instance will not be completed.

## 4.3 Prediction Scoring

When the prediction models are created, prediction scoring can be performed by the Prediction Computation Engine, using the latest independent metric data. When the processing of a non-termination event is completed, the QoS Monitor Manger notifies the Prediction Computation Engine, with updated metric values of the service instance. These metric values are used as inputs for the prediction model to make a prediction. Using the above example of a prediction model, if current service execution price is 1,500 and exception number is 4, the prediction of metric completed is true with probability of 75% and false with 25%. It should be noted that the prediction scoring module also feeds a prediction model quality manager. It monitors the accuracy of the prediction for each prediction model. In case the accuracy is decreasing, it will notify the prediction model manager to re-mine the models on more recent data.

## 5 KPI Prediction

We propose two approaches to perform KPI predictions, namely time series and metric-aggregation. In this section, we present these two approaches in detail.

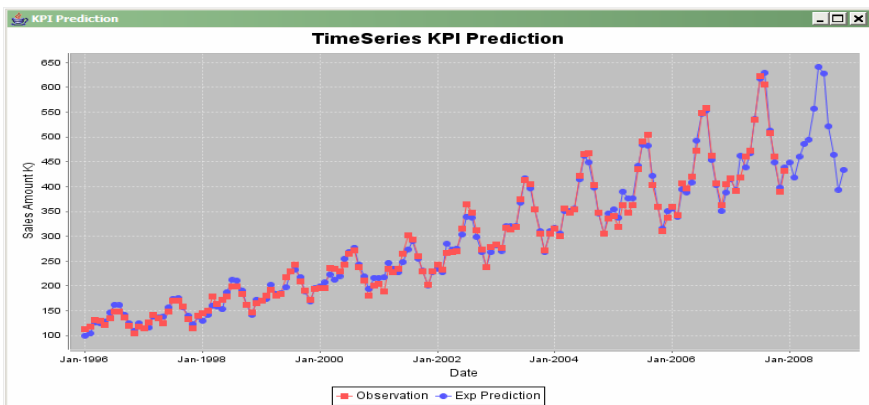
### 5.1 Time Series KPI Prediction

As discussed in section 3, our observation manager not only computes the KPI value, but also manages the historic values of the KPI. These historic KPI values can be considered to form a time series. Any standard time series prediction algorithm can be adopted, for example exponential smoothing [16] [17] [18] or ARIMA [19]. In our current implementation, an exponential smoothing method is adopted.

In many business applications both metrics and KPIs exhibit a periodic behavior with seasonal cycles. Gas consumption differs by month, sales figures often depend on the quarter or on the day of the week. Such seasonal cycles can in theory be detected automatically, for example using Fourier Transformation. For all practical purposes it can, however, be assumed that the user knows the length of seasonal cycles.

In our experience, in most of the cases, the seasonality of service KPIs is related to normal calendar periods: they oscillate hourly, daily, weekly, monthly, quarterly or yearly. A second issue occurs for running total KPIs (that aggregate with COUNT or SUM over a window of growing size). A KPI could for example measure the total sales in the current month, starting at a small value on the first of the month and increasing throughout the month. This behavior repeats in every month, seemingly showing a monthly seasonality. This is however not a true seasonality; in fact the seasonality might well be weekly with higher sales on the weekends. To handle that, we perform a differentiation, effectively defining a new KPI ‘Daily Sales’ which can be predicted in the usual way. When the prediction of the original (running total) KPI is done, integration is performed to compute the predicted values of the original KPI.

Figure 7 illustrates one of our KPI prediction results. Monthly sales amount history data are used to build an exponential smoothing prediction model.



**Fig. 7.** An Example of a Time Series Prediction

## 5.2 Metric-Aggregation KPI Prediction

In this approach, KPI prediction is performed based on the prediction of base metrics. Given to predict a KPI value in a target time, its value is based on metrics in three kinds of service instance (see Figure 8):

- A) the service instances are completed between the start time of the KPI window and the current time,
- B) the service instances are already initiated and may be completed before the target prediction time, and
- C) the service instances that may be initiated and completed between now and the target time.

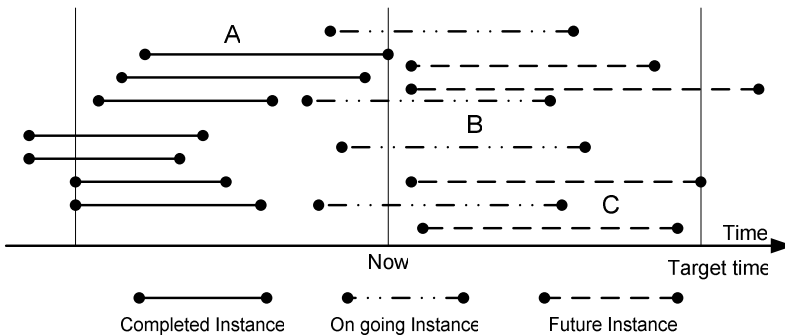


Fig. 8. Three kinds of service instances

For the completed instances metric values have been computed by the QoS Monitor Engine, so no predictions are necessary. For the ongoing instances, predict the base metric values and use a classification model to predict the probability whether the instances will be completed before the target time. For future instances, first, time series based prediction is required to predict how many instances are expected to be initiated between now and the target time. Second, a prediction of the percentage that will be completed before the target time is also required.

## 6 Related Work

In this section, we review work in the areas of QoS management. QoS monitoring has been widely studied in the context of middleware systems [1,10,11]. These efforts have addressed the following issues: QoS specification to allow description of application behavior and QoS parameters, QoS translation and compilation to translate specified application behavior into candidate application configurations for different resource conditions, QoS setup to appropriately select and instantiate a particular configuration, and QoS adaptation to runtime resource fluctuations. Most efforts in QoS-aware middleware, however, are focused on the network transport and system level. Little work has been done at the application and business service levels.



QoS-Aware service composition [2-7,10,12,13] aims at selecting component services to optimize the overall QoS of a service. In [2,7], the system assumes pre-existing QoS information of components, and a future QoS metric value is predicted as average of historic values. In [8], the formulas that compute the QoS of a workflow based on both the QoS of component services and the workflow schemas are discussed. However, no meaningful predictions are provided in all these work. In [5], a QoS-aggregation system is presented. It provides an editor for the QoS aggregation function that allows users to specify QoS attributes and their aggregation formulas. It also provides an interpreter that evaluates a workflow's global QoS. Again, it does not provide the prediction of QoS measurement. Different from the above works, this paper tries to tackle the prediction of QoS. The prediction of QoS enables the next level of optimization of QoS for services.

Intelligent business processes execution analysis [14] focuses on analyzing business process execution log, in order to provide metric prediction. It enables the automated prediction. However, process definitions are required when performing the analyzing the execution log. This may limit the scope of the services for which QoS management can be deployed. Especially in a heterogeneous environment, services may not able to expose their process definition, while about to emit the execution event to inform about the progress. Therefore, event based prediction systems like ours are desired. Further, the event-driven mechanism facilitates real-time prediction, which is critical to creating higher QoS.

## 7 Conclusion

In this paper, we advocate prediction of the QoS for services, using the data that is available through QoS monitoring. We design and implement a novel event-driven QoS prediction that can support automated service metric and KPI prediction in real time fashion. Our future work includes integrating Bayesian network prediction and other prediction framework, as well as a careful study of the system performance.

## References

1. Menasce, D.A.: QoS Issues in Web Services. *IEEE Internet Computing* 6(6) (2002)
2. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality Driven Web Services Composition. In: *WWW 2003* (2003)
3. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition based on Genetic Algorithms. In: *GECCO 2005*. ACM Press, New York (2005)
4. Canfora, G., Di Penta, M., Esposito, R., Perfetto, F., Villani, M.L.: Service composition (re)Binding driven by application-specific qoS. In: Dan, A., Lamersdorf, W. (eds.) *ICSOC 2006*. LNCS, vol. 4294, pp. 141–152. Springer, Heidelberg (2006)
5. Nguyen, X.T., Kowalczyk, R., Han, J.: Using Dynamic asynchronous aggregate search for quality guarantees of multiple Web services compositions. In: Dan, A., Lamersdorf, W. (eds.) *ICSOC 2006*. LNCS, vol. 4294, pp. 129–140. Springer, Heidelberg (2006)
6. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* 30(5) (2004)

7. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.J.: Modeling quality of service for workflows and web service processes. *Web Semantics Journal: Science, Services and Agents on the World Wide Web Journal* 1(3), 281–308 (2004)
8. Zeng, L., Lei, H., Chang, H.: Model-analysis for Business Event Processing. *IBM Systems journal* (2007)
9. Zeng, L., Lei, H.: A Semantic Publish/Subscribe System. In: *IEEE CEC, East* (2004)
10. Gillmann, M., Weikum, G., Wonner, W.: Workflow Management with Service Quality Guarantees. In: *SIGMOD* (2002)
11. Nahrstedt, K., Xu, D., Wichadakul, D., Li, B.: QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Comm. Magazine* 39(11) (2001)
12. Zeng, L., Lei, H., Jeng, J.-J., Chung, J.-Y., Benatallah, B.: Policy-Driven Exception-Management for Composite Web Services. In: *IEEE CEC* (2005)
13. Zeng, L., Jeng, J.-J., Kumaran, S., Kalagnanam, J.: Reliable Execution Planning and Exception Handling for Business Process. In: Benatallah, B., Shan, M.-C. (eds.) *TES 2003*. LNCS, vol. 2819, pp. 119–130. Springer, Heidelberg (2003)
14. Castellanos, M., Casati, F., Dayal, U., Shan, M.-C.: A Comprehensive and Automated Approach to Intelligent Business Process Execution Analysis. *Distributed and Parallel Databases* 16(3), 239–273 (2004)
15. Zeng, L., Lei, H., Chang, H.: Monitoring QoS for Web Services. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007*. LNCS, vol. 4749, pp. 132–144. Springer, Heidelberg (2007)
16. Brown, R.G.: *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall, Englewood Cliffs (1963)
17. Holt, C.C.: Forecasting seasonals and trends by exponentially weighted moving averages. In: *ONR Memorandum*, vol. 52. Carnegie Institute of Technology, Pittsburgh (1957); Available from the Engineering Library, University of Texas at Austin (1957)
18. Gardner Jr, E.S.: Exponential smoothing: The state of the art—Part II. *International Journal of Forecasting* 22(4), 637–666 (2006)
19. Box, G.E.P., Jenkins, G.M.: *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco (1970)

# Automatic Realization of SOA Deployment Patterns in Distributed Environments

William Arnold, Tamar Eilam, Michael Kalantar, Alexander V. Konstantinou,  
and Alexander A. Totok

IBM T.J. Watson Research Center, Hawthorne, NY, USA  
{barnold,eilamt,kalantar,avk,aatotok}@us.ibm.com

**Abstract.** Deployment patterns have been proposed as a mechanism to support the provisioning of SOA-based services. Deployment patterns represent the structure and constraints of composite solutions, including non-functional properties, such as performance, availability, and security, without binding to specific resource instances. In previous work [1], we have presented a formal mechanism for capturing such service deployment patterns using models. Our pattern models define abstract connectivity and configuration requirements which are then *realized* by an existing or planned infrastructure. Realization mapping is used to enforce policies, and is materialized at deployment time. In this paper we extend that work to address the problem of *automatic pattern realization* over a given infrastructure. We first formalize the problem and present three variations of increasing power and complexity. We then present a variation of a search-based graph isomorphism algorithm with extensions for our pattern model semantics. Next, we show that our worst-case exponential complexity algorithm performs well in practice, over a number of pattern and infrastructure combinations. We speculate that this is because deployment topologies represent heavily labeled and sparse graphs. We present a number of heuristics which we have compared experimentally, and have identified one which performs best across most scenarios. Our algorithm has been incorporated into a large deployment modeling platform, now part of the IBM Rational Software Architect (RSA) tool [2].

## 1 Introduction

From the perspective of an SOA deployer, the service layer specifying service hosting, connectivity, and binding can often be viewed as the tip of an iceberg. SOA services are typically implemented as components of distributed application platforms supported by large and complex middleware containers. These containers are often dependent on other remote middleware servers for messaging, database management, and authentication. The servers on which these containers execute must be monitored, secured, and audited and therefore have their own connectivity requirements. Server communication capabilities are constrained by the topology of the networks to which they are connected. The fact

that many of these communication paths are interdependent through layering [3], but often separately managed, presents a great challenge to SOA deployers [4].

Deployment patterns [5,6,1,7,8] have been proposed as an answer to the complexity of SOA deployment and the often subtle and difficult to quantify interactions and trade-offs between functional requirements, performance, security, and availability. There are several ways in which deployment patterns prove to be helpful. First, they simplify service deployment by codifying *best practices*. Second, they capture complex interdependent resource configurations that collectively achieve certain *non-functional* infrastructure properties, such as high availability, scalability, and security. These properties can then be used to satisfy non-functional requirements (NFRs) of the services being deployed. Third, deployment patterns capture intrinsic properties of composite services, while allowing them to be deployed in different environments, such as development, testing and production.

In our recent work [1] we presented a novel approach to formally capturing SOA deployment patterns. Our deployment patterns represent abstract deployment topologies specified at various levels of abstraction. They capture the structure and constraints of a composite solution, without bindings to specific resources, and without specifying provisioning actions. Deployment patterns are instantiated by deployers through *realization* of patterns in deployment topologies, representing existing or desired state of the infrastructure. Using our deployment platform, we enable non-expert users to safely compose and iteratively refine deployment patterns, resulting in fully specified topologies with bindings to specific resources. The resulting desired state topology can be validated as satisfying the functional service requirements, while maintaining the non-functional properties of the pattern.

In this paper, we turn our attention to the problem of *automatic pattern realization*, which is a function that produces a *valid realization* of a pattern in a given target infrastructure. There are several use cases for pattern auto-realization. First, it can greatly simplify the work of service deployers: if a composite solution's hosting requirements are represented as a deployment pattern, the task of solution deployment can potentially be reduced to running a simple automatic pattern realization wizard performing automatic resource selection. Second, it can be used for *compliance* purposes to verify that a given infrastructure conforms to certain organizational constraints, policies and best practices, which are captured in deployment patterns. Third, it can enable deployment *impact analysis*: the ability to plan deployment changes by playing "what-if" scenarios and assessing the planned changes. In addition, automatic pattern realization that supports *infrastructure reconfiguration* can be used for infrastructure provisioning, to drive reconfiguration of the infrastructure to conform to the pattern structure and constraints.

Our approach to the automatic pattern realization problem is based on the observation that it is reducible (with some variations) to the subgraph isomorphism problem [9]. We view deployment topologies as *labeled graphs* (augmented with constraints), and infrastructure reconfiguration actions as *graph edit*

operations [10]. We propose to use (modified) *graph matching* algorithms [9] for the pattern realization that does not allow changes to the target infrastructure, and *error-correcting graph matching* algorithms [10] for the situation where infrastructure reconfiguration is necessary. We present an algorithm for automatic pattern realization for the case, where no changes are allowed to the target infrastructure. We implement the algorithm in a large deployment modeling platform [1] and analyze its performance on a set of real-life deployment scenarios. We show that the algorithm's performance depends on the heuristic used to navigate the problem's *search tree*. We analyze performance of several such heuristics and identify one which provides good performance across a range of patterns and infrastructures. Our results show the practicality of using the algorithm, although its worst case complexity is exponential. We speculate that this is because deployment topologies represent heavily labeled and sparse graphs.

The paper is structured as follows. In Section 2, we describe our deployment pattern modeling platform. In Section 3, we formalize the problem of automatic pattern realization and introduce a number of variations. In Section 4, we present the algorithm for automatic pattern realization in a common case, where no changes are allowed to the target infrastructure. We analyze the behavior of the algorithm in Section 5. Finally, we discuss related work in Section 6, and conclude in Section 7.

## 2 Deployment Modeling and Validation

Our model-driven SOA deployment platform [1] supports the construction of deployment models at different levels of abstraction, ranging from *abstract models* (also termed, *patterns*) to *concrete*. Abstract models capture only intrinsic properties of a reusable deployment solution, they partially specify the configuration of resources, focusing on key parameters and structures, while leaving others to be determined at deployment time. Concrete models include detailed software stacks and configurations; valid and complete deployment models can be consumed by provisioning automation technologies (such as, Tivoli Provisioning Manager (TPM) [11]) to drive automated provisioning [12]. The platform, modeling concepts, and principles were introduced in [1], and are partially repeated here for completeness. To simplify the presentation, some definitions are simplified, where this does not affect the algorithm spirit and principles. For a complete description of the platform, modeling language, and analytic capabilities see [1].

The core model captures common aspects of deployment and configuration syntax, structure and semantics. Core types are extended to capture domain-specific information. Domain objects and links are contained in a *Topology* which is used to represent a composite solution. Figure 1 is an example of a deployment model (*Topology*). The *Unit* core type represents a unit of deployment, which may correspond to a hardware resource (x86 Server), a software product (Windows XP OS, WebSphere Application Server), or a software configuration node (J2EE Datasource). Subtypes of *Unit* group domain-specific configuration

attributes [1]. A *Unit* may represent an *installed* resource, or a resource *to be installed*. The *state* attribute on *Units* is an ordered pair which represents the *initial* and *desired* state of a *Unit*. In the example, *Windows2000Unit* and *Was6Unit* represent installed resources, and all other units are to be installed. Note that deployment topology examples used throughout the paper tend to either be abstract (not referring to specific resource types) or present low levels of SOA infrastructures: they were chosen to be just simple enough to demonstrate the idea of deployment pattern realization. We are continuing the work on modeling system and software configurations containing higher levels of SOA stacks and supporting more sophisticated considerations, such as messaging, security, high availability, etc.

Resource dependencies and requirements on other resources are represented by *Requirement* objects, contained in units. A *Requirement* is a tuple  $(t_r, t_l)$ , where  $t_r$  represents that *type of required resource*, and  $t_l$  represents the *type of relationship (link)*. We define three *relationship types*: a many-to-one *hosting* relationship; a one-to-one *dependency* relationship, and a many-to-many *membership* relationship. Relationships between *Units* are represented by a *Link* object, which is a quadruple  $(u, r, v, t)$ , where  $r$  is a *Requirement* contained in *Unit*  $u$ ,  $t$  is the *type* of the link (i.e., *hosting*, *dependency*, or *member*), and the link connects the *Requirement*  $r$  and a target *Unit*  $v$ .

*Local constraints* can be defined and contained in *Units*, where the context of evaluation is the containing unit, or in *Requirements*, where the context of evaluation is the associated relationship's target unit. An example of such a local constraint is “`version ≥ 1.4`” contained in a requirement of type *hosting* (contained in the *EARUnit*) for a target resource of type *J2EEContainer* (Figure 1), which restrict the version of the hosting application server. Note that *WAS6Unit* is a subtype of *AppServerUnit*, thus the constraint is satisfied in the example topology. A special *Membership constraint*, contained in a *Requirement* of type *membership*, can further restrict the multiplicity of a membership relationship instance.

*Structural constraints*, operating on pairs of *Units* can also be defined via a *Constraint Link*. Two common constraint links are: *Collocation* and *Deferred Hosting*. A *Collocation* constraint restricts the valid hosting of two units. It is associated with a *target type* property which determines the type of the host on which the two units' hosting stacks must converge (anti-collocation can be defined similarly). *Deferred Hosting* is a constraint that the source unit be eventually (indirectly) hosted on the target of the link.

**Semantics and Validation.** Topologies are evaluated against a set of *core platform validation rules*, including a core set of local constraints ( $=, \leq, \geq, <, >, \dots$ ), the two structural constraints described above, and link validity rules (multiplicity and endpoint types). The platform is extensible with (domain specific) validation rules and constraints (see [1]). Validation rules and constraints produce validation statuses of three types: *satisfied*, *undefined*, and *violated*.

<sup>1</sup> Some of the modeling concepts from [1] are simplified in this paper to focus on the algorithmic aspect. In particular, we collapsed the *Capability* concept into *Unit*.

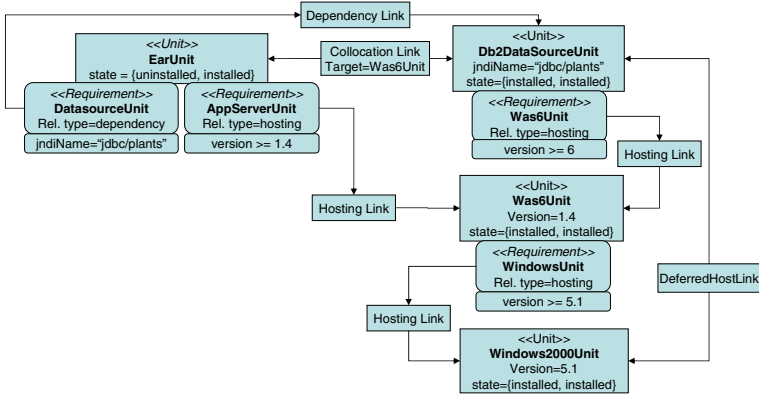


Fig. 1. An example deployment model

Undefined statuses will be generated in incomplete topologies, where there is not enough information to validate a rule or a constraint. For example, a Requirement that is not yet linked with a target unit, or a Deferred Hosting constraint connecting two units with an incomplete software stack, will both produce an undefined status. Violated statuses will be generated when units are improperly connected or when an attribute on a unit has an invalid value. For example, a link  $l=(u, r, v, t)$  that connects requirement  $r=(t_r, t_l)$  with a target unit  $v$ , where  $t \neq t_l$  or  $type(v)$  is not a subtype of  $t_r$  will produce a violated status. Note that a violated status can not be turned into a valid or undefined status just by adding units and links to the topology. A topology is weakly valid if its validation does not produce any violated statuses; valid if only satisfied statuses are produced; and invalid otherwise. Note that complete topologies (where all attributes are set, and all requirements are associated with links), can either be valid or invalid. Complete and valid topologies can be consumed by a provisioning technology for automated deployment. The topology in Figure 1 is valid and complete.

**Virtual Units and Realization Links.** To allow modeling at different levels of abstraction, we introduced the concept of a Virtual Unit. A virtual unit is one which does not directly represent an existing or an installable resource, but instead should be realized by another (concrete) unit. Typically, virtual units will be of an abstract type, will include attributes with unspecified values, and will be associated with constraints. A Realization Link connects a virtual unit with a concrete unit that forms its realization.

**Topology Realization Semantics.** It is convenient to separate the validation of the realization mapping from the core validation operating on topologies with only concrete units. The rules for locally validating realization of a virtual unit by a concrete unit are formally defined in two stages as follows. For any two requirements  $r=(t_r, t_l)$  and  $r'=(t'_r, t'_l)$ ,  $match(r, r')$  iff  $t_l = t'_l$  and  $supertype(t_r, t'_r)$ . We say that a concrete unit  $u_2$  is a locally valid realization of a virtual unit  $u_1$

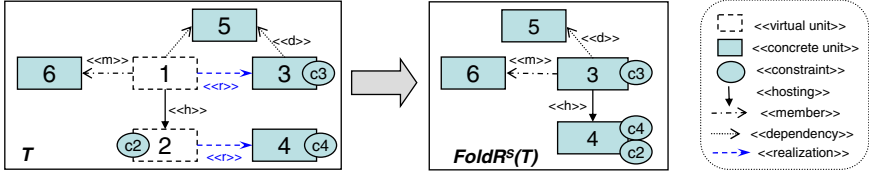


Fig. 2. A realization folding example

(denoted  $validR(u_1, u_2)$ ) iff (1)  $supertype(type(u_1), type(u_2))$ , (2) for every attribute  $a \in attributes(type(u_1))$ ,  $isSet(u_1, a) \rightarrow value(u_1, a) = value(u_2, a)$ , (3) for every constraint  $c \in constraints(u_1)$ ,  $c(u_2) = satisfied$ , (4) there exists a unique mapping of the set of requirements  $r_1, r_2, \dots, r_n$  on unit  $u_1$  to a set of distinct requirements  $r'_{i_1}, r'_{i_2}, \dots, r'_{i_n}$  on unit  $u_2$  (denoted  $map_{req}^R$ , w.r.t. unit realization mapping  $R$ ), such that  $match(r_k, r'_{i_k})$ <sup>2</sup> and (5) for every constraint  $c \in constraints(r_k)$ ,  $c(r'_{i_k}) \in \{satisfied, undefined\}$  (inclusion of the *undefined* validation status accounts for the fact that  $r'_{i_k}$ 's target may not be defined yet).

Given a topology with virtual units and realization links, it is not enough to locally check the validity of individual realization links. For example, consider a virtual unit  $u$  hosted on a non virtual unit  $v$ . A valid local realization of  $u$  can map it to a non virtual unit  $u'$  hosted on a non virtual unit  $v'$ , where  $v' \neq v$ , thus violating the hosting relationship many-to-one multiplicity constraint. To fully validate realization mappings, we define the *strict folded topology*  $FoldR^S(T)$  of a given topology  $T$ , where, intuitively, we collapse all realized virtual units, relationships and constraints (into the respective concrete units), and remove unrealized virtual units (see [1] for a formal definition, and Figure 2 for an example).

We say that a topology  $T$  forms a *weakly valid topology realization* iff (1) every virtual unit is realized by at most one unit, (2) each realization link in  $T$  is locally valid, and (3)  $FoldR^S(T)$  is *weakly valid*. A topology  $T$  forms a *valid topology realization* iff  $FoldR^S(T)$  in the definition above is *valid*. A topology realization is *complete* when all its virtual units are realized. Note that  $FoldR^S(T)$ , for a complete valid (or weakly valid) topology realization  $T$ , is a more succinct (normalized) representation of the deployment information available in  $T$ . In particular, new links, local, and structural constraints may be introduced on concrete and pairs of concrete units. Condition (3) prevents realizations that violate link multiplicity constraints, and checks validity of local and structural constraints defined on virtual units, against their corresponding realizing units.  $FoldR^S(T)$  can substitute  $T$  in an iterative, pattern based, deployment planning process. We will use  $FoldR^S(T)$  later, when defining the auto-realization algorithms. An example of topology realization is illustrated in Figure 2. Note that the folding introduces a new membership link between units 3 and 6. This

<sup>2</sup> To simplify the presentation of some of the definitions and algorithms, we assume that the injective mapping  $map_{req}^R$  is unique. The definitions and algorithms for pattern realization can be easily generalized to deal with multiple such mappings.



does not violate condition 3 in the definition above since the multiplicity of membership links is many-to-many. Assuming that local constraints are all satisfied on the respective concrete units, the figure represents a valid and complete topology realization. Finally, we say that a complete topology realization  $T$  is *strict* if  $T$  further satisfies the following property: for every non-constraint link  $l=(u, r, v, t) \in T$ , where both units  $u$  and  $v$  are virtual, there exists a corresponding link  $l'=(u', r', v', t') \in T$ , such that  $u' = R(u)$ ,  $v' = R(v)$ ,  $r' = \text{map}_{req}^R(r)$ , and  $t = t'$ , where  $R$  is the realization mapping function. This property requires effectively that no non-constraint links are “inherited” by a pair of concrete units from the pair of virtual units that they realize, during the topology folding process. Topology realization in Figure 2 does not satisfy this property, because the hosting link between units 1 and 2 does not have corresponding link between units 3 and 4.

### 3 Automatic Pattern Realization

In this section, we formally introduce the pattern realization problem, and several variants of it. We discuss the motivation for the problem and its variants based on real life use cases.

Let  $P$  be a *pattern topology*, where all units are virtual, and let  $T$  be a *target topology*, where all units are concrete. Let  $R$  be a set of realization links between units in  $P$  and  $T$ . We denote by  $P \cup T \cup R$  the topology formed by taking the union of  $P$ ,  $T$ , and  $R$  (following the common definition of graph unions). The *Pattern Realization (PR) problem* is formally defined as follows.

**The PR Problem.** Given a pattern topology  $P$  and a target topology  $T$ , produce  $PR = T \cup P \cup R$ , where  $R$  is a set of realization links between units in  $P$  and  $T$ , and  $PR$  forms a *complete weakly valid topology realization* (as defined in Section 2). ■

Note that the definition above can be easily generalized, where  $P$  and/or  $T$  contain both virtual and concrete units, and realization links. The *PR* problem definition is useful to describe the process of incremental elaboration and refinement for pattern-based deployment, where a basic step maps an abstract (pattern) topology into a concrete (but maybe incomplete) topology, in which some of the units represent resources that are installed or “to be installed”. Each such mapping step produces a normalized folded concrete topology that conforms to the input pattern, and can be used as input to the next elaboration and refinement step, eventually leading to a valid and complete deployment topology that satisfies multiple required patterns and that can serve to drive automated provisioning.

There are several variants of the problem that we find useful in real life scenarios. Specifically, consider a situation where the target topology  $T$  represents an existing computing infrastructure (where all units are *installed*). A very common case is where no changes are allowed to the infrastructure. In such a case, the automatic pattern realization process may be used for (1) resource selection, where a pattern topology is used to select an environment with certain

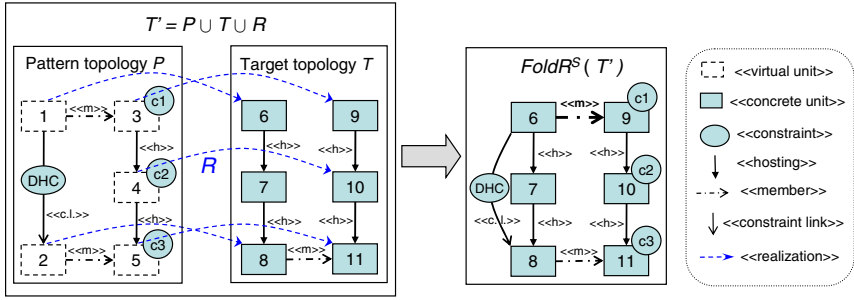


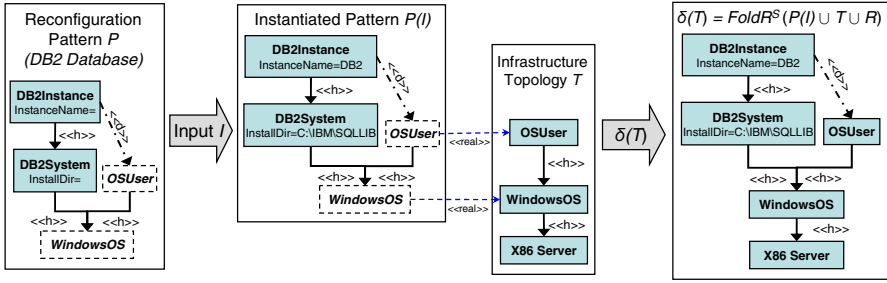
Fig. 3. Example of pattern realization

characteristics that can serve as, e.g., a hosting environment for downstream composite service deployment, or for (2) IT compliance verification, where a computing infrastructure is continuously validated against a set of organizational constraints, policies and best practices (represented as patterns). We term this variant of the problem *Strict Pattern Realization in Infrastructure Topology (PRIT)*. The *Strict PRIT Problem* is formally defined below.

**The Strict PRIT Problem.** Given a pattern topology  $P$ , and a target topology  $T$ , produce  $PRIT^S = P \cup T \cup R$ , where  $R$  is a set of realization links between units in  $P$  and  $T$ , and  $PRIT^S$  forms a *strict valid and complete topology realization* (as defined in Section 2).

Note that there are two differences between the *PR* and the *Strict PRIT* problems. In *Strict PRIT* the goal is to produce a *valid* topology realization (not just weakly valid). Further, the “strictness” property mandates that effectively no new non-constraint links are added between concrete units in the folded topology  $FoldR^S(PRIT^S)$ . Moreover, the only objects “inherited” by the folded topology  $FoldR^S(PRIT^S)$  from the pattern topology  $P$  are constraints and constraint links). Indeed, both these requirements stem from the fact that no changes are allowed in the infrastructure; new links such as a new membership link between two units, imply necessary infrastructure reconfiguration. For example, Figure 3 represents realization of pattern topology  $P$  (which contains among other constraints, a Deferred Hosting constraint) in target topology  $T$ . Topology  $T' = P \cup T \cup R$  is a complete valid topology realization (and thus a valid output of the *PR* problem), but not a strict topology realization, because it introduces a new member relationship between units 6 and 9 (in the corresponding folded topology  $FoldR^S(T')$ ). Thus it is not a valid output of the *Strict PRIT* problem.

**Pattern realization with infrastructure reconfiguration.** Consider a case where a deployment pattern does not have a *Strict PRIT* realization in the target infrastructure. However, if changes are allowed to the infrastructure, we may be able to *modify* it in a way that the pattern’s realization in the infrastructure is possible. In such a case, the automatic pattern realization process may be used to



**Fig. 4.** Example of valid application of a reconfiguration action to an infrastructure topology

drive *reconfiguration* of the infrastructure toward a state where it conforms to the pattern structure and constraints. We term this variant of the problem *Relaxed Pattern Realization in Infrastructure Topology (Relaxed PRIT)*. Note that ability to reconfigure the infrastructure to enable pattern realization depends on the set of allowed *infrastructure reconfiguration actions*. We first present a formal model for reconfiguration actions.

Infrastructure reconfiguration actions may include adding new hardware resources (e.g., adding new servers from the pool of available machines), installing new products from the product catalog (e.g., installing DB2 Database), configuring new managed middleware resources (e.g., creating a new *DB2 JDBC Provider* on a WebSphere Application Server), or configuring a new relationship between existing resources (e.g., adding existing JBoss application server to existing cluster). Note, that we only consider actions that *add* new resources or resource relationships. Reconfiguration actions that *remove* resources (or relationships) are beyond the scope of this paper.

There are two challenges in modeling reconfiguration actions. First, presence of certain resource types in a certain configuration state may be a precondition for the execution of a reconfiguration action. Second, the result of these actions may imply addition of a connected set of units and multiple relationships between new units and existing ones. To model both action preconditions and its effects we introduce the notion of a *Parameterized Reconfiguration Pattern*.

*Parameterized Reconfiguration Pattern* is a topology (consisting of virtual and concrete units) where some of the attributes on concrete units may be designated as *installation parameters*. Concrete units correspond to the affect of the action; namely, resources that will be added (provisioned) as a result of the action execution. Virtual units correspond to the pre-conditions for executing the action; for example, resources on which the new resources will be installed or created. *Installation parameters* correspond to values that are received from the user at installation time (e.g., name to be used for a newly created Database).

An *Instantiated Reconfiguration Pattern P(I)* is a pattern *P* and a set of input parameters *I*, where values from *I* are assigned to all *installation parameters* (note that default values may be used while still allowing downstream changes at the actual time of provisioning). A *bounded* re-configuration action w.r.t. a

topology  $T$  is a triple  $\delta = (P, I, R)$ , where  $P$  is the pattern topology associated with the action,  $I$  is the set of input installation parameters, and  $R$  is a realization function from  $P$  to  $T$ . A bounded action is *valid* iff  $R$  is a valid and complete realization. The *effect* of applying a valid bounded action  $\delta = (P, I, R)$  in a target topology  $T$  is the folded topology  $\delta(T) = FoldR^S(P(I) \cup T \cup R)$ . Note that we do not require that the topology realization be strict; this allows us to add links between existing concrete units in the target topology that represent required resource re-configuration. Figure 4 shows an example of a valid application of a reconfiguration action, corresponding to the installation of a DB2 Database on a Windows operation system. In this rather simplified example of product installation, pattern topology  $P$  consists of two concrete units ( $DB2System$  and  $DB2Instance$ ), which will be added to the topology. The set of input installation parameters  $I$  consists of  $DB2 InstanceName$  and  $InstallDir$ . The database is installed on a Windows operating system (represented as a virtual unit) and requires a ‘db2admin’ OS user, which is modeled as a virtual unit upon which the  $DB2Instance$  unit depends.

The input to the *Relaxed PRIT Problem* is a pattern  $P$ , a target topology  $T$ , and a set of allowable reconfiguration patterns  $\Gamma = \{P_i | i = 1 \dots M\}$ . To find a realization of  $P$ , it may be necessary to first apply a *sequence* of reconfiguration actions. Let  $\Delta = \{\delta_1, \delta_2 \dots \delta_n\}$  be a sequence of bounded reconfiguration actions, such that  $\delta_1 = (P_{i_1}, I_1, R_1)$  is a valid bounded reconfiguration action applied to  $T$ , and for  $k = 2, \dots, n$ ,  $\delta_k = (P_{i_k}, I_k, R_k)$  is a valid bounded reconfiguration action applied to topology  $\delta_{k-1}(\dots \delta_2(\delta_1(T)))$ . We also define  $\Delta(T) = \delta_n(\dots \delta_2(\delta_1(T)))$ .

**The Relaxed PRIT Problem.** Given a pattern topology  $P$ , a target topology  $T$ , and a set of allowable reconfiguration patterns  $\Gamma$ , produce a valid reconfiguration sequence  $\Delta$  and a topology  $R\text{-PRIT}^S = P \cup \Delta(T) \cup R$ , such that  $R$  is a valid strict topology realization of  $P$  in  $\Delta(T)$ . ■

To conclude this section, we discuss the relationship between the three variants of the pattern realization problem defined in this section (*PR*, *Strict PRIT*, and, *Relaxed PRIT*). *Strict PRIT* is a more strict version of the *PR* problem, i.e., every (*input, solution*) pair of the *Strict PRIT* problem is also an (*input, solution*) pair of the *PR* problem. *Strict PRIT* is also a more strict version of the *Relaxed PRIT* problem. The *Relaxed PRIT* problem is parameterized by the set  $\Gamma$  of allowable reconfiguration actions. If  $\Gamma = \emptyset$  then *Relaxed PRIT* becomes equivalent to the *Strict PRIT* problem. If the only allowed reconfiguration action in the *Relaxed PRIT* problem is link creation between existing units, then it is equivalent to a modification of the *PR* problem that requires valid (not only weakly valid) topology realization.

## 4 Algorithms for Automatic Pattern Realization

Our approach to the problem of automatic pattern realization is based on the observation that it is reducible (with some variations) to the subgraph isomorphism problem [9], where realization links represent the isomorphism mapping. We view deployment topologies as *labeled graphs* (augmented with local and

**Table 1.** Algorithm for the Strict PRIT problem

```

[01] FindStrictPRIT(Topology  $P$ , Topology  $T$ , Map  $R$ ) {
[02]   if ( $P_R = \text{units}(P)$ ) return  $R$ ; // all pattern units realized
[03]   select unit  $u \in \text{units}(P) - P_R$ ; // select unrealized pattern unit (heuristic)
[04]   for (unit  $v \in \text{units}(T)$ ) { // iterate over all target units
[05]     if (not(validR( $u, v$ ))) continue to next target unit; // locally invalid realization
[06]     for (non-constraint link  $l = (u, r, u', t) \in \text{links}(P)$ )
[07]       if ( $(u' \in P_R) \wedge (l' = (v, \text{map}_{req}^R(r), R(u'), t) \notin \text{links}(T))$ )
[08]         continue to next target unit; // target unit not linked to previous choices
[09]     for (non-constraint link  $l = (u', r, u, t) \in \text{links}(P)$ )
[10]       if ( $((u' \in P_R) \wedge (l' = (R(u'), \text{map}_{req}^R(r), v, t) \notin \text{links}(T)))$ )
[11]         continue to next target unit; // target unit not linked to previous choices
[12]     let  $T' = (T - \{v\}) \cup \{\text{FoldR}^S(u \rightarrow v)\}$ 
[13]        $\cup \{\text{constraint links } cl = (u', u) \vee cl = (u, u') \in \text{links}(P) \mid u' \in P_R\}$ ;
[14]     if ( $T'$  is a valid topology) { // no constraints violated
[15]       let  $R' = \text{FindStrictPRIT}(P, T', R \cup (u \rightarrow v))$ ;
[16]       if ( $R' \neq \emptyset$ ) return  $R'$ ; // all remaining realizations found.
[17]     }
[18]   }
[19]   return  $\emptyset$ ; // no realizations found (backtrack)
[20] }
```

structural constraints), and infrastructure reconfiguration actions as *graph edit operations* [10]. We propose to use modified *graph matching* algorithms for the Strict PRIT problem and *error-correcting graph matching* algorithms [10] for the Relaxed PRIT problem.

There are certain properties of automatic pattern realization that differentiate it from classic graph isomorphism. First, two virtual units can potentially be realized by (mapped to) a single concrete unit. Second, local and structural constraints defined in the pattern topology need to be satisfied in the target topology. In this section we present the algorithm for the *Strict PRIT* problem. Algorithms and heuristics for the *Relaxed PRIT* problem are deferred for future publications.

**Algorithm for the Strict PRIT Problem.** Our algorithm for *Strict PRIT* is based on the classic depth-first backtracking search subgraph isomorphism algorithm [10], modified to account for the properties of SOA pattern realization not exhibited in classic graph isomorphism. In Section 5, we analyze the complexity of the algorithm, and present its performance evaluation.

We use the following notations in describing our algorithm.  $R$  is the realization mapping of units in pattern  $P$  to units in target  $T$ , computed so far.  $P_R$  denotes the set of units in  $P$  that are already mapped by the realization. When a virtual unit  $u$  is realized by a concrete unit  $v$ ,  $\text{FoldR}^S(u \rightarrow v)$  denotes the unit  $v$  with constraints defined on the unit  $u$  folded onto it (this includes constraints defined in requirements contained by  $u$  folded on to the corresponding requirements in  $v$ ). Table 1 presents the algorithm in pseudo code. Function *FindStrictPRIT* should be invoked with arguments {pattern topology  $P$ , target topology  $T$ ,  $\emptyset$ }. At each step of the recursive iteration the following is true for the arguments of the recursive call:  $P$  is the pattern topology (unmodified),  $T$  is the folded topology  $\text{FoldR}^S(P \cup T \cup R)$ . The output of the function is a valid strict realization mapping of units in  $P$  to units in  $T$ , if one exists, or the empty set otherwise. Note that this algorithm finds the first such realization mapping, if it exists. In

our deployment modeling prototype, we have also implemented a variation of the algorithm that finds *all* such realization mappings.

The algorithm works by mapping (virtual) units in pattern topology  $P$  to (concrete) units in target topology  $T$ , one by one. At each iteration, the algorithm selects the next unmapped unit  $u$  in pattern  $P$  (line 3) and attempts to find a realization mapping for it against all the target units (unit  $v$  in topology  $T$ , line 4). For this, it checks that the realization is locally valid (line 5), and that for every non-constraint link in the pattern with  $u$  as source or target, and the other endpoint already realized, there exists a corresponding link in the target topology (lines 6–11). Note, that potentially several units in pattern topology  $P$  can be mapped to the same unit in target topology  $T$ . If such a unit  $v$  is found, the algorithm folds constraint links and constraints (those that now can be folded due to the newly computed unit realization  $u \rightarrow v$ ) from pattern  $P$  onto target topology  $T$  (lines 12–13). The modified topology  $T'$  is then validated to check if the realization satisfies structural constraints (line 14). Incremental validation techniques can be applied to enforce only the constraints affected. If all constraints are *valid* or *undefined*, the unit pair ( $u \rightarrow v$ ) is added to the realization mapping  $R$ , and the algorithm recurses to map the next unit in the pattern topology (line 15). If no further unit mapping is possible, the algorithm backtracks (line 19).

The incremental folding of constraints by the above algorithm can interact negatively with constraints that reason about the presence of other constraints. The above algorithm assumes that such “meta” constraints will return *unknown* when the constraint they are reasoning about is missing and could be added later in the process of realization. The performance of the algorithm will also be affected by the computational complexity of topology validation in step 14. Incremental validation techniques can be used to statically analyze declarative constraints, and monitor the access patterns of opaque constraints to reduce the number of operations per topology validation.

## 5 Performance Evaluation

We have implemented the *FindStrictPRIT* algorithm (Table 1) as an integral part of our deployment modeling platform [1], which has been recently released as an integral part of the IBM Rational Software Architect (RSA) tool [2]. Our implementation includes two versions of the algorithm: the *FindFirst* version finds the first realization mapping of the pattern and stops, while the *FindAll* version finds *all possible* realization mappings. *FindStrictPRIT* is a search algorithm that has, in the worst case, exponential complexity. However, we have identified that its performance greatly depends on the *heuristics* used to navigate the problem search tree. To evaluate the algorithm’s performance, we used different combinations of heuristics to execute searches for pattern realization for a fixed set of patterns in target topologies of varying sizes.

**Pattern Topologies.** We experimented with a variety of patterns including some that are small and abstract and some that are more complex and detailed.

**Table 2.** Summary of Pattern Topologies

Pattern	Description (number of units in the pattern)
<i>Pattern 1</i>	Standalone WebSphere Application Server collocated with a DB2 Instance on an x86 server; expressed using a detailed hosting stack. (12)
<i>Pattern 2</i>	<i>Pattern 1</i> expressed using <i>Collocation</i> and <i>Deferred Hosting</i> constraints instead of a detailed stack. (8)
<i>Pattern 3</i>	WebSphere cluster containing two application server members. (3)
<i>Pattern 4</i>	<i>Pattern 3</i> with an additional <i>Anti-Collocation</i> constraint between the servers. (5)
<i>Pattern 5</i>	<i>Pattern 3</i> with additional relationships to WebSphere nodes, a nodegroup and a cell. (17)

For example, one pattern for a two-server cluster contains three units while a more detailed version contains 17 units. In addition, some patterns include structural constraints while others do not. A summary of patterns used in the experiments is given in Table 2.

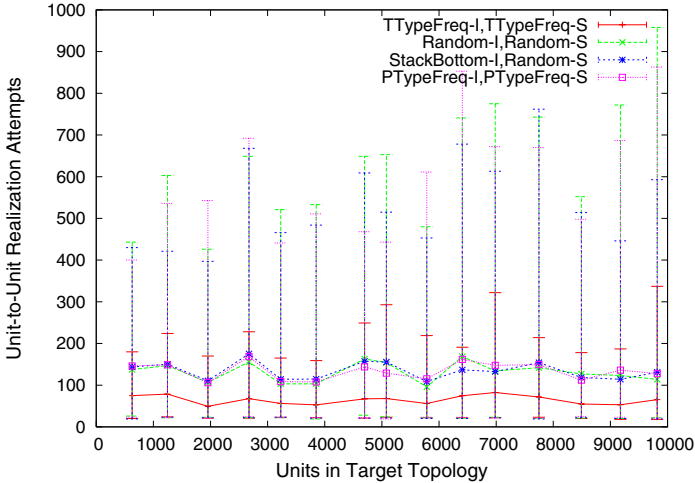
**Target Topologies.** For our experiments, we generated models of target topologies of varying sizes, designed to approximate a typical data center in which there are a large number of servers, each hosting a particular software. Each target topology contains a variable number of servers hosting one of the following software stacks: standalone WebSphere 6.0, DB2 8.2 Server, standalone WebSphere 6.0 with DB2 8.2 Server, standalone WebSphere 6.0 with DB2 8.2 Client, Apache, and WebSphere 6.0 ND. The WebSphere 6.0 ND stack contains multiple application servers hosted on different nodes grouped in multiple clusters. Each generated topology contains a fixed proportion of each type of stack. This allows us to linearly scale the size of the target topology using a simple scaling factor. Generation of target topologies ensures at least one match of a pattern in each target topology, thus allowing us to separate the impact of failed realizations.

**Search Heuristics.** We applied heuristics to the following key decisions in the search: (1) selection of the *initial unit* in the pattern to realize and (2) selection of the *next unit* in the pattern to realize. We assume that given a unit from the pattern topology, the set of units in the target topology of the same type can be identified in constant time. We believe this to be a reasonable assumption for infrastructures whose resources are maintained in indexed databases. In addition to indexing by type, resources are also typically indexed by one or more key attributes. The heuristics we tested are defined in Table 3. Those appended with “-I” apply to the selection of the initial unit in pattern to realize, while those appended with “-S” apply to the selection of subsequent units to realize. Given a pattern unit and a set of equivalent candidate target units, we select the target unit in a random order.

**Experiment Results.** For each pattern we executed the *FindFirst* and *FindAll* versions of *FindStrictPRIT* against target topologies ranging in size from 124 units (11 servers) units to 2764 units (210 servers). In each case we measured the number of *unit-to-unit realization attempts* as a function of the size of the target topology. We report realization attempts as a platform independent measure of the algorithm’s running time. Each test was repeated 100 times and the

**Table 3.** Summary of Heuristics

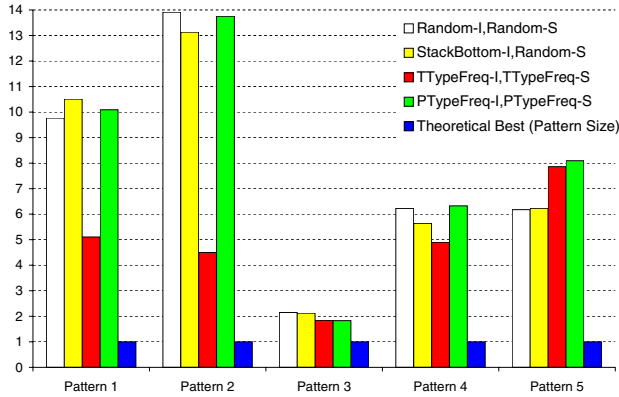
Heuristic	Definition
<i>Fixed-I (-S)</i>	The unit is selected in a fixed order from among all units (all units linked to the currently realized units). This heuristic allows us to minimize variance in tests of other heuristics.
<i>Random-I (-S)</i>	The unit is selected randomly from among all units (all units linked to the currently realized units).
<i>StackBottom-I</i>	The unit is selected randomly from among those at the base of hosting stacks. This heuristic tries to take advantage of the units related by hosting links.
<i>TTypeFreq-I (-S)</i>	The unit is selected randomly from among units (units linked to the currently realized units) with the type matching the least common type in the <i>target</i> topology. This heuristic tries to minimize the search space by starting with the least common target.
<i>PTypeFreq-I (-S)</i>	The unit is selected randomly from among units (units linked to the currently realized units) with the type matching the least common type in the <i>pattern</i> topology. This heuristic attempts to eliminate search combinatorics caused by common elements in the pattern by trying to select units with unique types.

**Fig. 5.** Number of unit-to-unit realization attempts vs. target topology size for different heuristics, applied to *Pattern 1*

results averaged. Figure 5 shows a representative result for the *FindFirst* algorithm version, used with four representative heuristic combinations, executed on *Pattern 1*. In addition to showing averages, min and max values are shown for each tested combination of heuristics.

Figure 5 reflects what we observed in most tests: that for our combinations of patterns and target topologies, *FindFirst* completes in approximately constant time independent of target topology size. We believe that this is a consequence of the target topology indexing which, given a pattern unit type, efficiently identifies a set of potential matches in the target topology. In such a set, many of the returned candidate units are in complete pattern realizations. The first complete realization can therefore be constructed in a constant number of steps. We also observed that heuristic *TTypeFreq-I* was most effective at reducing the number of unit-to-unit realizations and the variance. For those patterns where





**Fig. 6.** Number of unit-to-unit realization attempts of different heuristics, relative to the *Theoretical Best* one, applied to different patterns

the heuristic did not provide much advantage, the frequency of the pattern unit types in the target topology was approximately the same for all units, minimizing the advantage of the heuristic.

To measure the quality of *TTypeFreq-I* as a heuristic, we compared average number of unit-to-unit realizations for each heuristic (averaged over all target topologies of different sizes) to the theoretical minimum number of unit-to-unit realizations (which is equal to the number of units in the pattern, given an oracle that always makes the right choice). Figure 6 shows the relative number of unit-to-unit realizations: a measure of 1 is the theoretical minimum number of realizations. The figure shows that of the heuristics, *TTypeFreq-I* is usually better than the other heuristics.

Due to a lack of space we do not present our results for the *FindAll* version of the algorithm. They showed that the number of unit-to-unit realizations was directly proportional to the expected number of complete pattern realizations for each of the patterns in the target topologies. Further, the *TTypeFreq-I* heuristic was, again, most effective.

## 6 Related Work

The idea of using *patterns (templates)* to capture important properties of a reusable service solution and to drive its deployment and configuration has been recently explored in the SOA research literature [5,6,1]. In [5], patterns describing conditions needed for the deployment of a service, were used for network level service deployment in the domain of active networks. In [6], templates were used to capture parameterized provisioning workflows, where pattern selection is identified by mapping from a Service Level Agreement (SLA). In our work [1], a pattern captures the structure and constraints of a composite solution, without bindings to specific resources, and without specifying provisioning actions.

Instead, the pattern can be used in the *pattern realization* process, which drives resource selection and necessary reconfiguration of the target infrastructure, creating a detailed infrastructure reconfiguration plan, if necessary. Such plan can then be consumed by other tools such as [13,12] for provisioning.

We use (modified) graph matching algorithms [9] for pattern realization. Graph matching has been the focus of intensive research for several decades [14,15,16,17,18,19,20,10]. One of its drawbacks is computational complexity. It is known that both subgraph isomorphism and error-correcting subgraph isomorphism problems are NP-complete [9]. The most common approach for graph matching is to directly construct graph isomorphism in a procedural manner, using *depth-first backtracking search* [14]. Several variations of this algorithm have been proposed. Some employ additional checks such as forward checking in Ullman's algorithm [15] or lookahead procedures for backtracking [18]. Others employ different heuristics for navigating the search tree to improve algorithm performance in the specific area of its application [21,20]. Our approach belongs to the latest category.

Although general graph matching algorithms are exponential, polynomial algorithms have been proposed by imposing certain restrictions on the graphs. For example, graphs with bounded *valence* can be matched in polynomial time [19]. This algorithm, however, is not applicable to the pattern realization problem, because general deployment topologies have unbounded valence (e.g., a server cluster may contain arbitrary number of servers). Moreover, this algorithm has poor performance in practice due to a large constant overhead. Unlike *precise* algorithms for graph matching that are guaranteed to find a match if one exists, *approximate* algorithms do not always find the solution but require only polynomial time [22]. Applying approximate algorithms to pattern realization may be an area of future research.

## 7 Conclusions and Future Work

In previous work [1], we have shown how complex SOA deployment patterns can be effectively expressed in a formal object-relationship based modeling language. We further showed how such pattern models can be efficiently validated through a folding transformation into the target objects by which they are realized. In this paper we have shown that in practice, this realization mapping can also be efficiently computed. We defined three variations of the automatic realization problem, and detailed the algorithm and performance of the Strict PRIT problem. We then presented experimental results of its behavior, identifying the *TTypeFreq-I* heuristic as the most effective. The algorithm and heuristic have been incorporated into our model-driven deployment platform, which has been released as a part of the IBM Rational Software Architect (RSA) tool [2]. In addition to resource selection, automatic pattern realization is being used to assist in operation modeling [23]. We are in the process of extending our implementation to support *Relaxed PRIT*. In future SOA deployment patterns research we plan on investigating interactive pattern realization, reverse pattern discovery, pattern composition, and pattern maintenance.

## Acknowledgements

The authors would like to thank Daniel Berg, Harm Sluiman, Andrew Trossman, Michael Elder, John Pershing, and Edward Snible, for providing valuable feedback, contributing ideas, and helping to shape our vision.

## References

1. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A., Totok, A.: Pattern based SOA deployment. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 1–12. Springer, Heidelberg (2007)
2. IBM: Rational Software Architect for WebSphere Software (RSA) V7.5 (September 2008)
3. Mehra, P.: Global deployment of data centers. *IEEE Internet Computing* 6(5) (September 2002)
4. Brown, A.B., Keller, A., Hellerstein, J.: A model of configuration complexity and its applications to a change management system. In: *Integrated Management* (2005)
5. Bossardt, M., Mühlemann, A., Zürcher, R., Plattner, B.: Pattern based service deployment for active networks. In: ANTA (2003)
6. Ludwig, H., Gimpel, H., Dan, A., Kearney, B.: Template based automated service provisioning supporting the agreement driven service life-cycle. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 283–295. Springer, Heidelberg (2005)
7. Redlin, C., Carlson-Neumann, K.: WebSphere Process Server and WebSphere Enterprise Service Bus deployment patterns. Technical report, IBM (November 2006)
8. IBM: WebSphere Process Server (WPS) V6.1 (2007)
9. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, New York (1979)
10. Messmer, B.T.: *Efficient Graph Matching Algorithms*. PhD thesis, University of Bern, Switzerland (November 1995)
11. IBM: Tivoli Provisioning Manager, TPM (2006)
12. El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., Konstantinou, A.: Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools. In: van Steen, M., Henning, M. (eds.) *Middleware 2006*. LNCS, vol. 4290, pp. 404–423. Springer, Heidelberg (2006)
13. Keller, A., Hellerstein, J., Wolf, J., Wu, K.L., Krishnan, V.: The CHAMPS system: change management with planning and scheduling. In: *NOMS*. IEEE Press, Los Alamitos (2004)
14. Corneil, D., Gottlieb, C.: An efficient algorithm for graph isomorphism. *Journal of the ACM* 17, 51–64 (1970)
15. Ullman, J.: An algorithm for subgraph isomorphism. *Journal of the ACM* 23(1), 31–42 (1976)
16. Gati, G.: Further annotated bibliography on the isomorphism disease. *Journal of Graph Theory*, 96–109 (1979)
17. Kitchen, L., Rosenfeld, A.: Discrete relaxation for matching relational structures. *IEEE Transactions on Systems, Man, and Cybernetics* 9(12), 869–874 (1979)
18. Haralick, R., Elliot, G.: Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14, 263–313 (1980)

19. Hoffman, C.: Group-Theoretic Algorithms and Graph Isomorphism. Springer, Heidelberg (1982)
20. Kim, W., Kak, A.: 3-D object recognition using bipartite matching embedded in discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 13, 224–251 (1991)
21. Tsai, W., Fu, K.: Error-correcting isomorphisms of attributed relational graphs for pattern recognition. *IEEE Trans. on Sys., Man, and Cybernetics* 9, 757–768 (1979)
22. De Jong, K., Spears, W.: Using genetic algorithms to solve NP-Complete problems. In: Schaffer, J.D. (ed.) *Genetic Algorithms*, pp. 124–132. Morgan Kaufmann, San Francisco (1989)
23. Abrams, S., Bloom, B., Keyser, P., Kimelman, D., Nelson, E., Neuberger, W., Roth, T., Simmonds, I., Tang, S., Vlissides, J.: Architectural thinking and modeling with the Architects' Workbench. *IBM Systems Journal* 45(3) (2006)

# The LLAMA Middleware Support for Accountable Service-Oriented Architecture

Mark Panahi, Kwei-Jay Lin, Yue Zhang, Soo-Ho Chang, Jing Zhang,  
and Leonardo Varela

Department of Electrical Engineering and Computer Science  
University of California, Irvine  
Irvine, CA 92697, USA

**Abstract.** Enterprises are turning to SOA for the flexible deployment of business processes. While current monitoring tools can detect service execution problems in an enterprise's servers and report such problems to human decision makers, they may not closely monitor the external services they use, diagnose the root cause of process problems, and automatically reconfigure the process to replace faulty services. This paper presents the LLAMA middleware support for service process monitoring, run-time management, and configuration. Instances of accountability agents are deployed to selectively monitor some services' performance. These agents in turn allow LLAMA's Accountability Authority (AA) to diagnose process problems and apply any necessary reconfiguration. The project also builds tools to simplify the setup and deployment of LLAMA components. Our experimental results indicate that using LLAMA contributes only a modest amount of system overhead, and that the diagnosis process is swift and sufficiently accurate.

## 1 Introduction

Service-oriented architecture (SOA) is a software architecture for composing loosely-coupled services into one cohesive business process (BP) [1,2,3]. Enterprises can benefit from SOA to dynamically create BP's by integrating both legacy and new services. Some BP's may involve external services supplied by third-party partners/providers and invoke them with a service-level agreement (SLA) on the QoS performance of those services.

Service providers must be held *accountable* for the failures in the services they offer. However, identifying the source of SLA violations or service failures in a BP can be difficult. BP's can be very complicated, with many execution branches and involving services from several different providers. Performances from services may have dependencies; an irregular performance from one service may cause another to fail unexpectedly. In order to investigate such problems, the behavior of services that belong to a BP must be continuously monitored, logged, aggregated, and analyzed [4]. Harnessing such vital information requires intelligent support from the service infrastructure.

Given the dynamic and complex nature of many BP's, a powerful yet efficient mechanism is required to identify the source of problems when a BP has not delivered the service performance as expected. Such an error identification mechanism must be *powerful* enough to handle the complex and causal nature of service interactions and *efficient* enough to cope with a large number of service nodes without imposing significant system overhead.

In this paper, we present a middleware framework, called the intelligent Accountability Middleware Architecture (LLAMA), for accountable service-oriented computing. LLAMA includes three main components. The *Accountability Service Bus* (ASB) transparently and selectively monitors and records services, hosts (e.g. CPU), and network behavior information for services running on it. LLAMA also deploys instances of monitoring *Agents* to observe and to inspect groups of services. Finally, an *Accountability Authority* (AA) is in charge of diagnosing performance problems in processes and applying desirable reconfiguration measures. The LLAMA project also provides useful tools to simplify the setup, selection and deployment of LLAMA components, as well as a friendly user interface to inspect the diagnosis status of all processes.

The contribution of our research is that we have implemented LLAMA as an intelligent SOA middleware framework to detect, diagnose and defuse QoS issues in BP's. Our project goes beyond current business activity monitoring (BAM) tools to add comprehensive error identification capabilities. Some of the unique features in LLAMA include:

1. efficient decisions to select only a subset of monitoring locations (called *Evidence Channels*) in a BP to reduce monitoring cost and overhead [4];
2. a powerful diagnosis engine (using the Bayesian network model [5]) which conducts probabilistic analysis to identify the most probable service failures [4];
3. optimal Agent selection coupled with service selection so as to build BP's with a low overall cost including services and diagnosis [6];
4. intelligent capabilities for Agents to inspect individual service logs to confirm if a service is indeed faulty, and if so, whether the cause was due to service logic, host, or network behavior.

Following the work reported in [4,6], this paper concentrates on the LLAMA architecture, its components, implementation and performance study. We measure the extent to which monitoring and inspection add overhead to system performance. We also investigate how the accuracy of diagnosis varies according to the amount of time and resources devoted to the diagnostic process. Our experimental results indicate that using LLAMA contributes only a very modest amount of monitoring overhead, and the diagnosis process is swift and sufficiently accurate given reasonable time allowed.

The paper is organized as follows. Sec. 2 reviews the concept of accountability and challenges for building accountable SOA. Sec. 3 presents the LLAMA accountable middleware architecture. We present a performance study of the LLAMA implementation in Sec. 4. Related work is compared in Sec. 5.

## 2 Background

### 2.1 Challenges of Accountability in SOA

As defined in the Merriam-Webster dictionary, *accountability* is “an obligation or willingness to accept responsibility or to account for one’s actions”. We apply the concept of accountability to SOA so that all services are regulated for effective QoS delivered to a BP, and all root causes of faulty service executions can be clearly identified, inspected, and defused to control damage. In our work, QoS parameters can include any attribute that can be measured and quantified, such as timeliness, throughput, reliability, and availability.

To achieve accountability in SOA, we have identified the following challenges introduced by SOA’s inborn characteristics: 1) An SOA accountability mechanism should be capable of dealing with the causal relationship existing in service interactions and find the root cause of a process problem. 2) Probabilistic and statistical theory should be used to model the uncertainty inherent in network-based workflows and server workloads. 3) The problem diagnosis mechanism needs to scale well in large-scale distributed SOA systems. 4) To prevent excessive overhead, the amount of service performance data collected should be as little as possible but still enough for a correct diagnosis to be made. 5) The trustworthiness of services needs to be continually evaluated based on accumulated service behavior data. All these important challenges have motivated the design and development of our accountability model, algorithms, and architecture. These issues have also been elaborated in [4].

There are different degrees to which SOA systems can achieve these goals. At the very core, SOA systems must provide efficient *monitoring* feedback and error checking facilities on deployed services. Next, accountability systems may provide an intelligent and powerful *diagnosis* engine to analyze monitored data from the execution history and process structure. Finally, a mature accountability system must provide a means to *reconfigure* faulty processes and to *prevent* future problems. Depending on the application needs, the diagnosis and reconfiguration mechanism can be performed either *offline* or *online*, with the online approach more preferred so as to promptly react to problems as they occur.

### 2.2 Current Monitoring Support in SOA

The enterprise service bus (ESB) is a common service integration and deployment technology for SOA [7] that has been extended to provide support for monitoring and logging. Tools such as business activity monitoring (BAM) provide both analysis and visualization of data collected by ESB for the various services deployed on it. Users of BAM tools can view the performance of business processes and be alerted when problems occur. For example, a BAM tool may answer the question: “Why is it taking an average of 10 minutes for a representative to answer a customer’s call?” The Cape Clear BAM [8] is a commercial solution that claims to “combine an ESB with BAM to provide greater flexibility and ease-of-deployment within complex heterogeneous environments.” Another example is the Saturn product working with the Mule ESB [9].

While current solutions to business process monitoring and problem diagnosis exist, they still have limitations. Current BAM tools usually report information via a dashboard or email alerts to human decision makers, who then manually initiate diagnosis and corrective actions. We foresee a future where enterprises will be integrating systems involving third-party, geographically dispersed services and components, and may wish to compose services dynamically and automatically. The LLAMA framework is therefore designed to perform automated diagnosis and reconfiguration based on either pre-specified or dynamically calculated service causal relationships.

### 2.3 Transparency and Service Provider Participation in an Accountability Framework

The LLAMA framework is designed to help enterprises with pinpointing responsible parties when a process execution has problems. To achieve this, *transparency* on the (internal) state of a service provider is critical to the success of the diagnosis. However, some service providers may not be willing to grant this transparency to external users. Service providers therefore need to consider the tradeoffs between transparency on one hand, and privacy and security on the other. Providers of “healthy” services actually will benefit by allowing its performance data to be reported in order to clear any fault responsibility. In other words, we believe transparency may be more valuable than privacy to most service providers.

In order to participate in the accountability framework, external service providers must install the LLAMA ASB (see Sec. 3.2) to produce an audit trail of their services, and to allow ASB to push performance data to Agents. In LLAMA, to give more flexibility, we design Agents to be accountability-related services that can be deployed by service clients, service providers, or any third party providers. The only requirement is that Agents can be selectively requested by the Accountability Authority to report data about services that belong to a specific process.

In the case that some service providers are not willing to release their performance data unconditionally, LLAMA allows service providers to choose among various levels of transparency. **Simple Auditing** requires the service provider to only install the ASB layer for their services. This activates data collection for such services. However, this data is stored within the server and only provided to the AA, via an assigned Agent, when diagnosis is required. **Dynamic Monitoring** requires the ASB installation to allow dynamic monitoring of services via Agents installed by the service provider. Agents deployed need only conform to a standard interface. Therefore, the advantage of this transparency level is that service providers may use their own Agents to participate in the diagnosis process. **Dynamic Third-party Monitoring** is similar to the previous level except that data is collected and processed by third-party “collateral” Agents.

For the rest of this paper, we assume all external services are running on an ASB instance and monitored by Agents deployed by LLAMA. This assumption simplifies our discussion. Support for other transparency levels will be addressed in the future.



### 3 The LLAMA Accountability Middleware Architecture

Using SOA, the choices of which service to invoke at a specific instance may be made continuously depending on current service performance, cost, and many other factors. For a highly dynamic environment, few frameworks in existence are built to automate the analysis and identification of business process problems, and perform reconfigurations. In this section, we highlight the various components and features of the LLAMA framework that make automated process monitoring, analysis, and detection possible.

#### 3.1 LLAMA Overview

Figure 1 shows an example of service system deployed on LLAMA. As discussed in Sec. 2.3, all service nodes are assumed to be deployed on the ASB. In addition to the service requester and services deployed, the Accountability Authority (AA) and Agents are the two main LLAMA components to be discussed in detail in the next section.

Agents are deployed on servers and selected by AA to monitor a BP. AA and Agents collaborate to perform run-time process monitoring, root-cause diagnosis, service network recovery, and service network optimization. Multiple Agents are selected by AA to address scalability requirements. Each Agent is put in charge of monitoring a subset of services (as depicted by the circles in Fig. 1) during the execution of the service process. When undesirable process outcomes are reported by monitors, Agents provide AA relevant service status information that serves as evidence for AA to diagnose the root causes of run-time process problems.

The **Accountability Console** provides a Web-based graphical user interface for LLAMA. The operations in the console are classified into three types of behaviors: 1) registering a business process and SLA requirements in AA, 2)

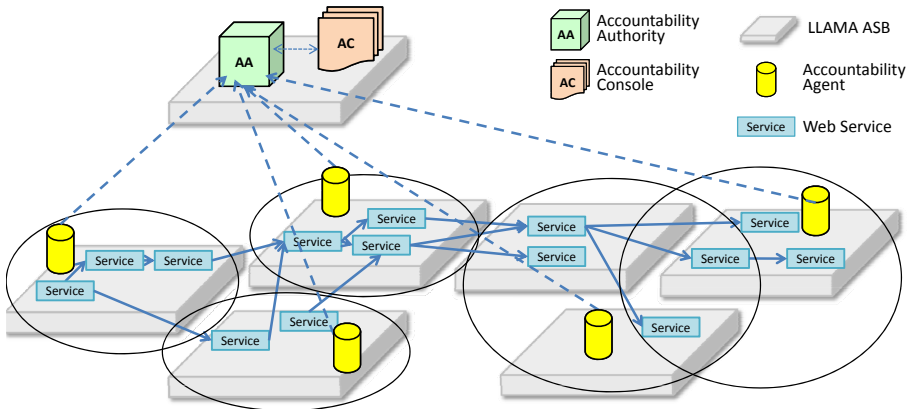


Fig. 1. An example layout of accountability framework

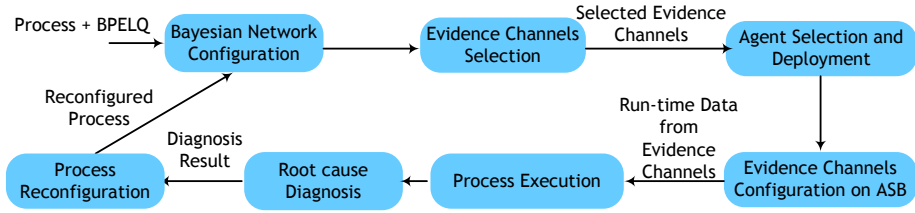


Fig. 2. Accountable SOA configuration and deployment flow

configuring AA parameters, and 3) reporting diagnosis results to users. The console provides the interface between human business process managers and the accountability framework through WSDM interfaces [10].

In addition to these components, LLAMA also has a QoS broker [11] (not in figure) which offers process composition and QoS-based service selection, in order to assist the service user in fulfilling user-defined end-to-end QoS requirements for a business process [12]. QoS broker composes BPEL processes and defines the QoS constraint for each individual service in a process. Furthermore, LLAMA deploys trust and reputation brokers for evaluating, aggregating, and managing the reputation of services. A service's reputation is considered as a QoS parameter that affects service network composition: services with a better reputation are more likely to be invoked by users.

Figure 2 shows the steps of the accountable SOA configuration and deployment. Given a user's service request with a user-specified end-to-end QoS requirement, the QoS Broker first composes the business process to be executed. The QoS Broker selects individual services that make up the service process, and identify the QoS constraint for each individual service in the process. These are specified in a special language designed in our project, called *BPELQ* (for "BPEL with QoS"), and sent to AA. In AA, the **Bayesian Network** for this process is configured based on the process graph, as well as both historical and expected service performance data [4]. AA then runs the **Evidence Channel Selection** algorithm using information from the Bayesian analysis to yield the best locations for collecting data about the process. The **Agents** that can best cover the services in this process are also selected and deployed by AA [6]. In addition, the hosts of the selected evidence channels are configured on the **ASBs**, ready to send monitored data at regular intervals to respective Agents.

Once the process starts to execute, performance data about services and the process will be collected by the Evidence Channels and delivered to Agents. If any abnormal situation is detected by an Agent, it will inform AA to trigger **Root Cause Diagnosis**. AA will produce the list of the most likely faulty services (with their respective probability readings), instruct the Agents to confirm the service status for fault identification, and initiate a **Process Reconfiguration** to resolve the problem. In this way, the framework can detect problems when they occur, find the root cause, and select a reconfiguration plan.

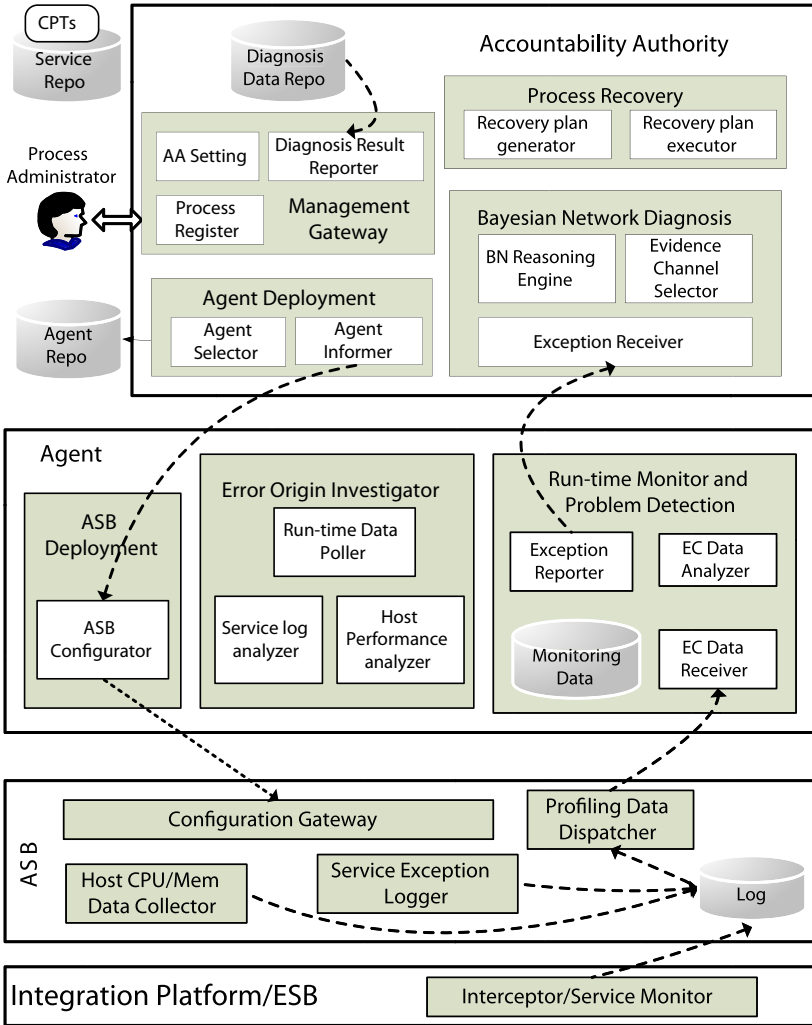


Fig. 3. LLAMA accountability architecture and components

### 3.2 LLAMA Components

In this section, we focus our discussion on the LLAMA accountability core (Fig. 3), comprised of the ASB, Agents, and the Accountability Authority.

**Accountability Authority (AA).** AA is the LLAMA component that performs intelligent management for the deployment, diagnosis, and reconfiguration of a service process. AA has the following responsibilities: (1) receive process

management requests from process administrators; (2) perform initial deployment and configuration of the accountability framework once the process is submitted (in BPELQ) for management; (3) perform root-cause diagnosis when exceptions are reported (sometimes concurrently) from Agents; (4) perform an automated process reconfiguration to recover process execution. The primary components of AA (in Fig. 3) are as follows.

- The **Management Gateway**, which serves as the portal for process administrators, allows them to submit processes and QoS constraints for accountability management. It also allows administrators to configure AA settings, such as the Evidence Channel selection strategy [4]. Moreover, it reports diagnosis results saved in the diagnosis data repository to administrators.
- At the deployment phase, the **Bayesian Diagnosis Engine** performs Evidence Channel selection algorithms [6] to decide the most cost-effective service data collection locations. At run-time, it performs Bayesian Network reasoning to identify the most suspicious services that may be the root-cause of problems. It also works with Agents to confirm the root cause and investigate possible error sources (e.g., service logic, host, network, etc.).
- The **Agent Selector** decides the set of Agents with the least cost to cover all services using good algorithms, such as the greedy set covering algorithm in [4].
- The **Agent Informer** notifies Agents about their selected Evidence Channels, i.e. the services each Agent is responsible for monitoring; the type of data to be reported; the frequency of data collection; and the criteria to decide if a service is normal or has an exception.
- The **Exception Receiver** receives filtered exception data from Agents, which usually triggers diagnosis procedures.
- The **Process Recovery** component generates process reconfiguration plans and executes them by notifying the ASB, via Agents, to reroute the process path using new services.

**Accountability Agents.** Agents are the intermediaries between where data is collected (i.e., ASB) and where it must be sent for analysis and diagnosis (i.e., AA). They are responsible for the following tasks: (1) configure evidence channels on ASB; (2) perform run-time data analysis based on the information pushed by ASB; (3) report exceptions to AA; and (4) initiate error origin investigation upon the request of AA. The components of Agents as illustrated in Figure 3 include:

- The **ASB Configurator** receives instructions from AA about which services are selected as Evidence Channels. Agents in turn contact ASB where a service is hosted and instructs ASB to send all relevant data about the service to the Agent.
- The **EC Data Receiver and Analyzer** receives pushed data from Evidence Channels located on ASB. It then applies exception criteria sent by AA on the data. Any data that meet the exception criteria are forwarded to Exception Reporter.

- The **Exception Reporter** promptly reports any abnormal situations detected by EC Data Analyzer to AA, which initiates the diagnosis process.
- The **Run-time Data Poller** assists AA to confirm root cause locations during the diagnosis process. It sends requests to ASB for data not in Evidence Channels, filters those data according to the exception criteria and sends them to AA.
- The **Error Origin Investigator** requests further information from ASB to determine if the source of an error is due to network, host, or the service itself once a problematic service is identified by AA.

**LLAMA ASB .** LLAMA ASB extends normal ESB capabilities by providing an API and framework for Agents to collect service performance data. Data can be pushed or pulled, and collected and sent at configurable intervals. LLAMA ASB can be installed on any existing ESB framework as long as it supports service request interception or other means of collecting service data. The following are the features of the LLAMA ASB (Fig. 3) that give it the unique monitoring capabilities.

- The **Configuration Gateway (CGW)** provides the service interface for configuring internal LLAMA ASB capabilities by Agents. CGW is used to configure service and host profiling data collection, including data collection and dispatch frequencies. Moreover, it allows the selection of Evidence Channels and dynamic routing configuration.
- **Profiling Interceptors** intercept service request/response messages and collect both timestamp and message output data. This data is stored in the ASB log.
- **Communication Interceptors** give the LLAMA framework its reconfiguration capabilities by providing the underlying mechanisms to route, forward, and redirect service requests [13] via the configuration of the routing table located on the ASB. It can accommodate both centralized and distributed process control models.
- The **Service Logger** collects software exceptions generated from service execution and stores them in the log. The exception log may be used for detailed diagnosis to understand the service failure.
- The **Host Data Collector** collects detailed statistics about the host's performance. ASB collects CPU, memory, and network data for each host on which it is deployed. The host data collector can be configured to collect data at intervals specified via CGW.
- The **Data Dispatcher** allows ASB to manage how data is pushed to Agents and can be configured to send data at intervals specified via CGW.

### 3.3 Error Origin Investigation

The LLAMA framework is unique in being able to diagnose process problems and heal the process if necessary by triggering a reconfiguration. It is important to pinpoint the exact cause of a process problem since it will influence the recovery

process. For example, if the network is at fault, a replacement service must be located outside of the original service's network. Similar action is taken if the host is at fault.

When an end-to-end QoS violation is detected during process execution, a list of likely causes is generated from the AA Bayesian network diagnosis engine. AA asks Agents to query data from ASB about a service, and such data is compared with the specified criteria to determine if an instance of data is an exception. All exceptions are gathered by Agents and sent to AA. AA then needs to determine the root source of the error. For this the Agent's error origin investigator is invoked by AA and the Agent requests further information from ASB on which the problematic service is deployed to determine the source of error. The Agent performs this investigation by looking up both system and service information, collected continuously by the ASB's Host Data Collector, based on the time at which the problematic service execution occurred. The Agent attempts to identify error causes by examining the sources listed below, based on some *preliminary* metrics.

- **Service exceptions:** Exceptions may be raised within a service's own implementation and may not only affect the correctness of a service's execution, but also could effect its performance. Such exceptions are recorded and stored within the ASB log. During error origin investigation, the log is checked to determine if any exceptions had been raised during the execution of the service.
- **CPU utilization:** High CPU utilization on a host will cause delays in all executing services. ASB by default collects CPU utilization data at 5 second intervals (using the `top` Linux command) and records them in a log. For analyzing CPU utilization, we refer to current research in load balancing literature to arrive at a problematic load threshold of 95% [14]. We compare this figure with the average CPU utilization over the time when the service was executed.
- **Memory consumption:** When memory consumption is high relative to available physical memory, paging may occur thus reducing system performance. ASB records memory consumption as a percentage of total available physical memory at regular intervals (using the `free` Linux command). If physical memory utilization exceeds 100% then we may assume that memory consumption is a contributing factor to the service's poor performance.
- **Network and infrastructure latency:** ASB records timestamps at both the beginning and the end of service executions. Using this data, Agents can calculate the time taken between the end of one service execution, and the beginning of the execution of the next service in the process. This time represents a combination of network time, as well as time spent in the communication infrastructure on each respective host. Agents can leverage network traffic data associated with each host (also collected by the ASB using the `ntop` Linux command) to further narrow the problem source to that of the network or host infrastructure. We apply techniques discussed in [15] to derive thresholds to which we compare network data when error origin investigation is invoked.

Although Agents may inspect all of the above data sources, sometimes it is still very difficult to distinguish between the various possible sources of service performance problems. In other words, the results from Agent investigations are by no means completely reliable. Such investigations are based on artificial thresholds and patterns that are empirically estimated and may vary from system to system. On the other hand, the investigation reliability may increasingly improve as more historical data is collected and correctly analyzed.

Once error investigation has been concluded by Agents, and the likely root cause of a process error is reported, AA then makes a reconfiguration decision. Various techniques for service selection and process reconfiguration [11], as well as ASB support for reconfiguration [13] has been presented in our earlier reports.

## 4 Empirical Results

A prototype of LLAMA has been implemented by building the ASB as an extension to the Mule ESB, and constructing the AA and Agents. We now study the overhead of the LLAMA prototype and the accuracy of its diagnosis process.

### 4.1 Example Scenario: Print and Mail

We use an example BP to test the effectiveness of the LLAMA framework. It is a BP for targeted mass mail advertising referred to as the Print and Mail process (Figure 4). For example, with such a BP a plumber can send fliers to all addresses in a certain zip code where the houses are older than 20 years.

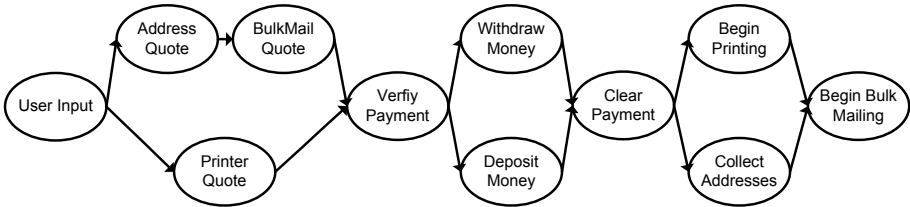


Fig. 4. The Print and Mail Business Process

The process begins with the user’s specifications, including address specifications and the image to provide to the printer. Next, various pricing quotes are gathered from print vendors, address providers, and bulk mailers. The process may retrieve quotes from several of these services in parallel to find the best one. Once the quotes have been settled, the form of payment is verified and the payments is made. In final stage, the bulk mail process can begin once the addresses are collected from the address vendor and the printing process has been completed.

## 4.2 Monitoring Overhead

Monitoring and problem diagnosis can potentially create overhead in any executing software system. In this section, we measure the overhead of the LLAMA framework for the Print and Mail BP. For this experiment, we deploy all services and two Agents on a single host. Therefore, data is being collected about all services. Moreover, six of the services are selected as Evidence Channels so that data from these services are being continuously pushed to the Agents. We set up three test scenarios: 1) execute the process 50 times with all profiling interceptors turned off, ensuring that there is no data collection overhead, 2) execute the process 50 times with all profiling interceptors activated ensuring that all data collection is taking place, and 3) the same as scenario 2, except that 6 Evidence Channels are configured and are pushing data at 5 second intervals to the Agents. Results from three separate runs are depicted in Table 1. These results indicate that there is no statistically significant difference between the first two scenarios and therefore monitoring overhead is negligible. Pushing data to local Agents, however, can create from 1% to 7% additional overhead.

**Table 1.** Monitoring Overhead (average of 50 runs for each test)

	Test 1 Average	Test 2 Average	Test 3 Average
No monitoring	29033 ms	27922 ms	28323 ms
Data collection at ASB	29053 ms	28515 ms	27862 ms
Data delivered to Agents	29354 ms	29199 ms	30437 ms

## 4.3 LLAMA Diagnosis Accuracy

In this section, we study how accurately LLAMA can find root causes when a BP is reported to have unacceptable performance according to specified requirements. The detailed design of the diagnosis engine and parameter setup have been reported in [4]. Here we test the actual performance in a deployed system.

To simulate performance problems, we inject 50-second delays into certain services in the process following the pre-defined reputation of those services. For example, if service A's reputation is 0.9, then there is 0.1 probability of delay injection. Since the response time of every service in our test is set to 10 seconds or less, we choose the acceptable response time for each service to be 15 seconds.

We have tested the Bayesian network reasoning as shown in Table 2. We find that if we inject delay to only one service, the Bayesian network reasoning can always find the correct root cause in 2 diagnosis rounds. If we inject delay in two services, the Bayesian engine can find out at least one root cause in 4 rounds.

We test two probability thresholds, 0.1 and 0.3, for initiating service inspection after the diagnosis report on the probability of service error. Obviously, a lower threshold will trigger more diagnosis rounds and cause a longer diagnosis duration. For single-cause cases, we find the threshold of 0.3 is good enough. But for multi-cause cases, the low threshold of 0.1 can help detect more problematic



**Table 2.** Diagnosis Performances with Thresholds of 0.1 and 0.3

<b>TH=0.1 cases</b>	1	2	3	4	5	6	7	8	9	10
BP duration	1m32s	1m31s	1m40s	1m40s	2m12s	1m58s	1m33s	2m6s	2m6s	2m7s
Diag. duration	1s	1s	2s	1s	2s	3s	1s	2s	2s	2s
#Diag round	7	4	4	2	6	6	6	4	8	7
Detection ratio	1/1	1/1	1/1	1/1	1/2	2/2	2/2	1/2	3/4	2/4
<b>TH=0.3 cases</b>	1	2	3	4	5	6	7	8	9	10
BP duration	1m33s	1m32s	1m45s	1m38s	2m5s	1m40s	1m36s	2m6s	2m8s	2m6s
Diag. duration	1s	1s	1s	1s	2s	2s	1s	2s	2s	2s
#Diag round	3	2	3	2	3	2	2	4	3	3
Detection ratio	1/1	1/1	1/1	1/1	1/2	1/2	1/2	1/2	1/4	1/4

services. The system may not catch all problematic services in multi-cause cases since if we inject errors in two services that are on the same path of a process, only the first one (executed first) may be reported as the root cause. We will continue to improve the AA’s diagnosis capability in our future work.

## 5 Related Work

Most enterprise software vendors have produced ESB products, including IBM, BEA, Sonic, Iona, Oracle, Cape Clear, etc. Several open source ESB projects have also been implemented, including Celtix, ServiceMix, Apache Synapse, Open ESB, etc. To our knowledge, none have included the sophisticated diagnosis capability like in our project. Our current implementation of LLAMA has been designed to run atop the Mule ESB [9], because it is a stable open-source project with highly extensible features. Mule allows customized interception and message transformation mechanisms to support profiling collection and process reconfiguration. However, LLAMA can run on any ESB which supports some means of collecting service invocation data.

In [16], Errandi et. al. present wsBus, a Web service middleware for reliable and fault-tolerant computing. To handle faults, wsBus performs run-time monitoring of functional and QoS aspects, as well as run-time adaptation based on the policies specified by users. However, the monitoring and adaptation components of wsBus are designed and executed on an individual service basis, whereas the LLAMA framework uses the accountability model to monitor the end-to-end quality of processes and adapt at the process level with an alternate path. In addition, the configuration gateway and rerouting facility in the LLAMA ESB are designed to be transparent and very efficient.

In [15], a performance metrics monitoring and analysis middleware named Tiresias is developed to collect application level (e.g., response time) and system-level performance metrics (e.g., CPU usage, memory usage, and network traffic). These performance data are used to make black-box failure-prediction through trend analysis. Our LLAMA framework differs from Tiresias in terms of data

dependency reasoning and analysis since our target applications are business processes.

Autonomic computing [17] aims to develop computing systems that can manage themselves given high-level objectives from administrators. The essence of autonomic computing systems is self-management, which comprises four aspects: *self-configuration*, *self-optimization*, *self-healing*, and *self-protection*. Accountability provides the solid foundation for the fully-scaled autonomic computing. It targets the self-healing and self-optimization aspects of autonomic computing – a system automatically detects, diagnoses, and repairs software and hardware problems. They share the common goal of identifying, tracing, and determining the root cause of failures in complex computing systems.

The goal of fault tolerance is to enable systems to continually operate properly. Therefore, the major technologies to achieve fault tolerance focus on recovery from failures. Those technologies mainly include replication, checkpointing and recovery lines [18]. Accountability complements dependable SOA systems by being able to identify and determine problematic services (e.g., violations of SLAs) and then defuse the problem.

## 6 Conclusion

As enterprises continue to construct a larger proportion of their business processes from outsourced third-party services, there will be an increased need to maintain the run-time knowledge of these services in order to accurately diagnose difficult-to-pinpoint problems when they occur. To address this emerging trend, we have developed the LLAMA middleware framework. It includes components to help monitor services on behalf of a process user, find the root cause of problems when they occur, and perform reconfigurations if necessary.

Due to space constraints, we present only the overall architecture, while omitting many algorithmic foundations and system details, which may be found in our earlier reports [4,6]. From the experiments reported in this paper, we have discovered that the LLAMA middleware creates little overhead, and provides a reasonably accurate root cause diagnosis. We believe that this is a promising approach toward implementing accountable SOA systems.

## References

1. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. *IEEE Computer* 40, 38–45 (2007)
2. Bichler, M., Lin, K.J.: Service-oriented computing. *IEEE Computer* 39(3), 99–101 (2006)
3. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. *IEEE Internet Computing* (January-February 2005)
4. Zhang, Y., Lin, K.J., Hsu, J.Y.: Accountability monitoring and reasoning in service-oriented architectures. *Journal of Service-Oriented Computing and Applications (SOCA)* 1(1) (2007)

5. Korb, K.B., Nicholson, A.E.: Bayesian Artificial Intelligence. Chapman & Hall/CRC, London (2004)
6. Zhang, Y., Panahi, M., Lin, K.J.: Service process composition with QoS and monitoring agent cost parameters. In: IEEE Joint Conf. on E-Commerce Technology (CEC 2008) and Enterprise Computing (EEE 2008) (July 2008)
7. Chappell, D.: Enterprise Service Bus. O'Reilly Media, Sebastopol (2004)
8. CapeClear: Capeclear bam (2008), <http://developer.capeclear.com/>
9. MuleSource: Mule 2.0 (2007), <http://mule.codehaus.org/display/MULE/Home>
10. Oasis: WSDM (2008), <http://www.oasis-open.org/>
11. Yu, T., Lin, K.J.: Service selection algorithms for composing complex services with end-to-end QoS constraints. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSSOC 2005. LNCS, vol. 3826, pp. 130–143. Springer, Heidelberg (2005)
12. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Transactions on the Web (May 2007)
13. Lin, K.J., Panahi, M., Zhang, Y.: The design of an intelligent accountability architecture. ICEBE, 157–164 (2007)
14. Balasubramanian, J., Schmidt, D.C., Dowdy, L., Othman, O.: Evaluating the performance of middleware load balancing strategies. In: EDOC 2004: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International, pp. 135–146. IEEE Computer Society, Washington (2004)
15. Williams, A.W., Pertet, S.M., Narasimhan, P.: Tiresias: Black-box failure prediction in distributed systems. In: The 15th International Workshop on Parallel and Distributed Real-time Systems. IEEE, Los Alamitos (2007)
16. Erradi, A., Maheshwari, P., Tosic, V.: Policy-driven middleware for self-adaptation of web services compositions. In: Middleware, pp. 62–80 (2006)
17. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
18. Tsai, W.T., Song, W., Paul, R., Cao, Z., Huang, H.: Services-oriented dynamic reconfiguration framework for dependable distributed computing. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004), pp. 554–559 (2004)

# *ubi*SOAP: A Service Oriented Middleware for Seamless Networking\*

Mauro Caporuscio, Pierre-Guillaume Raverdy,  
Hassine Moun gla, and Valerie Issarny

INRIA Paris-Rocquencourt  
Domaine de Voluceau - 78153 Le Chesnay, France  
{First.LastName}@inria.fr

**Abstract.** The computing and networking capacities of today's wireless portable devices allow for pervasive services, which are seamlessly networked. Indeed, wireless handheld devices now embed the necessary resources to act as both service clients and providers. However, the seamless networking of services remains challenged by the inherent mobility and resource constraints of devices, which make services a priori highly volatile. This paper discusses the design, implementation and experimentation of the *ubi*SOAP service-oriented middleware, which leverages wireless networking capacities to effectively enable the seamless networking of services. *ubi*SOAP specifically defines a layered communication middleware that underlies standard SOAP-based middleware, hence supporting legacy services while exploiting nowadays pervasive connectivity.

## 1 Introduction

With network connectivity being embedded in most computing devices, networking environments are now pervasive. As a result, any networked device may seamlessly consume but also provide software applications over the network. Service-Oriented Computing (SOC) then introduces natural design abstractions to deal with pervasive networking environments [2]. Indeed, networked software applications may conveniently be abstracted as autonomous loosely coupled services, which may be combined to accomplish complex tasks. In addition, the concrete instantiation of SOC paradigms provided by Web Services (WS) technologies by means of Web-based/XML-based open standards (e.g. WSDL, UDDI, HTTP, SOAP) may be exploited for concrete implementation of pervasive services. However, while Web services standards and implementations targeting wide-area domains are effective technologies, supporting Web service access in pervasive networking environments is still challenging. In such kind of networking environments, mobile applications, acting as both service consumers and providers, often run on scarce resource platforms such as personal digital assistants and mobile phones, which have limited CPU power, memory,

---

\* This work is part of the IST PLASTIC project and has been funded by the European Commission, FP6 contract number 026955, <http://www.ist-plastic.org/>

and battery life. Moreover, these devices are usually interconnected through one or more heterogeneous wireless links, which compared to wired networks are characterized by lower bandwidths, higher error rates, and frequent disconnections. The former issue has led to the introduction of lightweight middleware enabling base WS-oriented communication patterns among wireless portable devices (i.e., SOAP-based messaging and dynamic service discovery) [10,11]. The latter issue has further led to examine alternative SOAP transports [5]. However, a key feature of pervasive networking environments is the diversity of radio links available on portable devices, which may be exploited towards seamless connectivity. Specifically, as nodes get connected via multiple radio links, thorough scheduling and handover across those links allow enhancing overall connectivity and actually making it seamless [23,18,20]. This calls for making services network-agnostic [21], so that the underlying middleware takes care of scheduling exchanged messages over the embedded links in a way that best matches Quality of Service (QoS) requirements [4], and further ensures service continuity through vertical handover [9]. In this setting, a primary requirement for supporting service-oriented middleware is to provide a comprehensive networking abstraction that allows applications to be unaware of the actual underlying networks while still exploiting their diversities in terms of both functional and extra-functional properties.

This paper introduces the *ubi*SOAP communication middleware, which underlies SOAP-based middleware and strives to provide pervasive networking to services. Specifically, *ubi*SOAP defines a two-layer architecture composed of a *multi-radio networking* layer and a *WS-oriented communication* layer, which respectively provide network-agnostic connectivity and SOAP-based unicast and group communication in pervasive networking environments. The design rationale for *ubi*SOAP is further discussed in the next section, while Section 3 details its core constituents. Then, Sections 4 and 5 assess the proposed middleware, discussing respectively *ubi*SOAP usage for implementing an advanced middleware service (i.e., pervasive service discovery), and *ubi*SOAP performance based on experiment. Finally, Section 6 concludes with a summary of our contribution with respect to related work, and our perspectives for future work.

## 2 Design Rationale

With the drastic evolution of wireless technologies, software services can become truly pervasive, being not solely accessed but also hosted by wirelessly networked portable devices. As a result, legacy applications can become available anytime, anywhere, but also revisited to take full advantage of pervasive networking. Further, new application services may emerge, in particular based on the nomadic feature and ad hoc connectivity of wireless portable devices, as exemplified by emergency rescue scenarios, where mobile portable hosts serve sensing the environment and coordinating rescue actions. Still, enabling pervasive service provisioning on mobile hosts requires special care, as resources

are far more constrained than resource-rich Internet servers originally targeted by service oriented computing and its Web Service instantiation. Further, the mobility of wireless hosts requires special attention. Indeed, early solutions introduced towards nomadic computing targeted service hosts with which connectivity can eventually be restored, while this cannot be assumed in general when services are hosted by mobile devices that connect in an ad hoc way [24]. Overcoming resource constraints of wireless devices in the support of Web services has led to the introduction of custom SOAP engines, among which the open source iCSOAP engine from INRIA, which was developed as part of the WSAMI middleware<sup>1</sup> featuring Web services for ambient intelligence [10].

Regarding mobility issues, portable devices now embed multiple radio interfaces, which may be combined to bring seamless networking to mobile applications [21]. Specifically, embedded networking technologies differ from several respects, among which range, latency, bandwidth, energy consumption, financial cost, availability and so on. Therefore, the scheduling of communications over embedded interfaces according to application requirements can significantly increase the overall QoS. The collection of wireless technologies may in particular be considered as hierarchical *wireless overlay networks*, which are structured according to respective coverage [12]. Then, the interface used for communication may simply depend on the location of the target host. However, different network parameters must be taken into account in the scheduling of communications, so as to meet applications' QoS requirements, while optimizing the overall resource usage [18]. In particular, saving energy is critical for enhanced autonomy of hosts and thus requires selecting as far as possible the network interface that consumes the least energy among those eligible [20,4]. Further, vertical handover across heterogeneous wireless networks must be supported so as to maintain connectivity with nodes despite their mobility across networks [23], and thus bring seamless connectivity to/from mobile nodes.

Among one of its major goals, the *ubiSOAP* communication middleware aims at effectively using the diverse networking technologies in the handling of service-oriented communication, hence offering network-agnostic connectivity to/from nodes. As discussed above, this requires addressing a number of critical issues such as *network availability*, *user and application QoS requirements* and *vertical handover*. The latter issue is particularly important with respect to the *service continuity* requirement. In fact, when switching from a given network to one of a different type, the device is required to change its status according to the new environment it is entering. Indeed, changing the device's status affects also the status of all the devices that are currently interacting with it. Specifically, in an all-IP networking environment, the IP address meaning is twofold: end point identification (i.e., an IP address uniquely identifies a host in a given network) and location identification (i.e., the network in which the host is located). Hence, when a host changes its point of attachment (vertical handover between two networks), the IP address must be modified (i.e., the internal status) accordingly in order to route packets to the new network. Then, since the IP address is

---

<sup>1</sup> <http://www-rocq.inria.fr/arles/download/ozone/>

the base of any application-layer connection, all the ongoing connections break (i.e., the handover affects the status of the interacting parties). Furthermore, as devices can bind various networks at the same time, two interacting parties might communicate through multiple paths. Hence, choosing the best connection path to serve a given interaction is a key issue to deal with in pervasive networks, as this significantly affects the QoS at large (e.g., availability, performance with respect to both resource consumption and response time, security) [3].

Multi-radio connectivity further allows deploying bridges in the network, effectively realizing a multi-network service overlay [6]. Specifically, resource-rich nodes (e.g., laptops or even reachable stationary nodes) that embed multiple radio interfaces may act as bridges that route messages across heterogeneous networks. Such a feature is beneficial for pervasive services, enabling to overcome the resource limitation and mobility of nodes and contributing to achieve seamless service connectivity. Indeed, this allows energy-limited devices to use the least consuming interface while being able to reach all the devices of the overlay. Further, a networked service that changes physical network following host mobility may still be reachable in the overlay.

Another of our goals for *ubi*SOAP is to support legacy Web services and thus transparently bring the added value of today's pervasive networking environments to existing services. This has in particular led us to layer *ubi*SOAP as a specific transport for SOAP engines (e.g., Axis2, iCSOAP) and to leverage WS-addressing to integrate multi-radio, multi-network connectivity in SOAP headers. In this context, it is crucial to examine carefully the performance of SOAP transports. In particular, it has been shown that the performance of default SOAP over HTTP is poor in wireless environments, further leading to study alternative transports such as TCP and UDP [14,5]. While SOAP over UDP clearly offers the best response time, SOAP over TCP has the advantage of built-in reliability and is further suitable for applications with short requests. *ubi*SOAP thus realizes SOAP-over-TCP unicast messaging as a tradeoffs solution, while integrating SOAP over UDP is an area for future work. Still, another SOAP transport that is of much interest for pervasive networking environments is multicast group communication. Indeed, group-based interactions are central in a number of pervasive computing scenarios, due to the user-centric nature of pervasive computing and the innate group interaction skills of people [7,22]. *ubi*SOAP thus features a base SOAP transport for group communication.

### 3 *ubi*SOAP Middleware for Pervasive Services

Following the above discussion, the architecture of *ubi*SOAP layers the following constituents below SOAP-based middleware functionalities (see Fig. 1): (i) multi-radio networking provides network-agnostic connectivity (see § 3.1), (ii) multi-network routing implements a multi-network overlay (see § 3.2), and (iii) point-to-point and group transports leverage multi-radio, multi-network message routing, and further introduce communication primitives targeted at pervasive computing systems (see § 3.3).

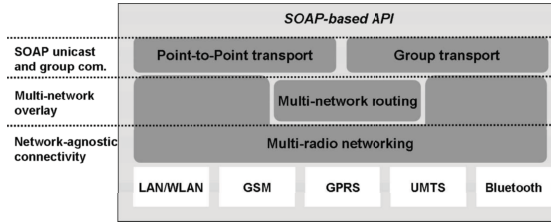


Fig. 1. ubiSOAP software architecture

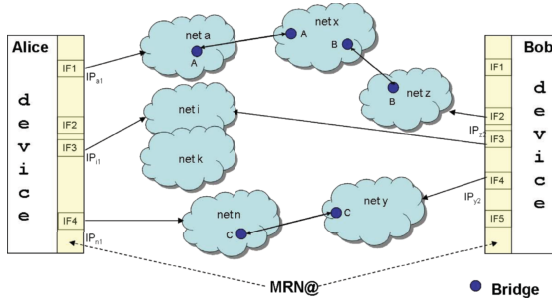
### 3.1 Network-Agnostic Service Connectivity

The *multi-radio networking layer* of ubiSOAP provides core functionalities to effectively manage multi-radio connectivity by providing: (i) a network-agnostic addressing scheme together with (ii) QoS-aware network link selection.

**Network-agnostic addressing.** Devices embedding multiple network interfaces may have multiple IP addresses, at least one for each active interface. Thus, in order to identify uniquely a given application in the network we associate to it a *Multi-Radio Network Address* (MRN@). The MRN@ of an application instance is specifically the application's Unique ID, which resolves into the actual set of IP addresses (precisely,  $network\_ID \oplus IP$  addresses) bound to the device (at a given time) that runs the given instance (see Fig. 2). Then, upper layers shall use MRN@ as part of their addressing scheme (e.g., through WS-addressing in the case of Web services), which replaces the traditional IP-based addressing scheme. MRN@s are automatically generated and managed by the multi-radio networking layer. Furthermore, the multi-radio networking layer allows for performing a lookup operation that, starting from an MRN@, returns the set of IP addresses actually bound to it. The basic operations provided by the multi-radio networking layer are as follows. First, *Registration* allows the application to register within the multi-radio networking layer and generates the MRN@ that uniquely identifies it. In particular, the user application provides as input an identifier (locally unique), which is used to generate the MRN@ to be returned. Then, *Lookup* allows user applications to retrieve the actual set of IP addresses related to a given MRN@. If the resolution of MRN@ is not cached or needs to be updated, a request is multicast to all the networks currently accessible and, if the device related to such MRN@ is reached, it will directly reply to the requester by supplying the actual set of IP addresses. Base unicast and multicast communication schemes are provided on top of MRN@ network-agnostic addressing: (i) *Synchronous unicast* is provided by means of a packet input/output stream that is used to read/write packets to be exchanged during the interaction between client and server applications; and (ii) *Asynchronous multicast* allows the user application to send multicast packets to all members of a given group.

**QoS-aware interface activation and network selection.** Next to MRN@ addressing, it is crucial to activate and select the best possible networks (among





**Fig. 2.** *ubiSOAP* multi-radio multi-network connectivity using MRN@ and bridging

those available) with respect to required QoS. *Interface activation* allows the user application to activate the best possible interfaces (among those available) with respect to the required QoS. In particular, the application submits its QoS requirement (specified as set of pairs  $\langle QoS_{attribute}, QoS_{value} \rangle$ ) to the multi-radio networking layer, which in turn compares it with the QoS of each available interface. In this case, since the interface is switched off, QoS refers to the theoretic values of a network interface declared by the manufacturer (e.g., GPRS maximum bitrate = 171.2Kb/s). If the interface satisfies the requirement posed by the application, within a given approximation expressed in percentage, it is activated. It is also possible to define priorities upon the various quantitative parameters, in order to specify if a given parameter is more important than the others. *Network selection* is performed during the establishment of the communication and takes into account the QoS attributes required by the client application that is initiating the connection, as well as the networks active on the server listening for incoming connections, as given by the server's MRN@. If the client and the server share only one network that satisfies the requirements, it is used to carry on the interaction. On the other hand, when the two parties share more than one network, the selection algorithm selects the one that best meets the required QoS.

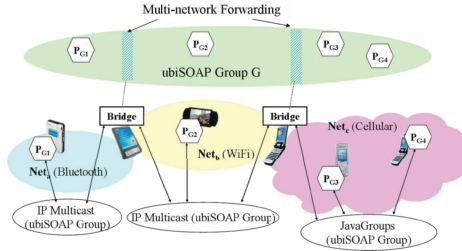
### 3.2 Multi-network Service Overlay

Thanks to the *ubiSOAP* multi-radio networking layer, communication among nodes exploits the various network links that the nodes have in common, further selecting the link that provides the required QoS. However, in some cases, it might also be desirable for nodes to be able to access services that are hosted in networks that the requesting node is not directly connected to (e.g., to provide continuity of service despite node mobility). For this purpose, *ubiSOAP* introduces an overlay network that bridges heterogeneous networks, thus enhancing overall service connectivity. Specifically, nodes that are connected to two (or more) different networks through their network interfaces can assume the role of bridge nodes. Bridge nodes quite literally “bridge” between two separate networks, relaying point-to-point and multicast *ubiSOAP* messages across those

networks. Still, we assume that nodes will not request services that would require the consecutive traversal of more than five wireless networks (see [8,15] for a detailed analysis on wireless communication) in order to access them.

**Multi-network point-to-point routing.** In *ubiSOAP* multi-network, multi-radio environments, the  $network\_ID \oplus IP$  address embedded in the MRN@ of a host contains, along with the network address of the service host, the network ID that uniquely identifies the network (e.g., a BSSID, MAC address of the Bluetooth master, etc.) that the host resides in under its given address. For instance, in Fig. 2, the of device Alice is connected to networks **a**, **i**, and **n**, through its various network interfaces. Clearly, the device can trivially access services hosted in these networks. However, in order to access services hosted in the distant networks **x**, **y**, and **z**, the device has to route its request through an appropriate bridge node (i.e., Bridges **A**, **B** and **C**, noting that each bridge node is displayed in each network it is part of). To achieve effective routing across bridges, we propose a straightforward approach based on the principle of Mobile Ad hoc NETwork (MANET) routing. In this approach, bridge nodes advertise their presence to the nodes in their corresponding networks and exchange routing information. For this purpose, bridge nodes run an instance of OLSR [11] among each other. Instead of concrete node addresses, however, bridges store as destinations the identifiers of the various present networks (i.e.,  $network\_ID$ ) and as next hop the bridge that needs to be contacted next to eventually reach the target network. Being a proactive routing protocol, this inter-bridge OLSR instance gives each bridge the required routing information to reach all connected networks. Whenever a non-bridge node wants to access a service outside one of the networks it is itself connected to, it may simply route the request to any bridge of choice that will then forward the request accordingly. As mentioned above, bridge nodes periodically advertise their presence to the nodes in their respective networks. As a further optimization, bridge nodes may include in their advertisements their OLSR routing tables so that non-bridge nodes may choose bridge nodes according to metrics such as network hops, etc.

**Multi-network multicast routing.** It is crucial to support both point-to-point and group interactions in the multi-radio, multi-network environment. In particular, multicasting is central to advanced middleware services like dynamic discovery [25]. We thus introduce multi-network multicast routing, building upon multicast facilities of the composed networks. The base principles of the *ubiSOAP* multi-network multicast routing are depicted in Fig. 3; multicast routing is such that within an IP network, the network's multicast facility (i.e., IP multicast or higher level group communication like Java Groups) is used for communication among group members. Then, multicast messages are forwarded by *ubiSOAP* bridges up to a fixed number of hops (i.e., 5 as discussed previously), while avoiding cycles and duplication. The *ubiSOAP* multi-network routing facility enables the definition of application-level groups. Specifically, application-level groups are individually managed at the *ubiSOAP* middleware layer while a single *ubiSOAP* multi-network group is managed in the network layer. The latter



**Fig. 3.** *ubiSOAP* multi-network multicast routing

is actually a composition of *ubiSOAP* multicast groups, one for each of the composed networks.

### 3.3 Custom SOAP Transports for Pervasive Services

In order to leverage the provided multi-radio, multi-network service connectivity, *ubiSOAP* introduces custom SOAP transports. Specifically: (i) the provided SOAP transport for point-to-point communication brings multi-radio multi-network routing to legacy SOAP messaging, while (ii) the SOAP transport for group communication enhances the SOAP API to meet the corresponding base requirement of pervasive networking environments [13,75,22,25].

**Point-to-point communication.** The *ubiSOAP* point-to-point transport is a connection-oriented transport for supporting communication between a client and a service. This transport interacts with the multi-radio networking layer to send and receive messages over the network based on the MRN@ that identifies the remote party. It also interacts with the SOAP engine or the client SOAP library to receive or dispatch SOAP messages locally. When sending a message, the *ubiSOAP* point-to-point transport must first evaluate if the destination is directly reachable (i.e., the MRN@ of the sender and of the destination share a common network). If true, the message is then sent directly to the destination. If not, the transport retrieves the MRN@ of a *ubiSOAP* bridge directly reachable, encapsulates as plain data the application’s SOAP message into a specific forwarding message, and sends this forwarding message to the bridge. This message is forwarded between bridges until it reaches the destination where the application’s SOAP message is extracted and dispatched. While the client blocks until the response is received, the forwarding message is routed between *ubiSOAP* bridges using connectionless communication. The response message may thus follow a different route. The first bridge returns the response to the client, or terminates the connection after a given timeout.

On the service side, the *ubiSOAP* point-to-point transport interacts with the SOAP engine to deliver the message to the appropriate service. For messages that have been routed by bridges, the destination must extract the actual SOAP message, and also set up properly the SOAP message parameters (i.e., URL

of the service, action to perform). This is achieved by storing (on the source side) and retrieving (on the destination side) the relevant information in the *ubiSOAP* header of the SOAP message. In particular, the set of IP addresses associated to an MRN@ is embedded in the header of request (for the client) and response (for the service) messages. This enables communicating devices in different networks to keep track of mobile nodes and maintain sessions (as long as a communication path exists). In some cases however (i.e., when both the client and the service simultaneously change the complete set of IP addresses associated to their MRN@), and no direct link exists), the session will close and the client will need to perform a service discovery (See Section X) to find the same service again and restart the communication.

**Group communication.** The *ubiSOAP* group transport is a connectionless transport for one-way communication between multiple peers in multi-network configurations. The *ubiSOAP* group transport component interacts with the multi-radio networking layer to send group messages based on an MRN@ identifying the group, and with the SOAP engine to deliver the group's messages to the registered services. As noted above, groups are identified with an MRN@. Multicast-based applications usually assume that all group members agree beforehand on a specific IP address for the group. We therefore also assume that all group members use the same MRN@ for the group. While services are not able to directly return a result to a client (one-way multicast), a service may send a message (one-way unicast) on the group directed at a specific peer (i.e., similar to the *sendTo* socket call). As group communication in the underlying multi-radio networking layer is multicast-based, it does not guarantee the ordering or the delivery of messages. While ordering may be easily achieved on the receiving side, the overhead to provide group reliability is deemed too costly due to the dynamics of pervasive networks. Also, while many mobile devices may run the same collaborative application, a user may only be interested in interacting with the ones at its location. Such scoping may be achieved by limiting the forwarding of group's messages or by adding forwarding constraints.

## 4 *ubiSOAP* in Action: Pervasive Service Discovery

*ubiSOAP* is being developed as part of a larger initiative on assisting the development of dependable services for pervasive networking environments, which is undertaken by the European IST PLASTIC project<sup>2</sup>. The PLASTIC project specifically investigates the development of the PLASTIC platform, decomposing into a development environment, service-oriented middleware and validation framework for the target pervasive services. *ubiSOAP* then defines the PLASTIC core middleware while advanced middleware services are being developed on top of it to address the requirements of pervasive networking (i.e., dynamic service discovery and composition, security and context management). Further, the PLASTIC platform is being assessed against actual case studies in the area

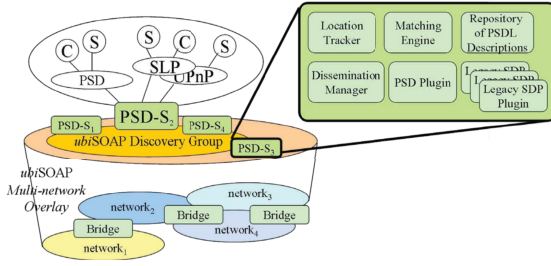
<sup>2</sup> <http://www.ist-plastic.org/>

of eHealth, eLearning, eBusiness and eVoting. This in particular allows us to extensively experiment with the *ubiSOAP* middleware. In this section, we focus on one use of *ubiSOAP* that is realizing dynamic discovery of pervasive services, which benefits from all the advanced features of *ubiSOAP*, i.e., group communication, and multi-radio, multi-network message routing.

Service discovery is an essential function of SOC as it enables the runtime association to the networked services. Three basic roles are identified for service discovery: (i) Service provider is the role assumed by a software entity offering a networked service; (ii) Service requester is the role of an entity seeking to consume a specific service; (iii) Service repository is the role of an entity maintaining information on available services and a way to access them. A service description formalism or language to describe the functional and non-functional properties (such as QoS, security or transactional aspects of networked services) complemented with a service discovery protocol enables service providers, requesters and repositories to interact with each other. Many academic and industry supported SDPs have already been proposed and leading SDPs in dynamic environments use a pull-based approach (SLP, WS-Discovery, Jini, SSDP), often supporting both the centralized and distributed modes of interaction: clients send requests to service providers (distributed pull-based mode) or to a third-party repository (centralized pull-based mode) in order to get a list of services compatible with the request attributes.

Building on the tremendous number of proposed service discovery protocols and accounting for the specifics of pervasive computing [25], we introduce a Pervasive Service Discovery (PSD) Service that provides dynamic, interoperable, context-aware service discovery. PSD is mainly a reengineering of the open source MUSDAC multi-protocol service discovery platform<sup>3</sup> [19] on top of *ubiSOAP* in order to support service discovery in multi-radio, multi-network environments. PSD uses a hierarchical approach for service discovery in multi-network environments (see Fig. 4). Indeed, a (logically) centralized repository (PSD-S) coordinates service discovery within an independent network, while PSD-Ss in different networks communicate together in a fully distributed way to disseminate service information. While in MUSDAC service discovery and access were tightly integrated, PSD-Ss are only concerned with service discovery, and rely on *ubiSOAP* group communication to disseminate service information across networks. Changes in the multi-network topology (e.g., broken propagation paths) are then taken care of transparently. PSD-Ss (see Fig. 4) provide an explicit API supported by the *PSD plugin* that enables clients (resp. providers) in a network to discover (resp. advertise) a service in the multi-network environment. It further enables clients and providers to benefit from advanced discovery features (e.g., context-awareness) by directly issuing requests or advertisements in the PSDL format. Specific *legacy SDP plugins* register with the active SDPs in the network, and translate requests and advertisements in legacy formats to PSDL (e.g., SLP and UPnP in Fig. 4), which are stored in the *PSD repository*. The *matching engine* then combines various matching algorithms [17] to support

<sup>3</sup> <http://www-rocq.inria.fr/arles/download/ubisec/>



**Fig. 4.** Pervasive Service Discovery

the various elements of the service description (for both requests and advertisements), and thus provides comprehensive interoperability between SDPs. Finally, the *dissemination manager* controls the dissemination of local requests and the compilation of the results returned by distant PSD-Ss, while the *location tracker* collaborates with lower-level services in the *ubiSOAP* middleware to maintain the physical address of mobile services discovered in the environment.

As described above, the PSD repository stores PSDL service descriptions that are either generated by legacy SDP plugins (e.g., UPnP2PSD plugin) or directly registered by service providers using the PSD plugin. We use a hierarchical service description format that actually combines a number of distinct documents specifying different facets of the service. The PSDL description acts primarily as a top-level container for additional files describing facets of the service. For example, a WSDL document may be used to describe the service interface while non-functional properties can be described using existing QoS and context models. The *ubiSOAP* grounding of host, that identifies the networks and IP address at which the service's host is network-reachable (i.e., MRN@ and mapping) is described in such separate document thus facilitating dynamic updates.

## 5 Experimentation

We have implemented a prototype of the *ubiSOAP* middleware using Java for both desktop (J2SE) and mobile (J2ME CDC) environments. As mentioned previously, the *ubiSOAP* prototype is being extensively experimented with as part of the PLASTIC project, and has been released under open source license<sup>4</sup>. To assess the efficiency of the *ubiSOAP* middleware, we evaluate the processing time to call a simple Echo Web service under various network configurations. We evaluate both the time required to call the Web service the first time, which includes the dynamic service creation/instantiation, and the time required to call the Web service after it is deployed. Tests are performed on a Windows XP PC with a 2.6GHz processor and 1 GB of memory for the desktop platform, and on

<sup>4</sup> *ubiSOAP* is composed of the Multi-radio Networking and B3GSOAP packages available at <http://www.ist-plastic.org/>

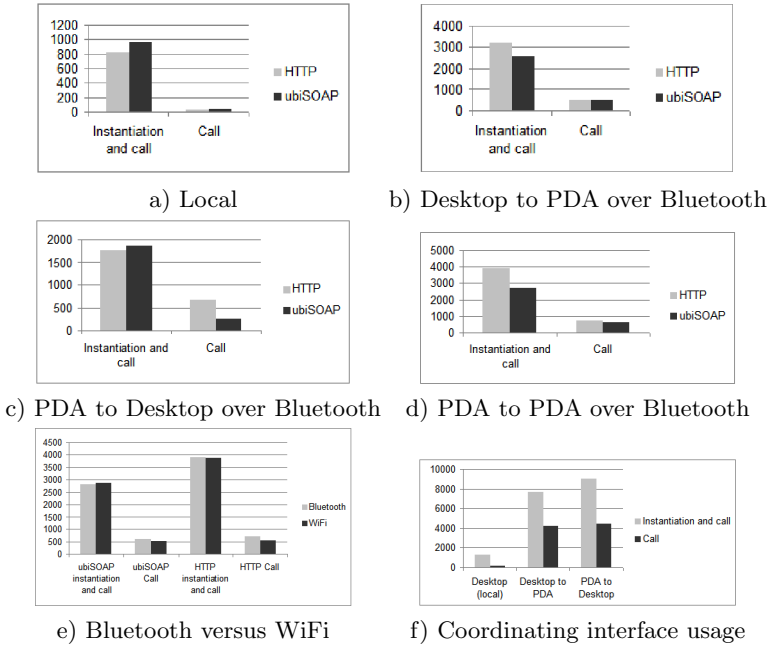


Fig. 5. ubiSOAP performance

a HP iPaq hw6910 (Intel PXA 270 at 416 MHz) and a HP iPaq 110 (PXA310 at 624 MHz) for the mobile platforms. We further use IBMs J9 JVM (J2ME CDC 1.1), and the open source iCSOAP lightweight SOAP engine<sup>5</sup>. Results presented are the average of 5 runs with 100 call each.

Figures 5.a) to d) assess the performance in ms, of ubiSOAP unicast transport versus HTTP in the following configurations: a) both client and service provider running on the desktop platform, b) client running on desktop and service provider on PDA, c) client running on PDA and service provider on desktop, and d) client and server running each on a PDA. Provided results subdivide into the response time of the initial call including the service instantiation (i.e., *Instantiation and call*), and the average response time of subsequent calls (i.e., *Call*). Configuration a) shows that the time taken for service instantiation far exceeds the one of calls. The response time of ubiSOAP is further slightly higher than the one of HTTP, which is due to the management of MRN@ whose processing gets noticeable due to the fact that the communication is local. Configuration b) then demonstrates that ubiSOAP outperforms HTTP by about 25% at instantiation time. This is explained by the fact that the processing overhead (header and session management) is much lighter in ubiSOAP, and the difference in processing is emphasized by the limited capacities of the devices (slow task switching, memory access, ...). As shown by Configuration c), in the case where the service provider runs on the desktop, HTTP performance

<sup>5</sup> Available at <http://www-rocq.inria.fr/arles/download/ozone/>

for instantiation is slightly better, because the difference in the management of session is negligible on the desktop, while *ubiSOAP* on the desktop adds the cost of MRN@ resolution through multicast. On the other hand, the *ubiSOAP* performance for calls is better, again due to lightweight session management on the client side and the fact that the MRN@ needs only to be resolved at the first call. Finally, Configuration d) aggregates the above results b) and c), showing the enhanced performance of *ubiSOAP* over HTTP when both client and service provider run on a PDA.

Figure 5.e) complements the above measures by showing the performance of wireless communications between PDAs using both WiFi and Bluetooth. The higher bandwidth of WiFi makes it obviously better positioned for handling communication compared to Bluetooth, even for small size messages. However, this takes into account performance only, while other criteria are of relevance like power consumption [4], as addressed by the QoS-aware network selection realized by the *ubiSOAP* multi-radio networking layer. Although the theoretical bandwidth for Bluetooth and WiFi is significantly different, the actual bandwidth available to applications depends on the hardware, drivers, and, in our case, the Java JVM ability to cope with the load. As demonstrated in the experiments, the actual bandwidth for Java applications on PDA is almost identical for Bluetooth and WiFi.

In general, the scheduling of communication over network interfaces shall account for the QoS requirements of networked applications. Such requirement and specifically conflicts between different applications should be taken into account when activating/desactivating network interfaces or performing handover. We thus have implemented a version of *ubiSOAP*, which deals with coordinated usage of the network interfaces, so that the actual network used for communication is selected according to the applications that are currently run. Specifically, a daemon process is introduced, which coordinates the usage of the network interfaces. However, such a solution suffers from the resource availability of PDAs that is still currently limited, and accommodates poorly the concurrent execution of applications. Indeed, as shown by results of Figure 5.f) that provides the response time of *ubiSOAP* with the daemon running, while the overhead on the desktop is reasonable, it increases dramatically when performed on the PDA due to the constant process switching.

## 6 Conclusion

Service-oriented computing appears as a promising paradigm for pervasive computing systems that shall seamlessly integrate the functionalities offered by networked resources, both mobile and stationary, both resource-rich and resource-constrained. In particular, the loose coupling of services makes the paradigm much appropriate for wireless, mobile environments that are highly dynamic. However, enabling service-oriented computing in pervasive networking environments raises key challenges among which overcoming resource constraints and volatility of wireless, mobile devices. This has in particular led to introduce



lightweight service-oriented middleware [10,21,16]. However, to the best of our knowledge, none of the existing solutions comprehensively integrate the full capacity of today's pervasive networking environments, which allow wireless devices to interact via multiple network paths. Such a feature actually enables seamless networking and in particular overcoming the nodes' mobility through vertical handover across networks. This further allows for tuning network usage according to application requirements, and thus enhancing overall QoS.

Exploiting multi-radio connectivity has led to the definition of various algorithms for optimizing the scheduling of communications over multiple radio interfaces, e.g., [12,23,18,20,4]. Building on this effort, this paper has introduced *ubi*SOAP, which implements SOAP transports that leverage multi-radio, multi-network connectivity and may be coupled with lightweight engines like *i*CSOAP [10]. As a result *ubi*SOAP enables the seamless networking of Web services that may be deployed on various devices, including mobile devices like today's smart phones embedding multiple radio interfaces. *ubi*SOAP is now being extensively experimented as part of the PLASTIC European project, being the basis for the development of various pervasive services, ranging from middleware-layer pervasive service discovery to application-layer services in the area of eBusiness, eHealth, eLearning and eVoting. Experiment further shows that the performance of *ubi*SOAP are in general better than default SOAP-over-HTTP transport, thanks to lightweight session management.

Our current work relates to the aforementioned experimentation of *ubi*SOAP for thorough assessment prior to its release under open source license. Our future work relates to further evolution of *ubi*SOAP to meet the numerous requirements of pervasive computing. Part of this effort lies in introducing additional SOAP transports, like an UDP-based one, which significantly improves response time, while affecting reliability [14]. Obviously, *ubi*SOAP needs to be complemented with a number of middleware services to deal with QoS; such an issue is addressed as part of the PLASTIC project where middleware solutions for security and context awareness are being investigated. We are also examining the coupling with semantic-based solutions to enable more precise, yet more flexible, descriptions of pervasive services, as introduced in, e.g., [16].

## References

1. Aijaz, F., Hameed, B., Walke, B.: Towards peer-to-peer long lived mobile Web services. In: Proc. of IIT (2007)
2. Bellur, U., Narendra, N.C.: Towards service orientation in pervasive computing systems. In: Proc. of ITCC (2005)
3. Caporuscio, M., Charlet, D., Issarny, V., Navarra, A.: Energetic performance of service-oriented multi-radio networks: issues and perspectives. In: Proc. of WOSP (2007)
4. Charlet, D., Issarny, V., Chibout, R.: Energy-efficient middleware-layer multi-radio networking: An assessment in the area of service discovery. *Comput. Netw.* 52(1) (2008)
5. Gehlen, G., Aijaz, F., Walke, B.: Mobile Web service communication over UDP. In: Proc. of VTC (2006)

6. Grace, P., Coulson, G., Blair, G.S., Porter, B.: Addressing network heterogeneity in pervasive application environments. In: Proc. of InterSense (2006)
7. Grudin, J.: Group dynamics and ubiquitous computing. *Com. ACM* 45(12) (2002)
8. Gupta, P., Kumar, P.: The capacity of wireless networks. *IEEE Transactions on Information Theory* 46(2) (2000)
9. Huang, H., Cai, J.: Improving TCP performance during soft vertical handoff. In: Proc. of AINA (2005)
10. Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N., Talamona, A.: Developing ambient intelligence systems: A solution based on Web services. *Journal of Automated Software Engineering* 12(1) (2005)
11. Jacquet, P., Mühlenthaler, P., Clausen, T., Laouiti, A., Qayyum, A., Viennot, L.: Optimized link state routing protocol for ad hoc networks. In: Proc. of INMIC (2001)
12. Katz, R.H., Brewer, E.A.: The case for wireless overlay networks. In: *Mobile Computing*. Kluwer Academic Publishers, Dordrecht (1996)
13. Kindberg, T., Fox, A.: System software for ubiquitous computing. *IEEE Pervasive Computing Magazine* 1(1) (2002)
14. Lai, K.Y., Phan, T.K.A., Tari, Z.: Efficient SOAP binding for mobile web services. In: Proc. of LCN (2005)
15. Li, J., Blake, C., Couto, D.S.J.D., Lee, H.I., Morris, R.: Capacity of ad hoc wireless networks. In: Proc. of MobiCom (2001)
16. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: Easy: Efficient semantic service discovery in pervasive computing environments with QoS and context support. *J. Syst. Softw.* 81(5) (2008)
17. Mokhtar, S.B., Raverdy, P.-G., Urbietta, A., Cardoso, R.S.: Interoperable semantic & syntactic service matching for ambient computing environments. In: Proc. of AdhocAmC (2008)
18. Qureshi, A., Gutttag, J.: Horde: separating network striping policy from mechanism. In: Proc. of MobiSys (2005)
19. Raverdy, P.-G., Riva, O., de La Chapelle, A., Chibout, R., Issarny, V.: Efficient context-aware service discovery in multi-protocol pervasive environments. In: Proc. of MDM (2006)
20. Sorber, J., Banerjee, N., Corner, M.D., Rollins, S.: Turducken: hierarchical power management for mobile devices. In: Proc. of MobiSys (2005)
21. Su, J., Scott, J., Hui, P., Crowcroft, J., de Lara, E., Diot, C., Goel, A., Lim, M., Upton, E.: Huggle: Seamless networking for mobile applications. In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) *UbiComp 2007*. LNCS, vol. 4717, pp. 391–408. Springer, Heidelberg (2007)
22. Wang, B., Bodily, J., Gupta, S.K.S.: Supporting persistent social groups in ubiquitous computing environments using context-aware ephemeral group service. In: Proc. of PerCom (2004)
23. Wang, H.J., Katz, R.H., Giese, J.: Policy-enabled handoffs across heterogeneous wireless networks. In: Proc. of WMCSA (1999)
24. Zarras, A., Fredj, M., Georgantas, N., Issarny, V.: Engineering reconfigurable distributed software systems: Issues arising for pervasive computing. In: Butler, M., Jones, C.B., Romanovsky, A., Troubitsyna, E. (eds.) *Rigorous Development of Complex Fault-Tolerant Systems*. LNCS, vol. 4157, pp. 364–386. Springer, Heidelberg (2006)
25. Zhu, F., Mutka, M.W., Ni, L.M.: Service discovery in pervasive computing environments. *IEEE Pervasive Computing* 4(4) (2005)

# Towards a Service-Oriented Approach for Managing Context in Mobile Environment

Waskitho Wibisono, Arkady Zaslavsky, and Sea Ling

Caulfield School of Information Technology, Monash University  
900 Dandenong Rd, Melbourne, VIC, Australia

{waskitho.wibisono,arkady.zaslavsky,chris.ling}@infotech.monash.edu.au

**Abstract.** The current development of context-awareness has introduced various emerging research areas to reduce complexity in developing context aware applications by applying service-oriented approach in managing context and establish context service. The establishment of context service will enable context aware systems to access and utilize context from context providers without paying necessary attention on how context information are composed and managed. Frequent changes of available context providers with different context quality are common phenomena in mobile environment. Hence, dealing with quality of context is a very important issue to provide reliable services for context management in this environment. We have identified some key requirements to establish context service and propose a service-oriented framework to facilitate context management in mobile environment. Furthermore we show our approach to deal with problem in providing appropriate context based on its quality requirements and the preferences of the corresponding context request.

## 1 Introduction

The proliferation of mobile computing and networking technology has led to the emergence of context-aware applications which are capable of adapting current situations in the environment without having explicit user interventions [1]. This phenomenon has highlighted the need to provide reusable support to facilitate management of context independently from applications by establishing *context service* [2].

Service-Oriented Computing (SOC) is a new computing paradigm to support application development by utilizing services as its essential element [3]. This paradigm has motivated us to use service-oriented approach to develop *context service framework* for mobile environment. The development of the service will enable context-aware applications to access and utilize context from context sources without worrying about details of context management [2][4]. Furthermore, it will enhance the reusability of context sources for multiple applications.

Dynamic changes of available context providers that generate contexts with different qualities are common phenomena in mobile environment. Similar contexts may be available concurrently; however the relevance of these contexts to

context-aware applications can vary according to the individual application's requirements and their current situations in environment. Hence, dealing with context quality is also a very important issue to provide a reliable service for context-aware systems in addition to the basic context management mechanisms.

To illustrate these problems, imagine a context-aware application that help a user to find best location for the user to do outdoor activities by providing him/her recommended locations using available environmental information provided by the context service. To perform the task, the application submits necessary context requests to the *context service* and requires a periodic update of contexts from the preferred locations.

In mobile environment, particular environmental information such as temperature, wind or tide of a particular location can be published by multiple context providers. Therefore, similar context information may also be available with different quality information such as *freshness* or *distance* to the desired locations, affected by time and location of data acquisition. Furthermore, other quality information such as *precision*, *probability of correctness* and *spatial resolution* can vary based on individual data aggregation or reasoning technique being used or due to limitation of individual sensors.

In this paper, we have identified several key requirements to establish the context service and propose our service-oriented framework to manage contexts in mobile environment. We adopt ConSpaF [5], a heuristic data fusion technique for situation reasoning based on Context Spaces theory [6], as the basis to infer the *matching confidence* between available contexts in the system and the corresponding context request. The *matching confidence* represents degree that the provided context response can attain quality requirements and preferences defined by applications in their requests. Furthermore, we integrate the notion of *quality of context* into the Context Spaces model and develop our framework prototype as our initial step to establish a context service framework for mobile environment.

## 2 Context in Mobile Environment

Various definitions have been proposed to define *context* in current literature. However, the interpretation of context often depends on the domain in which context is utilized. Schilit et.al [7] defined important aspects of context information in mobile computing in a user-centered view, "*three important aspects of context are: where you are, who you are with, and what resources are nearby*". On the other hand, contexts can also represent very broad information coming from various and dynamic sources while similar information can be provided by different context sources with different data models [8].

Development of pervasive computing application has to deal with information which is captured from real world by sensing devices. The captured information may have different quality information which can be influenced by sensor limitation, data transformation, aggregation or brokering [9]. On the other hand, the

context quality can have significant impacts to influence applications behaviors and their adaptations to the changing environment [10] [11]. Therefore, managing the quality of context in addition to the general context management is a very important issue nowadays.

## 2.1 Quality of Context

Information about quality of context is commonly related to characteristics of sensing devices. Different devices may produce different context with different qualities. For example, an expensive device may be capable to produce information with a high precision compare to low a cost device. Furthermore physical constraint, situation of measurement, transformation process or brokering can also influence quality of the created context information [9].

Bucholz et.al [10] defined *quality of context* (QoC) as "*any information that describes the quality of information that is used as context information*". The quality of context was also described as any inherent information that can be used to determine the worth of information to the applications [9]. The notion of quality of context is different from quality of service since context information has inherent quality metric produced by context sources even when the context is not provided to clients as a service [12].

We refer to [10] [11] [12] to identify important attributes which determine the quality of context. They are:

- Freshness: defines time elapsed between the determination of context information and its delivery. It represents the age of information.
- Precision: defines how precise the information mirrors the reality.
- Probability of Correctness: defines the correctness probability of the provided information.
- Resolution: defines the granularity of information.
- Spatial validity : defines the spatial location in which the context information is applicable.

In the mobile environment, contexts can be generated by distributed context providers. Their quality values can be closely related to physical location and can be less valuable or even not applicable to the current situation of applications. Moreover, the relevance of context for context-aware applications can also change due to user's mobility. For that reasons, providing context with its quality information is essential.

## 3 Context Service Framework

There are several challenges that need to be dealt with when developing context service. In this section we describe our framework and describe its constituent components.

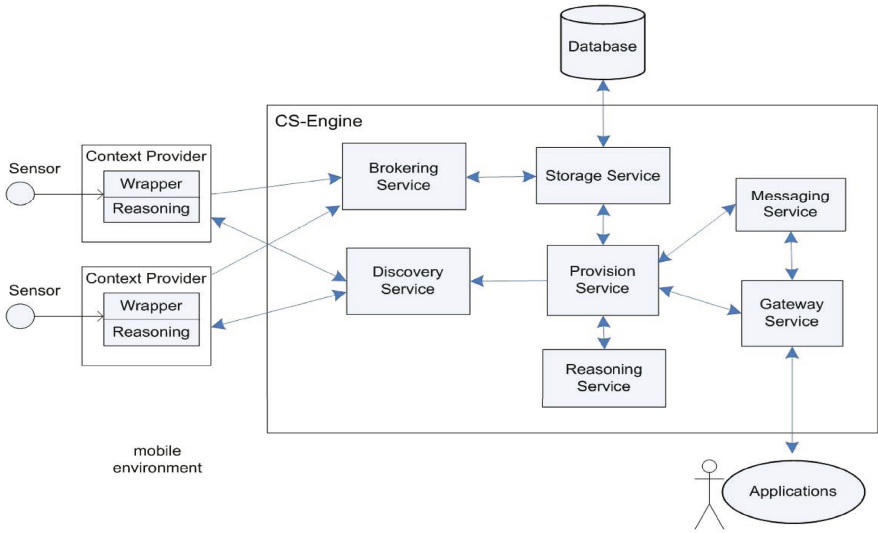
### 3.1 CS-Engine Internal Services

CS Engine is the core of our framework comprising multiple services. It is responsible for facilitating context management and providing relevant context responses for incoming context requests from applications. The services include:

- **Gateway Service:** This service is responsible for handling incoming context requests from applications. A context request can be categorized into either a *direct reply* or a *subscription-based* type of requests. For the direct-reply request, this service can directly invoke the *provisioning service* to obtain the relevant contexts from available contexts in the system. For the subscription-based, the gateway service has to subscribe it to the *messaging service*.
- **Messaging Service:** The *messaging service* has to be established to enable *event-based* interaction for the subscription-based request. This service will send a notification along with the composed context response to the *gateway service* if relevant context of the subscribed request is available.
- **Provisioning Service:** The *provisioning service* is responsible for composing context response for the corresponding context request. This response can be accompanied with *matching confidence* values to represents degrees that the provided context response can attain quality requirements and preferences have been defined in the request. In some case the provision service may need to invoke the *reasoning service* for a particular request that need a new context to be inferred from existing contexts in the systems. The *discovery service* may also need to be triggered to obtain update of the required contexts depends on availability of the registered context providers.
- **Reasoning Service:** The *reasoning service* is a service for deriving high-level context, such as to infer the occurrence of a *situation* given several contexts information by numerous context providers. This component plays a vital role in providing context in mobile environment since necessary context information may not available and context information can be imprecise or inconsistent.
- **Brokering Service:** The *brokering service* facilitates services and flexible mechanisms for context providers to submit their contexts to the system.
- **Storage Service :** This service is responsible for managing and providing services to access the context database. It can also be extended to deal with *security* or *privacy enforcement* issues.
- **Discovery Service :** The service is responsible for managing a list of available context providers in environment. It can also be extended to have capability to notify the available context providers to update their contexts in the system in necessary.

### 3.2 Service Composition

Figure 3.1 illustrates the service composition of the CS-Engine including its interaction with distributed context providers and applications. Generally, each context provider has their own data capturing component and may have different



**Fig. 1.** CS-Engine Services Composition

reasoning techniques to produce contexts from their sensor(s). To provide their contexts to the framework, the context providers can utilize services provided by the brokering service. To obtain context from the framework, applications can submit their context requests to the gateway service and specify type of the request into subscription-based or direct-reply type. Upon receiving the request, the gateway service then decomposes the request to obtain detail of context request including the assigned quality requirements and preferences.

For the *direct-reply* request, the gateway service invokes the provisioning service to obtain context responses accordingly. For the subscription-based, the request will be submitted to the messaging service to facilitate event-based interactions for predefined criteria. Accordingly, the messaging service can organize asynchronous interactions with the provisioning service, to obtain necessary context responses. For both types of context request, the gateway service can then deliver the provided contexts to the corresponding applications.

Generally, processes to generate context response are initiated by extracting details of context request by the provisioning service. The service then gathers relevant contexts from storage service. In special cases, the provisioning service then may ask the reasoning service if it requires derivation of a new context to fulfill context request. In other cases, the discovery service can also be triggered to obtain the latest update contexts from available context providers in the directory list. The Figure 2 depicts the discussed services interaction for processing both types of context requests and the Figure 3 illustrate services interaction for composing context responses.

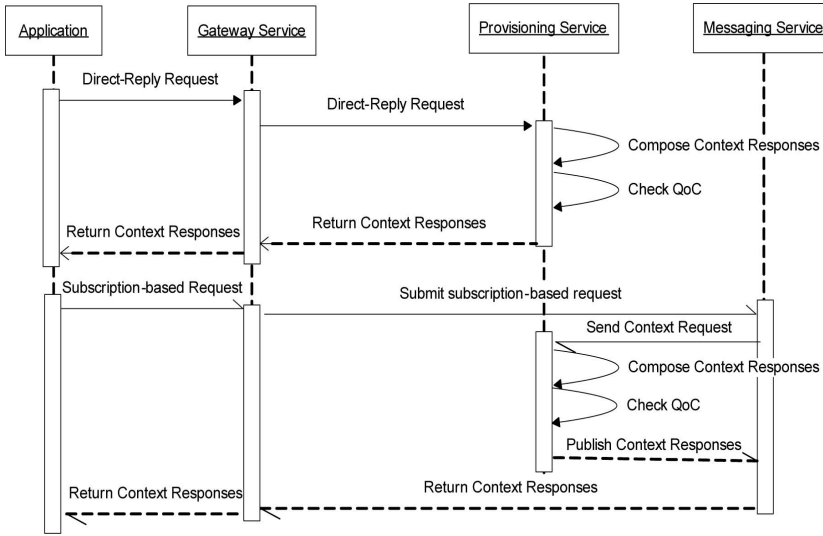


Fig. 2. Service Interaction for processing context request

## 4 Modeling Context and Context Request

Providing contexts responses for numerous applications requires a formal context representation to deal with heterogeneity of context sources. It will enable a service to provide uniform access of context by both context providers and applications. Furthermore, it can also help various and independent context aware applications to be developed with ease and enable collaborations among them.

Padovitz et.al.[\[5\]](#) proposed a general approach that use *geometrical spaces* intuition to model context and to infer situation for context-aware environment called *Context Spaces*. We use Context Spaces as the basis for context modeling, furthermore we have also developed a model of *context request* as an *object* consist of required context's class and attributes to specify quality requirements and predefined weights to specify relevance of each attributes to other attributes in the context request. In addition, individual contribution for each attribute in the request is also need to be specified. Finally, *matching confidence* values for all of relevant contexts then can be computed. These values represent degrees that the contexts can satisfy quality requirements and preferences of the submitted request.

### 4.1 Context Spaces

The Context Spaces[\[6\]](#) defines *situation* in pervasive environment as a collection of *accepted regions* in a multidimensional spaces. It also defines *context attribute* as any type of data that is used in the process of situation reasoning that can be associated with sensor data and denoted as  $a_i^t$ . As an example, a value of



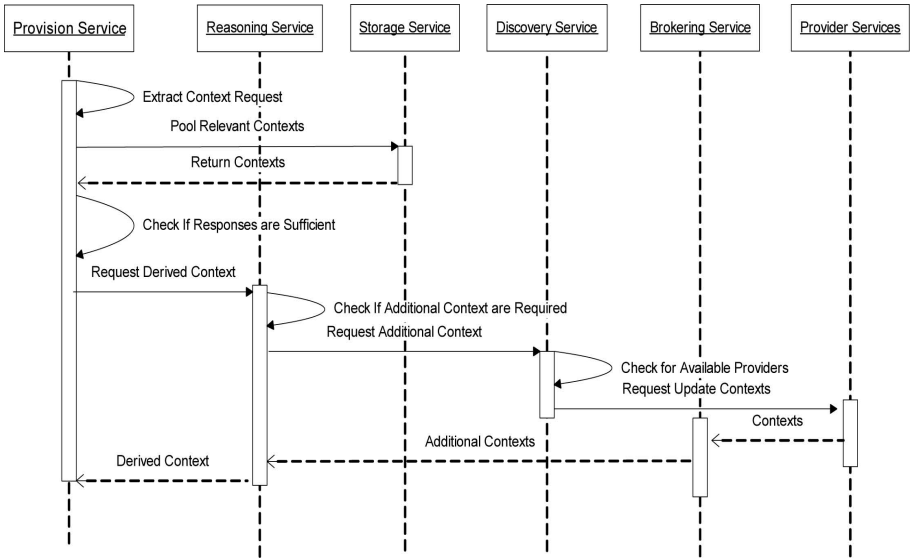


Fig. 3. Services interaction for composing context responses

sensor reading  $i$  at time  $t$  can be represented as context attribute  $a_i^t$ . A vector consists of a collection of context attributes at time  $t$  forms a *context state*. The context state is denoted as  $C^t = \{a_1^t, a_2^t, \dots, a_n^t\}$  that represent a current state of application while  $n$  is defined as the number of context attributes. Context Spaces defines a real situation in context-aware environment as a *situation space* and denoted as  $R_i = \{a_1^R, a_2^R, \dots, a_n^R\}$  that is a collection of  $n$  acceptable regions corresponding to a predefined situation. A region of acceptable values can be defined as a set that satisfy some predicates [5]. The Figure 4 illustrates this concept and show an example of a context state being inside ( $C_i$ ) or being outside ( $C_j$ ) of a defined situation space  $R_i$ .

A heuristic based *data fusion* technique for situation reasoning called ConSpaF [5] was proposed to compute *degree of support* to verify situation occurrences for the Context Spaces model. The fusion approach assigns *weights* for the corresponding regions of the situation space  $R_i$  to represent the relative importance of a region to other regions to infer a situation; and *individual contributions* that specify individual support of a region that a corresponding situation is occurring. Ideas of *symmetrically/asymmetrically contributing* of the context attribute were defined. The *symmetrically contributing* attribute will increase the confidence of situation occurrence if the value of a particular context attribute is inside the corresponding region; otherwise it will decrease the confidence if it is located outside the corresponding region. On the other hand, the *asymmetrically contributing* attribute will not decrease the confidence if the value of the value of the context attribute is outside the acceptable ranges of the corresponding region.

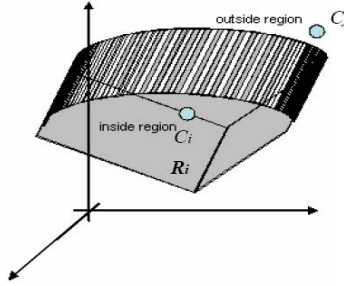


Fig. 4. Visualisation of context state and situation space[6]

## 4.2 Modeling Context Request

In our approach, we combine context *quality attributes* with the *context attribute* for our *context data*. It is defined as a tuple  $(a_i^t, Q)$  where  $a_i^t$  is the *context attribute*  $a_i^t$  and  $Q$  is the set of context quality attribute-values and denoted as a vector  $Q = \{q_1, q_2, \dots, q_k\}$  where  $k \in \mathbb{N}$  represents the number of quality attributes for the context attribute  $a_i^t$ . To model a context request, we create an object called *request space* that represents a context request. The object is composed of the corresponding *context class* and some additional properties consist of regions that represent quality requirements including their assigned weights and individual contributions. We adopt the concept of *situation space* in the Context Spaces to compose the request space. The algorithm 1 shows steps to compose a context request in our framework as illustrated Fig 5.

## 4.3 Inferring Degree of Matching Confidence of Available Contexts

In mobile environment, distributed context providers may produce similar contexts with different qualities. In this sub-section, we discuss how we deal with this problem by adopting ConSpaF [5] data fusion technique to compute *matching confidence* of relevant context data with the corresponding context request concerning its quality requirements and preferences.

To measure this value, Algorithm 2 starts the process by collecting relevant context data in the system followed by extracting their quality attributes into a vector object called *ContextQualityState*. This vector represents the value of quality attributes for the corresponding context data. To compute the degree of matching confidence for each corresponding context data, the Algorithm 2 then invoke Algorithm 3. This algorithm is an adoption of ConSpaF [5] data fusion approach for situation reasoning that is utilized to infer the required matching confidence of a particular context data with the corresponding context request. Figure 7 depicts details steps of the discussed Algorithm 3.

---

**Algorithm 1 : Composing Context Request**


---

```

ContextRequest=new ContextRequest(ContextClass,threshold)
TotalRegionNumber=QocRequirements.totalRequirements
for i=1 to TotalRegionNumber
  Region[i] = new(QoCAAttribute)
  for j=1 to QocRequirements[i].predicateNumber
    QocPredicate[j]= new Predicate(QocRequirement[i].predicate[j])
  end for
  Region[i].AddPredicate{QocPredicate[1],...,QocPredicate[j]}
  Region[i].Set(RelevanceWeight,SymmetricStatus)
  ContextRequest.AddRegion(Region[i],IndividualContribution)
  ContextRequest.SetTimeValidity(timeValidity)
end for
ContextResponse = GatewayService.Put(ContextRequest)

```

Fig. 5. Algorithm to compose context request

## 5 Implementation

In this section, we discuss our initial implementation of the framework. We have simulated a number of distributed context providers and described a scenario to highlight problems in providing relevant contexts in mobile environment in a situation where multiple context providers offer similar context information with different values and quality information.

### 5.1 Application Scenario

To illustrate the problems, imagine a context-aware application that helps a user to find best location for the user to do outdoor activities by providing him/her the choice of best locations using updates of environmental contexts. To obtain these contexts, the application needs to compose necessary context request(s) and subscribes the request along with the quality requirements and preferences to the context service framework. In addition, the application may also need to get a periodic update of necessary context information from all of preferred locations. However, specific environmental information such as temperature, wind or tide at such a particular location in mobile environment can be generated by multiple context providers simultaneously. Therefore, similar context information may have different value of their quality attributes such as, freshness or distance to desired location that relate to time and location of data acquisition; or different values for other quality attributes as identified in section 2.1 which can be influenced by individual aggregation and reasoning technique or the limitation of sensors. In the simulation, we assume that each context provider publish their contexts within the framework along with their quality information.

---

 Algorithm 2 : Extracting QoC attributes and composing context response
 

---

```

for i=1 to totalContextData do
  if ContextData[i].class == ContextRequest.ContextClass then
    new ContextQualityState
    for each QoCAttribute in ContextData[i] do
      Attribute = QoCAttribute.ExtractAttribute
      ContextQualityState.add(Attribute)
    end for
    requestSpace = ContextRequest.getRequestSpace
    MatchingConfidence =
    Algorithm3.Infer(requestSpace,ContextQualityState)
    if MatchingConfidence > requestSpace.treshhold then
      Response[i]= new Response(ContextData,MatchingConfidence)
    end if
  end for
Return (Response[1],...,Response[n])

```

**Fig. 6.** Algorithm to extract QoC attributes and composing context responses

---

 Algorithm 3 : Computing Matching Confidence
 

---

```

for each Region in RequestSpace do
  QualityAttribute = ContextQualityState.getAttribute(Region)
  regionContribution = getContribution( Region,QualityAttribute)
  accumulatedSupport += regionContribution * Region.getWeight
  if Region.Optional == false then
    productSupport * = regionContribution
  end if
end for
q2,q1 = ComputeCoefision(Region[1].Weight,.., Region[n].Weight)
MatchingConfidence = q1*accumulatedSupport+q2*productSupport
Return(MatchingConfidence)

```

**Fig. 7.** Algorithm to compute matching confidence

For an example, local weather stations may publish their observations every 4-6 hours while national bureau of meteorology may update their information on daily basis. Furthermore, there may be numerous individual context providers such as private boats or sophisticated vehicles that are equipped with environmental sensors and communication system which can report occasionally weather information using cellular network or other available connection to the system. Table 1 shows examples of similar contexts produced by distributed context providers including their distances of observation to the required location and Table 2 shows the corresponding quality information we use to simulate our scenario.

As we can see from the Table 2, similar contexts can have different quality information that are influenced by factors that as we have discussed before. For instance, information from national bureau of meteorology may have high probability of correctness (PoC) but may have a low freshness due to its daily

**Table 1.** Available contexts and their providers

ID	Source	Class	Data(m)	Distance (m)
Provider 1	Bureau Meteorology	Tide	2.4	50000
Provider 2	Local Stations A	Tide	3	70000
Provider 3	Local Stations B	Tide	2.2	30000
Provider 4	Local Stations C	Tide	2.1	45000
Provider 5	Private Observation	Tide	1	5000
Provider 6	Private Observation	Tide	0.8	13000
Provider 7	Private Observation	Tide	1.8	8000
Provider 8	Private Observation	Tide	2.5	70000
Provider 9	Private Observation	Tide	1.2	15000
Provider 10	Private Observation	Tide	0.9	3000

**Table 2.** Quality attributes values of the available contexts

ID	Freshness(h)	Resolution (km)	Precision(m)	PoC
Provider 1	20	50	0.2	0.85
Provider 2	4	12	0.15	0.85
Provider 3	6	15	0.1	0.9
Provider 4	5	10	0.15	0.95
Provider 5	1	1	0.3	0.75
Provider 6	0.8	1	0.25	0.8
Provider 7	2	1	0.25	0.8
Provider 8	4	5	0.25	0.7
Provider 9	0.5	0.8	0.3	0.75
Provider 10	0.8	0.8	0.3	0.7

update mechanism. Moreover, the information has a lower resolution since it covers area of 50 square kilometers or even more. On the other hand, a certain environmental information can also varies or even change frequently in each location within smaller areas required by applications.

Distributed local weather stations may provide their contexts that have a spatial resolution between 10-15 square kilometers. However, the context information they provide can have different *precision* or *PoC* in relation to individual sensing equipments or reasoning technique they may use. In addition, *distances* and *freshness* of their observation to particular locations required by the application can vary. In some cases, numerous individual context providers such as private boats that report their observations to the system might have high freshness and spatial resolution in their information, however their information can have low *PoC* or precision which varies according to their sensing equipments. In some cases, they may have closer distances and higher freshness for particular locations of the application. An application need to determine criteria for each quality attributes in the context request. One example of specifying these criteria is shown in Table 3.

**Table 3.** Requirements for quality attributes

Freshness	PoC	Precision	Resolution	Distance
< 6 h	> 0.7	< 0.3	< 11 km	< 10000 m

**Table 4.** Quality attribute preferences

Parameter	Asymmetric / Optional	Importance (1-5)	Contribution
Freshness	No	5	0.95
PoC	No	3.5	0.85
Precision	Yes	3	0.75
Resolution	Yes	3	0.7
Distance	No	4.5	0.9

An example of quality preferences of a submitted context request is illustrated in Table 4. For each quality attribute in the context request, an application then determines weights (from 1 to 5) to represent relative importance of an attribute compared to other attributes of the context request. The individual contribution for each quality attribute is also need to be specified. It defines individual support of a quality attribute in the data fusion process to compute the matching confidence. This table shows that the application has decided that freshness and distance are more important compared to other quality attributes of the submitted context request.

## 5.2 Initial Implementation

We have implemented the prototype of the context service framework using *Java*. We have also simulated mobile context providers who provide their context information over network into our context service framework. A client prototype to simulate the submission of various context requests has been developed to show capability of the framework. This is shown in Fig. 8.

The sorted computation result of matching confidence between available contexts and the defined context request has been defined earlier are shown in Table 5. The table shows that context data produced by provider 7 and provider 5 have higher matching confidence for the defined context request and defined criteria compared data from other providers. Referring to the previous tables, the QoC information of context provided by provider 7 can fulfill all the requirements of the context request while provider 5 can meet all requirements except the precision requirement ( $< 0.3$ ). On the other hand, information data provided by provider 1, has the lowest matching confidence since it can only attain PoC and precision requirements and fail to accomplish other quality requirements e.g. distance, freshness and resolution.

The differences in matching confidence values are also influenced by the assigned weights, individual contributions and symmetrically contributing status of the request as we have discussed before. Finally, by considering the specified

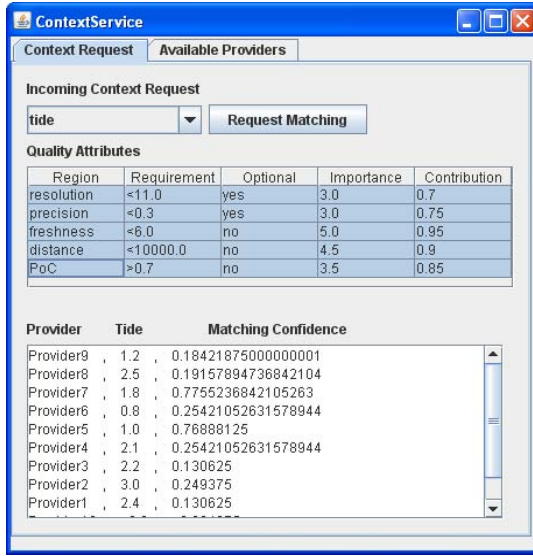


Fig. 8. Basic experimental run

threshold of confidence level of the context request, the acceptable contexts can then be delivered to the corresponding application.

## 6 Related Work

There are many existing approaches of managing context in pervasive environment that relevant to our work. Context Toolkit [13] introduced widgets as an abstraction layer to hide the detail of low-level context sensing mechanisms and uses aggregators to mediate interaction between application and the widgets as well as to provide context aggregation service. Context Broker Architecture (CoBrA) [14] is an agent-based approach for supporting context aware systems in a smart space. CoBrA implements ontology as its data model and enable sharing information and for reasoning the context among the agents. Context Managing Framework (CMF) [15] uses a centralized blackboard-based approach to store and share context from any available context source terminals. The CMF incorporates ontology as the basis for its context data model and provides common vocabulary and knowledge among users.

Some existing approaches have already brought QoC issues to manage their context. CIS [4] and CMF [15] use some QoC parameters for their query interface to enable application to specify the minimum QoC of the request but they do not specify how to compose context responses based on quality attributes requirements and preference of the application if there are multiple context providers that offer similar contexts .

**Table 5.** Sorted computation result

Source	Data	Confidence
Provider 7	1.8	0.775523684
Provider 5	1	0.76888125
Provider 6	0.8	0.254210526
Provider 4	2.1	0.254210526
Provider 2	3	0.249375
Provider 10	0.9	0.204375
Provider 8	2.5	0.191578947
Provider 9	1.2	0.18421875
Provider 3	2.2	0.130625
Provider 1	2.4	0.130625

In this paper we integrate the notion of quality of context into the Context Spaces modeling technique and propose algorithms to provide context responses based on quality preferences and requirements of the submitted context request from applications using a heuristic data fusion technique.

## 7 Conclusion

Frequent changes to incurred context providers, providing varying context quality is common in mobile environment. Hence, dealing with quality of context in managing context is a very important issue to provide services for context-aware systems in the mobile environment

We have proposed an initial service oriented approach to facilitate context management in mobile environment by establishing the context service framework. Our preliminary implementation shows the capability of the framework to provide relevant context in situation where multiple context providers offer similar contexts with different quality information.

Further investigation and development still need to be done to address challenges to facilitate context management in mobile environment such as trustworthiness of context providers, context ambiguity, mobility and frequent disconnection of context providers and their impacts to quality of context. In the next phase of our research, we plan to develop service for data fusion to be incorporated with the reasoning service and enable adaptation of the discovery service to deal with these issues.

## References

1. Baldauf, M., Dustdar, S.: A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2(4), 263–277 (2007)
2. Lei, H., Sow, D.M., Davis II, J.S., Banavar, G., Ebling, M.R.: The Design and Application of a Context Service. *Mobile Computing and Communication Review* 6(4), 45–55 (2002)



3. Papazoglou, M.P., Georgakopoulos, D.: Service-Oriented Computing. *Communications of ACM* 46(10), 25–28 (2003)
4. Judd, G., Steenkiste, P.: Providing Contextual Information to Pervasive Computing Applications. In: *Proceeding of The First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, Fort Worth, Texas, pp. 133–142 (2003)
5. Padovitz, A., Loke, S.W., Zaslavsky, A., Burg, B., Bartolini, C.: An Approach to Data Fusion for Context-Awareness. In: *Fifth International Conference on Modelling and Using Context*, Paris, France, pp. 353–367 (2005)
6. Padovitz, A., Loke, S.W., Zaslavsky, A.: Toward a Theory of Context Spaces. In: *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshop*, Orlando, Florida, pp. 38–42 (2004)
7. Schilit, B.N., Adams, N., Want, R.: Context-Aware Computing Applications. In: *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1994)*, Santa Cruz, CA, US, pp. 89–101 (1994)
8. Broens, T., Quartel, D., Sinderen, M.V.: Toward a Context Binding Transparency. In: *Proceedings of the 13th EUNICE Open European Summer School*, Enschede, The Netherlands, pp. 9–16 (2007)
9. Krause, M., Hochstatter, I.: Challenges in Modelling and Using Quality of Context. In: Magedanz, T., Karmouch, A., Pierre, S., Venieris, I.S. (eds.) *MATA 2005*. LNCS, vol. 3744, pp. 324–333. Springer, Heidelberg (2005)
10. Buchholz, T., Kupper, A., Schiffers, M.: Quality of Context: What It Is and Why We Need It. In: *Proceedings of the 10th International Workshop of the HP Open-View University Association (HPOVUA)*, Geneva, Switzerland (2003)
11. Sheikh, K., Wegdam, M., Sinderen, M.V.: Quality of Context and Its Use for Protecting Privacy in Context Aware Systems. *Journal of Software* 3(3), 83–93 (2008)
12. Huebscher, M.C., McCann, J.A.: Adaptive Middleware for Context-aware Applications in Smart-homes. In: *Proceedings of the 2nd workshop on Middleware for Pervasive and Ad-Hoc Computing*, Toronto, Ontario, Canada, pp. 111–116 (2004)
13. Deu, A.K., Abowd, G.D., Salber, D.: A Context-based Infrastructure for Smart Environments. In: *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE 1999)*, Dublin, Ireland, pp. 114–128 (1999)
14. Chen, H., Finin, T., Joshi, A.: An Intelligent Broker for Context Aware Systems. In: *Adjunct Proceedings of Ubicomp 2003*, Seattle, USA, pp. 183–184 (2003)
15. Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., Malm, E.-J.: Managing Context Information in Mobile Devices. *IEEE Pervasive Computing Magazine* 2(3), 42–51 (2003)

# An Autonomic Middleware Solution for Coordinating Multiple QoS Controls

Yan Liu<sup>1</sup>, Min'an Tan<sup>2</sup>, Ian Gorton<sup>3</sup>, and Andrew John Clayphan<sup>2</sup>

<sup>1</sup> National ICT Australia, NSW, Australia  
jenny.liu@nicta.com.au

<sup>2</sup> University of New South Wales, Australia  
{minant,ajc}@cse.unsw.edu.au

<sup>3</sup> Pacific Northwest National Laboratory, U.S.A  
ian.gorton@pnnl.gov

**Abstract.** Adaptive self-managing applications can adapt their behavior through components that monitor the application behavior and provide feedback controls. This paper outlines an autonomic approach to coordinate multiple controls for managing service quality using executable control models. In this approach, controls are modeled as process models. Moreover, controls with cross-cutting concerns are provisioned by a dedicated process model. The flexibility of this approach allows composing new controls from existing control components. The coordination of their dependencies is performed within a unified architecture framework for modeling, deploying and executing these models. We integrate the process modeling and execution techniques into a middleware architecture to deliver such features. To demonstrate the practical utilization of this approach, we employ it to manage fail-over and over-loading controls for a service oriented loan brokering application. The empirical results further validate that this solution is not only sensitive to resolving cross-cutting interests of multiple controls, but also lightweight as it incurs low computational overhead.

## 1 Introduction

As Service Oriented Architecture (SOA) becomes more widely adopted in large software systems, the typical SOA environment has become more complex. Management of these increasingly complex environments is exacerbated by cross-cutting components and services as well as overlapping SOA environments with service providers beyond the administrator's control. While some human inspection or administration tools can and should be provided, it is unrealistic to expect that all configurations and management can be effectively handled manually. Being fully dependent on manual controls would void the improvements in timeliness and adaptivity gained with an increased level of automation. Consequently, incorporating adaptive and self-managing capabilities into services [4,15] is attracting considerable attention as a means to respond to both the functional and environmental changes that occur after service deployment.

In principle, a system exhibiting adaptive and self-managing capabilities [7][11][12] consists of two parts: (1) a base system that implements the business logic and provides concrete functionalities; and (2) a set of controls that comprise control components for constantly monitoring the system, analyzing the situation and deciding on the actions to affect the system's behavior. When the base system is composed of services in a SOA, the addition of these control components results in adaptive and self-managing service-oriented systems.

In practice, individual control components are dedicated to a specific quality attribute, such as load balancing for performance and scalability or failover for reliability. These are normally constructed independently. In reality, these control components need to be coordinated at runtime to resolve their dependencies that are incurred by cross-cutting concerns. For example, the operation to switch to a backup service may come at a cost of performance by degrading the throughput over a given period of time. While component-based development helps to modularize and encapsulate adaptive and self-managing computation, there is still tight logical coupling and interdependencies between control components. Examples of such tight coupling include systems where the monitoring, analysis and configuration control components explicitly invoke one another without an intervening layer of logical abstraction. Therefore, it is essential to abstract the controls and their dependencies so that their actual implementation is separated from the coordination logic.

In this paper, we propose a novel architecture-based approach to represent, execute and coordinate multiple adaptive and self-managing controls. In this approach, controls are modeled as executable process models. The process engine is integrated with the middleware, allowing the controls to be executed by the same middleware that hosts services. The models can be modified, composed and deployed to the middleware at runtime without affecting the business logic of the services. Moreover, dependencies between controls are also modeled and coordinated using the process models. Our unified approach is realized by an architecture framework, whose default implementation can be further customized and extended to multiple controls. This solution is evaluated by implementing a realistic test scenario. Quantitative measures are collected in terms of the response time overhead, service throughput and CPU usage. The results demonstrate that this architecture solution is lightweight and efficient.

The structure of this paper is as follows: Section 2 discusses the problem through an illustrating example. Section 3 proposes the architectural approach with details of its principle and technical solution. Section 4 further discusses the techniques for coordinating control dependencies. Section 5 presents a case study utilizing this architecture. Section 6 evaluates the overall architecture with measures collected from a test bed. The paper concludes with Section 7.

## 2 The Problem

We use a typical service oriented system to illustrate the problems involved in coordinating multiple adaptive controls. In addition, we derive the requirement of

the architectural solutions from this example. This application is a representative enterprise integration example derived from industry best practices [6].

Consider the loan brokering application in [6], where a customer submits requests for a loan quote to a loan broker. The loan broker checks the credit worthiness of a customer using a credit agency. The request is then routed to appropriate banks who each give a quote, and the lowest quote is returned to the customer. This application has been deployed (see left of Fig. 11) over an Enterprise Service Bus (ESB) with messaging capabilities provided by Java Messaging Services (JMS), bringing together Web Services, Plain Old Java Objects (POJO) and remote Enterprise Java Beans (EJB). Event flow is driven by the arrival of events. In this application as described in [6], there are two scenarios concerned with adaptive and self-managing controls: (1) failover and (2) overload.

**Failover Control.** Suppose the responsiveness to requests of the credit agency is in question, and the administrator wants to allow a graceful failover to an alternative credit agency should the primary agency fail. One solution is to insert an additional switching component between the loan broker and the credit agency that can reroute traffic to an alternative credit agency (see Fig. 11). In this solution, a test message sensor constantly sends test messages to the credit agency to ensure its correct operation. A notification message is sent to the switch to reroute traffic to a backup credit agency if the test message fails. This forms a feedback control between the test message sensor (feedback) and the switch (control). It is worth noting that this failover occurs at the service level: the primary and secondary services can be from different service providers across the organization boundary. Failures at this level cannot be addressed with system level solutions, such as clustering, and need to be explicitly dealt with at a higher level.

**Overload Control.** In addition to ensuring the robustness of the credit agency, the administrator also wishes to prevent the loan broker from becoming saturated with requests. As shown in the right of Fig. 11, a throttling component can be used to regulate the flow of requests by limiting the number of concurrent requests being processed. A traffic flow sensor is also used in this situation to detect the flow rate. Beyond the threshold of the system's computing capacity, higher flow rates reduce the number of concurrent processes handling requests and vice versa.

To support ease of service evolution, the composition of control components with existing services should be transparent to business logic. Hence the control logic should not require modification of the original service operation. However, in a component based implementation of the controls, flexibility is reduced as the control logic would be embedded in the components. For example, if the criteria to trigger the failover switch is changed, a rewrite of the basic switching and test message components would be required to coordinate their logic. Another possible solution is to use a "coordinating" component to control the interaction of components in the feedback loop, but again, a change to the hard coded logic is required to this coordinating component if the control structure changes. The high coupling mentioned is still present.

Moreover, introducing control components also creates dependencies between the business and management flows. For example, the loan broker needs to be

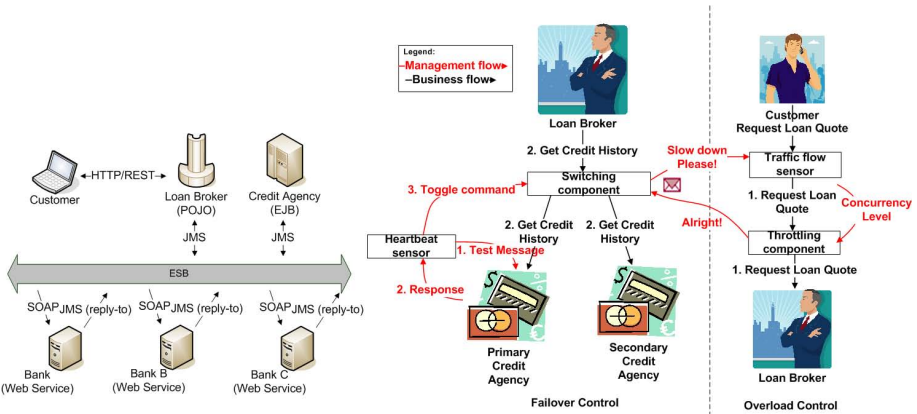


Fig. 1. Self-managing Loan Brokering Service Oriented Systems

aware of the switching component in order to send messages correctly to the switch and not to the credit agency. As more management controls are added, the introduced dependencies both obscure the original business flow as well as reduce the system’s flexibility to changes in both flow types.

The desired solution should therefore address the following architectural requirements: (1) represent, execute and coordinate multiple adaptive and self-managing controls; (2) seamless integration of controls, middleware and service business logic; (3) controls can be composed, modified and deployed at runtime; and (4) the solution should be lightweight, otherwise it could adversely degrade overall performance and scalability. We design an architecture framework to address the above issues, since a variety of middleware mechanisms can be leveraged to realize such a framework.

### 3 The Architecture

We propose a framework to address the relevant architecture issues raised in the prior section. Conceptually, the architecture has five layers. The left of Fig. 2 demonstrates a simplified general architecture with only core components, not specific to any control logic or middleware. The right of Fig. 2 illustrates the customization of the architecture components to specific controls.

The framework aims to provide a modeling based approach towards coordinating multiple controls in service-oriented systems. Adaptive and self-managing controls follow logic that transitions the system from one state into another in response to events dynamically generated at runtime. In addition, the logic represented by the models needs to be executed as well. Given this consideration, we use process models as the tool to present and execute controls. The choice of process models is motivated by their rich semantics, alignment with business goals, integration with Web services and tool support for visual design. The

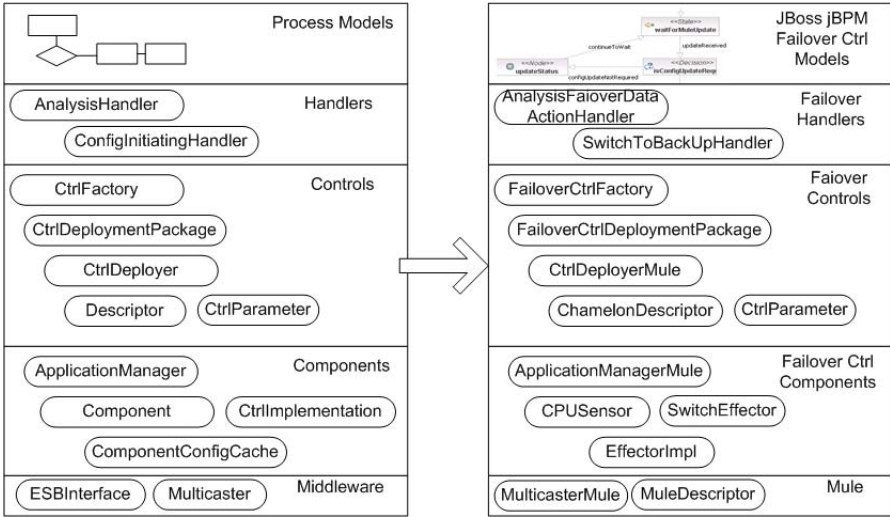


Fig. 2. Simplified Conceptual Architecture

process models can also be executed by process engines with middleware integration features, such as the Oracle BPM engine and JBoss jBPM. In this paper, we use the term *control model* to refer to such a process model designed and executed in a similar way to the JBoss jBPM technology [8].

At the top layer, the control models are firstly designed in diagrams. A model includes nodes for states and transitions triggered by events. Furthermore, these control models are not only for the purposes of presentation, but can be executed. Source code called actions can be attached to either states or transitions of the model. The layer below the control models comprises handlers that encapsulate the action code. Upon entering/leaving a state or before/after triggering a transition, the process engine checks and executes the handler for actions. Our architecture has default implementations for two handlers, *AnalysisHandler* and *ConfigInitialisingHandler*, which are responsible for managing dependencies between control models, and checking a data cache for individual control components respectively. Their usage is addressed in Section 4.1. The combination of these two layers focus on architecture requirement one.

Fig. 3 shows a sample GUI for designing a control model and attaching action code to it. These actions are encapsulated in handlers, and can be executed by a process engine. Such an engine can be embedded at

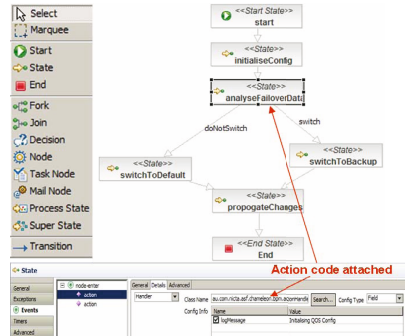


Fig. 3. Control Models and Action Attachment

the middleware level. Therefore, the advantage of using process models in modeling controls is that controls can be visually designed and executed. In addition, the integration of the model execution and the middleware is much simplified by the process engine. This approach is similar to that used in tools such as JBoss jBPM, which is a realization of a full-fledged process engine with IDE support to design process models [8].

The component layer aims to address architecture requirement two. The realization of controls depends on monitoring and actuating components, such as software *sensors* to collect status data to feed into the model, and *effectors* to execute actions. These components are placed into the component layer to separate the control implementation from the business logic. The *ApplicationManager* is responsible for initializing the component instances. As control components participate in service oriented applications, each component has a unique endpoint as its identifier, so that messages can be received from and sent to individual components by service bus middleware.

The control layer aims to fulfill the architecture requirement that controls can be composed, modified and deployed at runtime. Control components are deployed as the unit of the *ControlDeploymentPackage*. Each control has a default *ControlDeploymentPackage* generated by the framework. It contains methods to access all the components involved in a control. Each *ControlDeploymentPackage* uses the *ControlDeployer* to deploy its components. The *ControlDeployer* is responsible for (un)deploying components, creating component *descriptors* and setting the implementation class for each component. This separated deployment of the component instance from its actual implementation further enhances the customization of the adaptive controls. This is because the modification of the implementation does not impact the control models nor the deployment structure, and the implementation can be updated at any time. Once the deployment is finished, an event is broadcasted to other controls about the availability of the new control components.

The bottom layer is the middleware platform. In this paper it is a specific Java ESB – Mule [14]. Mule platform specific mechanisms are used to devise utilities such as concurrency configuration and event multicasting.

In summary, the architecture supports visual and declarative design of adaptive control logic. Controls are modeled as executable process models. These models are executed by a dedicated process engine, which is seamlessly integrated with the middleware. Hence these models can interact with service applications hosted by middleware, receiving and sending messages to realize the control logic. In addition, the architecture supports dynamic update and deployment of controls. As a result, the overall architecture is loosely coupled between business logic and adaptive controls. In the following sections, we further discuss the coordination of multiple controls.

## 4 Techniques of Coordinating Controls

A challenging issue to solve in this architecture is control dependencies occurring at runtime. Controls designed and deployed independently may involve cross-cutting

concerns. For example, Fig. 10 illustrates that when the failover control takes place, it requires the collaboration from the overload control to slow down its current processing for the period of time that the failover is being executed.

Our architecture can address this issue by the techniques of modeling such concerns as coordination controls. The components coordinated are the *sensors* and *effectors* from individual controls. The dependencies are declared in a control model representing the cross-cutting concern; the specific resolution strategy, be it by heuristic hints or some form of machine learning, consists of implementation specific handlers attached to the process nodes. This leverages on the architecture framework presented, building on the basic idea of sensors, effectors and coordinating components. In the following subsections, we discuss the technical details of achieving such coordination among multiple controls.

#### 4.1 Control Dependencies and Composition

In our approach, the dependencies of controls are declared by developers in a dedicated coordination control, as discussed at the top layer of the architecture. The developer registers controls with dependencies using an *AnalysisHandler* that belongs to the handler layer. This coordination control is modeled and deployed the same way as ordinary QoS controls. When it is deployed, another handler, a *ConfigInitialisingHandler*, checks if an instance of the registered controls exist. After the *ConfigInitialisingHandler* checks the controls and their dependencies, the *AnalysisHandler* can retrieve the *configuration* of individual components in one control. A configuration is part of the control layer. It is an abstraction of what the component does. A configuration contains information about interfaces and properties of a component. Through the configuration, the coordination control and the *AnalysisHandler* can access data that the component contains, invoke its interface on behalf of the coordination and change property values in order to change the control parameters. Sample code is shown in Fig. 4.

Using this approach, individual controls are not aware of other controls nor their dependencies. They are transparently managed by coordinating controls. This approach also benefits from the architecture in that a coordination control can flexibly be composed by existing components, which allows quick composition and prototyping of alternative options for adaptive and self-management strategies. An example of composing coordination controls is given in Section 6.1. In addition, coordination controls can be updated, deployed

```
public AnalyseDataHandler(){
    registerDependency("Overload");
    registerDependency("Failover");
} //Register dependencies

OverloadConfig qosConfig = (OverloadConfig)
ConfigInitialisingHandler.getSensorConfig(executionContext,
"Overload");
FailoverConfig failoverConfig = (FailoverConfig)
ConfigInitialisingHandler.getSensorConfig(executionContext,
"Failover");
// retrieve component configuration

ComponentPoolConfig cpConfig = new ComponentPoolConfig();
cpConfig.setId("QosEffector0");
...
if(testeeIsDown(failoverConfig.getTesteeStatus()) &&
failoverConfig.getMessageBacklog() >
MESSAGE_BACKLOG_THRESHOLD) {
    cpConfig.setInterval(qosConfig.getInterval() +
THROTTLE_INTERVAL);
} else if(cpConfig.getInterval() > 1000){
    cpConfig.setInterval(qosConfig.getInterval() -
THROTTLE_INTERVAL);
}
//set new configuration
```

Fig. 4. Code Sample



or undeployed at runtime. This equips developers with the flexibility to trial-and-test different designs.

## 4.2 Control Deployment

The deployment of controls takes two steps. First, the control design models in the format of an XML file are deployed to the process engine using an IDE shipped with the process engine. Any action code is attached to the states or transitions in this model. Second, the unit of deployment *ControlDeploymentPackage* in our architecture framework is generated, with a mapping to the component implementation record. Following this, the *ControlDeploymentPackage* invokes the *deploy()* method of *ControlDeployer* to deploy itself, creating instances of participating components using their descriptors.

Besides the above functionality of deployment, the architecture requires the ability to intercept incoming requests, and modify outgoing messages. This is also achieved through the control deployment. The control deployment automatically generates intercepting components as a proxy to the intercepted components. The intercepting component takes the identity – the unique endpoint of the intercepted component and forwards requests to and replies from the intercepted components. This feature enables the control composition by redirecting messages to/from any other component transparently to the intercepted components. Fig. 5 depicts components before and after the deployment of the overload control. Details of this control are discussed in Section 5.3.

## 4.3 Quality Attributes and Optimization

An important architecture requirement discussed in Section 2 is that the computing overhead incurred by this architecture should be optimized. By nature of this service oriented architecture, the optimization problem falls into the category of minimizing messaging overhead. Research on messaging oriented middleware and Web services has demonstrated that the communication rate and payload of the messages have a significant impact on the overall performance and scalability of SOAs [10]. Hence our optimization focuses on reducing the number of messages and their payload with regards to sending collected data among control components including sensors, data analyzers and effectors. Rather than wrapping data as a SOAP attachment, data collected by sensors are stored in a distributed cache. Whenever necessary, a distributed cache is attached with the control components such as software sensors. In this case, we select an open source distributed cache framework – Ehcache [13]. The performance and scalability of Ehcache has proven to satisfy large scale distributed systems [5]. In order to correlate data collected from different sensors, a sensor aggregation component is created at deployment time. In this paper, a default time-based correlation is implemented in the aggregator. The only limitation with using a distributed cache is that the data transition is separated from the web service messages and it is specific to the distributed cache framework.

## 5 Example Application

We demonstrate our architecture solution using the loan brokering services discussed in Section 2. In addition to verifying the feasibility of our architecture in implementing a practical set of services, we also highlight the flexibility of our architecture for trial-and-test deployments by providing two options to coordinate the failover and overload controls, subsequently referred to as simple and auction-based coordination. In this section, we discuss the specifics of the individual components making up our implementation, as well as two coordination heuristics employed.

### 5.1 Overload Control

The overload control implements the classic token-bucket algorithm for admission control. It consists of a *Token Bucket Sensor*, a *Throttling Component*, a *Throughput Sensor* and a *Coordination Component*, as shown in Fig 5. The *Token Bucket Sensor* maintains a token bucket with  $x$  tokens, where a single token is used for each request. If no tokens are available, the request is dropped and does not enter the system. The token bucket is refilled at rate  $\lambda$ . The *Throttling Component* controls  $W$ , the number of concurrent requests that can be processed. Each processed request is delayed by a throttling interval  $I$ . The *Throughput Sensor* measures  $\delta$ , the rate of requests being processed by the system. Finally, the *Coordination Component* constantly aggregates the throughput  $\delta$  and the number of tokens left in the token bucket. It then feeds the status to the control model in the process engine, and multicasts to effectors the decision on the new values of  $\lambda$ ,  $W$  and  $I$  accordingly. The adjustment of  $\lambda$  is given by:

$$\alpha * \frac{W}{I} + (1 - \alpha) * \delta$$

where  $\alpha$  is a tuning parameter to adjust the component weight.

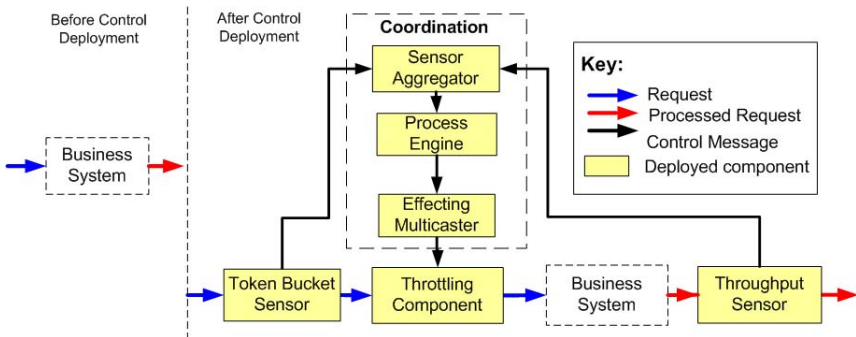


Fig. 5. Overload Control Deployment

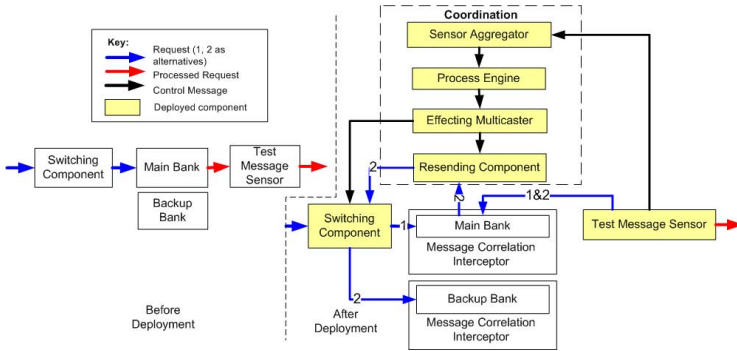


Fig. 6. Failover Control Deployment (switch in  $off^1$  or  $on^2$  mode)

### 5.2 Failover Control

The failover control shown in Fig. 6 consists of a *Test Message Sensor*, a *Switching Component*, a *Resending Component* and a *Coordination Component*. The *Test Message Sensor* constantly sends test messages to the main service. It uses the test messages to determine if the main service is active or has failed. The *Coordination Component* constantly receives inputs from the *Test Message Sensor* and adjusts the state of the *Switching Component* (on or off). If the main service has failed, the *Switching Component* routes incoming requests to the active service when its state is toggled to on by the *Coordination Component*. A *Message Correlation Interceptor* maintains a queue of messages by intercepting incoming requests to the main service. When a request is successfully routed, the request is removed from the queue. The *Resending Component* sends unprocessed requests from the *Message Correlation Interceptor* to the active services when the *Switching Component* is toggled.

### 5.3 Coordinating Multiple Controls

To coordinate these controls, our general approach is to let the overload control throttle the workload when failover takes place. In our implementation, two options are provided to realise this approach.

Our implementation of the architecture is deployed as shown in Fig. 7(a), which also shows the failover and overload controls employed. The core of the coordination is the control model shown in Fig. 7(b). This was created using the JBoss jBPM process model designer. As both *Coordination Components* multicast their status data, the coordination between these two controls collects updated status data from each control using its *SensorAggregator* and sends out action decisions through the *EffectingRouter*. Handlers are attached to nodes and transitions to realize the two control options: (1) simple coordination and (2) auction-based coordination.

The simple coordination control tunes the concurrency level of processing new, incoming requests in the middleware. The tuning is based on the number

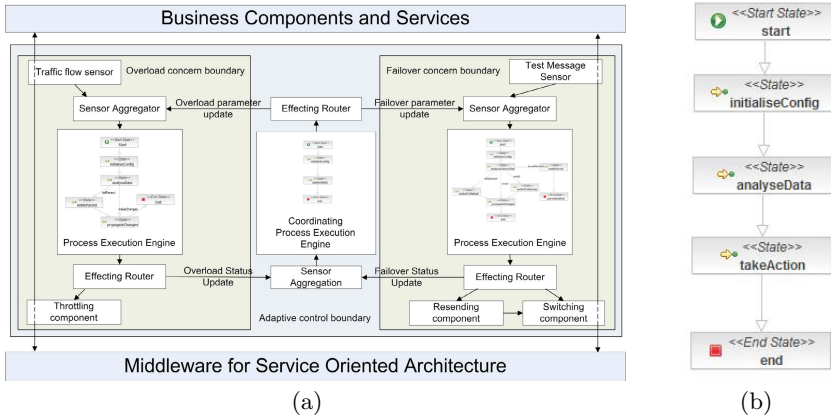


Fig. 7. Implemented Example Architecture (a) and Coordination Control Model (b)

of messages yet to be resent by the failover control. This control is easier to implement, but has limitations when producing the optimal concurrency levels for a large set of services.

In the auction-based control, requests being resent by the failover control and new incoming requests at the overload control bid for tokens. Tokens are dynamically allocated to requests both from failover and overload controls. Only requests with a token can be processed, otherwise there is a wait for the next available token. In general, the auction-based control incurs more overhead in communication as a bid is multicast. However, the auction-based control is more practical and suitable when it is nontrivial to tune the concurrency level of the middleware.

Both options reuse the failover and overload controls, and it should be noted that the control model is identical for both options. The difference is in the way each of them process status data and the actions taken. This is reflected by the different options having different handlers attached to the appropriate control model nodes.

### 5.4 Discussions

In this example, process modeling tools and middleware mechanisms are used to customize the general architecture to a specific implementation. As mechanisms from middleware (such as interceptors) and modeling features from the process engine (such as handlers) are commonly supported, other process modeling tools and service bus middleware can be applied to the framework. The only condition however, is that the process engine should be able to communicate with the middleware. For example, Mule provides a common interface for process engines to access its features [14]. We could have used an Oracle BPM implementation of the interface instead of JBoss jBPM, without any change to other implemented components. This illustrates the generic nature of our architectural solution.

## 6 The Evaluation

Each option of the coordination control is measured and the results are compared to identify key performance factors.

### 6.1 Testbed Setup

We deploy the loan brokering services as shown in Fig. 11 on the Mule ESB. The credit agencies are developed as Java EJBs and deployed on a JBoss application server. Bank services are Web services deployed on Apache Tomcat servers. The brokering service is a Mule application, and it communicates with other services through Mule. The adaptive controls (failover and overload) are designed using JBoss jBPM and their models are deployed into the jBPM engine. The handlers and control components are built upon the architecture framework discussed in Section 3 using Java. Together with the process engine, the models and components are deployed on Mule.

We also develop a simple workload generator which injects a number of requests into the system under test with a bounded random time between request arrivals. For example, the interval [75,200] means the request arrival time bound is between 75 to 200 milliseconds. In order to observe performance, a simple console showing charts of metrics was developed (see Fig. 8 for example).

The testing environment includes two identical Windows XP machines with 2.4GHz Dual Xeon Processors, one hosting loan broker services, credit agencies and adaptive controls and the other hosting five bank services which are identical to simplify implementation. The workload generation are 500 requests with the interval [75,225]. If the overload control is enabled, the throttling component controls  $W$ , the number of concurrent requests that can be processed.  $W$  is set to 100 initially.

### 6.2 Performance Results

We test four scenarios: (1) only the overload control is enabled; (2) only the failover control is enabled; (3) simple coordination; and (4) auction-based

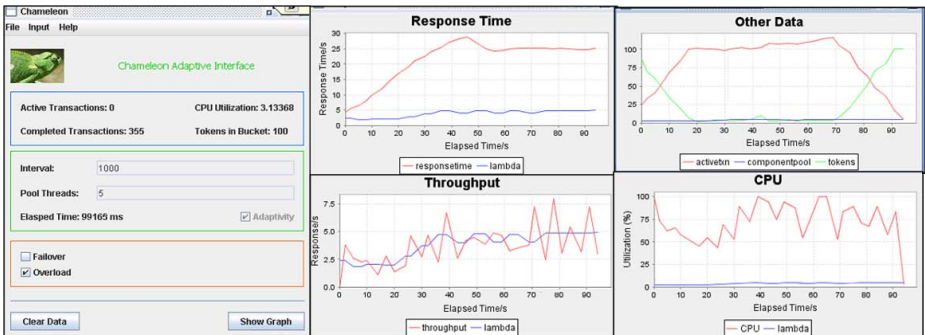


Fig. 8. Overload Control Only Observation

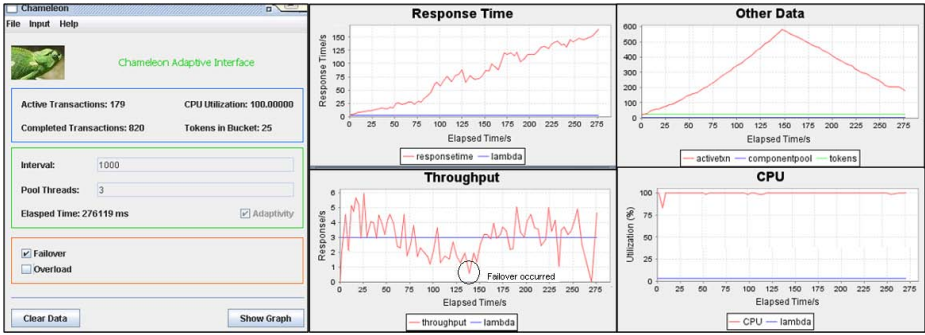


Fig. 9. Failover Control Only Observation

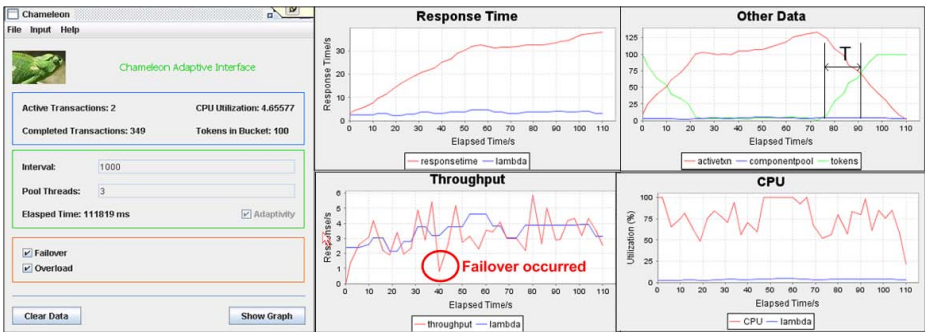


Fig. 10. Simple Coordination Observation

coordination. Obviously in (3) and (4) both failover and overload controls are enabled. The same environment configurations are used for each test. Fig. 8 to Fig. 11 show sample performance measurements from the testing scenarios.

Fig. 8 shows that the overload control is efficient in self-management of performance and scalability. It is shown on the other data chart (top right of Fig. 8) that approximately after 20s has elapsed, the token number hits zero, meaning there are already 100 requests being processed. From the CPU chart, it shows that the CPU utilization starts increasing and it triggers the overload control before 40s elapsed time. The response time chart illustrates that the overload control takes effect around 45s elapsed time, and the response time reaches a plateau rather than continuing to linearly increase.

Fig. 9 shows the failover control also works. At around elapsed time 140s, the primary credit agency service is deliberately shut down, and the requests are routed by the failover control to the secondary credit agency service. This is consistent with the other data chart that shows the active transactions reach the peak at around elapsed time 140s, and then degrades when the failover occurs. The CPU resource is saturated without the overloading control and the response time increases. These separated performance testing scenarios confirm

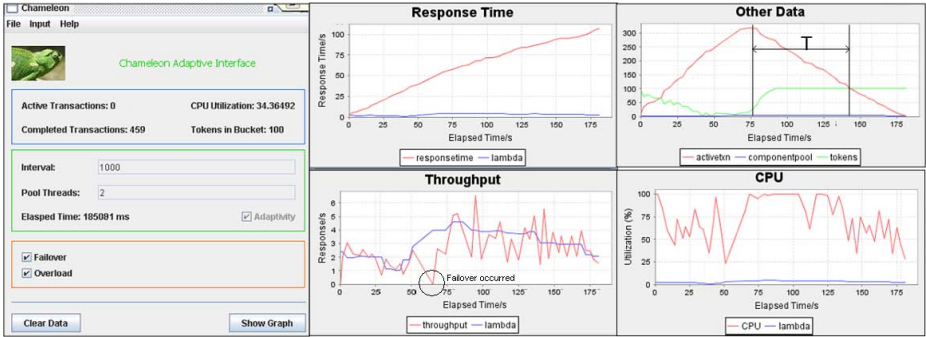


Fig. 11. Auction-based Coordination Observation

the motivation for coordinating two controls to yield a better quality of service. The results from a single control indicate that the overhead of the architecture framework itself is insignificant, and the performance factors are determined by the adaptive self-managing strategies.

The results of the simple coordination are shown in Fig. 10. Compared with the case of failover control only, the coordination helps to improve the performance. Now the response times reach the plateau and the CPU utilization is not saturated. From the results of the auction-based coordination shown in Fig. 11, the performance improvement is less than the simple coordination, which we attribute to the additional communication overhead incurred in the auction-based coordination as mentioned in Section 5.3. An interesting observation is the time (annotated as  $T$  in the diagrams) spent on processing queued requests. Requests are put in a queue by the overload control when all the tokens are consumed, and are only processed when the token bucket is refilled and more tokens are available. The auction-based coordination spent longer time ( $T$ ) than the simple option, which contributes to the degradation of performance.

It is worth noting that our focus is not on studying and evaluating individual coordination controls but rather on demonstrating the practical usage of the architecture to compose them. The difference observed by prototyping and testing shows that our architectural solution can be applied to the development of realistic self-managing service oriented systems. The resulting architecture framework to support this solution is useful to trial different control options.

## 7 Related Work

Applying business process models to support self-managing software systems has recently been investigated in various domains [2, 18]. For example, Verma and Sheth envisioned autonomic web processes [18]. Similar to the core of this paper, they elevated autonomic computing concepts from infrastructure to a process level. Their paper discussed existing technologies and steps needed to shorten the gap from current process management to autonomic web processes.

In this paper, we present a practical architecture solution to integrate process management with middleware-based services.

Extensive research has been done in the translation of business process models to execution languages, and there has been an increasing adoption of business process modeling (BPM) tools to coordinate business process flows, as opposed to hard-coded application logic. A good summary of the relevant techniques and tools is covered in [1]. As the core of our approach, control models leverage business process models to represent adaptive logic. It is an open research question with regards to the integration of the semantics essential to adaptive self-management with general business process modeling capabilities. The on-going research in evaluating the expressiveness of business process definitions will contribute insights to this research space [19].

A comprehensive survey [3] discussed existing technologies that could enable dynamic composition of adaptive software. It also classified different approaches by how, when and where composition might occur. The core of all these approaches was intercepting and redirecting interactions among program entities. These mechanisms help to customize our architecture framework to a specific middleware platform.

## 8 Conclusion

This paper proposes an approach to develop adaptive self-managing controls for service oriented systems. The contribution of this work is twofold. Firstly, it leverages process models to design adaptive controls with a visual context. Moreover, an architecture framework is built to integrate the models with a middleware platform and enable the execution of the design models. Secondly, multiple controls can be coordinated using this solution. Different options can quickly be prototyped by composing the coordination from existing control components. This architecture-based solution provides benefits towards modifiability, reusability and maintainability of self-managing service oriented systems. The quantitative evaluation is based on a realistic enterprise service bus application. The results demonstrate the performance efficiency of this approach. Our on-going work involves developing tools to simulate the controls when it is designed and deployed from the process models. The simulation tools combined with the rest of the architecture framework further help developers test their adaptive and self-managing controls at an early design stage.

## References

1. van der Aalst, W.M.: Business process management demystified: A tutorial on models, systems and standards for workflow management. In: Lectures on Concurrency and Petri Nets, pp. 1–65 (2004)
2. Baresi, L., Guinea, S., Pasquale, L.: Self-healing bpm processes with dynamo and the jboss rule engine. In: ESSPE 2007: International workshop on Engineering of software services for pervasive environments, pp. 11–20. ACM, New York (2007)



3. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. *Computer* 37(7), 56–64 (2004)
4. Naccache, H., Gannod, G.C.: A self-healing framework for web services. *Icws 00*, 345–398 (2007)
5. Gorton, I., Wynne, A., Almquist, J., Chatterton, J.: The MeDICi Integration Framework: A Platform for High Performance Data Streaming Applications. In: *WICSA 2008: 7th Working IEEE/IFIP Conference on Software Architecture*, pp. 95–104. IEEE Computer Society, Los Alamitos (2008)
6. Hohpe, G., Woolf, B.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, Reading (2003)
7. IBM. An architectural blueprint for autonomic computing. *IBM Autonomic Computing* (2004)
8. JBoss jBPM, <http://www.jboss.com/products/jbpm>
9. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Tool Support for Model-Based Engineering of Web Service Compositions. In: *Proc. of Intl. Conf. on Web Services (ICWS 2005)*, pp. 95–102. IEEE Computer Society, Los Alamitos (2005)
10. Juse, K., Kounev, S., Buchmann, A.: PetStore-WS Measuring the Performance Implications of Web Services. In: *CMG 2003: Proc. of the 29th International Conference of the Computer Measurement Group* (2003)
11. Kephart, J.O.: Research challenges of autonomic computing. In: *ICSE 2005: Proceedings of the 27th international conference on Software engineering*, pp. 15–22. ACM, New York (2005)
12. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: *FOSE 2007: Future of Software Engineering*, pp. 259–268. IEEE Computer Society, Los Alamitos (2007)
13. Luck, G., Suravarapu, S., King, G., Talevi, M.: EHCACHE Distributed Cache System, <http://ehcache.sourceforge.net/>
14. Mule ESB, <http://mule.mulesource.org/>
15. P.M., et al.: The wsdm of autonomic computing: Experiences in implementing autonomic web services. In: *SEAMS 2007: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, p. 9. IEEE Computer Society, Los Alamitos (2007)
16. Anthony, R.J.: Policy-based techniques for self-managing parallel applications. *Knowl. Eng. Rev.* 21(3), 205–219 (2006)
17. Kumar, V., Cooper, B.F., Eisenhauer, G., Schwan, K.: Enabling policy-driven self-management for enterprise-scale systems. In: *HotAC II: Hot Topics in Autonomic Computing on Hot Topics in Autonomic Computing*, pp. 4–23. USENIX Association (2007)
18. Verma, K., Sheth, A.P.: *Autonomic Web Processes*. LNCS. Springer, Heidelberg (2005)
19. Zhu, L., Osterweil, L., Staples, M., Kannengiesser, U., Simidchieva, B.: Desiderata for languages to be used in the definition of reference business processes. *International Journal of Software and Informatics* 1, 37–65 (2007)

# Transparent Runtime Adaptability for BPEL Processes

Adina Mosincat and Walter Binder

Faculty of Informatics, University of Lugano, Switzerland  
adina.diana.mosincat@lu.unisi.ch, walter.binder@unisi.ch

**Abstract.** Dynamic service binding is essential for runtime adaptability of BPEL processes, particularly in the case of service failure. BPEL's support for dynamic service binding is coupled with the process business logic, requiring the process developer to deal with dynamic service selection and failure recovery. Changing these aspects requires modification and redeployment of all affected processes. In this paper we present a novel infrastructure that handles dynamic (re)binding of stateful and stateless services independently of process business logic. Our infrastructure is transparent both to the process developer and to the BPEL engine. It offers automated failure recovery and allows for runtime customizations, such as changes of service binding policies. We also assess infrastructure overhead and explore the impact of service failures on system throughput.

## 1 Introduction

The Business Process Execution Language (BPEL)<sup>1</sup> is widely used for web service composition. A process<sup>2</sup> defines business logic by modeling message exchange sequences in an executable manner, invoking service functionalities described in WSDL<sup>3</sup>. Processes are executed by BPEL engines, which create a *process instance* when one of the receive activities (a start activity) in the process is triggered, and end the instance after completion of the corresponding reply activity. The process instance interacts with services (partner links) through invoke activities.

While processes benefit from service features, such as accessibility and platform independence [1], they also have to deal with unpredictable changes. In an open, dynamically changing environment such as the Web, the capability of processes to adapt to changes in the environment is vital. For processes this means that every process instance must be able to dynamically bind the necessary services.

The changes a process should adapt to include the availability of new, better services, but also the failure of bound services. BPEL offers constructs that can

<sup>1</sup> <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>

<sup>2</sup> In this paper we assume that a web service composition is represented as a BPEL process and we will refer to it as *process*; the term *service* refers to a web service.

<sup>3</sup> <http://www.w3.org/TR/wsd120/>

be used to implement different fault-tolerance strategies [2], and the dynamic partner link assignment can be used to change the partner link that represents a service in a process instance. However, it is neither possible to add new services at runtime, nor to change the service selection strategy at runtime. Furthermore, BPEL's support for dynamically binding services is coupled with process business logic.

The generic infrastructure presented in this paper addresses these issues by separating *service selection* and *fault handling*, specified by customizable *binding policies*, from the process business logic. Our approach does not require the process developer to provide fault-tolerance constructs and allows for policy updates without disrupting running processes. Our solution provides runtime adaptability for the process by automatically transforming the process at deployment time. The services used by the process are dynamically selected for every process instance by a customizable selection strategy. Our approach achieves transparency for the process developer, compatibility with standard BPEL engines, and openness for integration with existing service discovery and monitoring mechanisms.

An important concern when dealing with failure recovery is the state of the failed service. While a stateless service can be replaced upon each invocation, a service replacing a stateful service must first be brought into the right state. One way to achieve this is to restart the process upon replacement of a stateful service. Our solution caters to automated process restart in order to recover from the failure of a stateful service. Existing user-defined compensation handlers are executed before process restart.

As an example scenario, consider a process provided by a mobile network operator that takes a start and a destination address and gives travel information (directions, traffic information, weather report, points of interest along the route, etc.). The process relies on two types of external services,  $S_1$  that translates a textual address into GPS coordinates, and  $S_2$  that provides travel information for a trip between two GPS coordinates. While  $S_1$  is stateless, providing a single translation operation,  $S_2$  is stateful and requires two interactions, first the creation of a session specifying the kind of travel information the user is interested in, and second the query (within the previously created session) for travel information for a given trip. While  $S_1$  can be replaced with another service with the same interface upon each invocation, replacing  $S_2$  upon query failure presumes that first an appropriate session is created in the replacement service. Our approach handles such a situation by automatically restarting the whole process before binding a stateful replacement service.

In the aforementioned scenario, the process is not very complex, involving only a few service invocations. However, it may be concurrently invoked by a large number of clients who shall experience a highly available service. As long as there are replacement services, our infrastructure hides service failures from the clients. Moreover, service selection and failure handling behavior can be changed by the operator without disrupting the service offered to the clients.

The scientific contributions of this paper are threefold: (1) We introduce a novel approach to transparently (re)binding both stateful and stateless services for every process instance, enabling fault-tolerant process execution. (2) We provide a customizable solution for separating failure recovery and service selection from process business logic. (3) We implemented our architecture using state-of-the-art technologies and present detailed evaluation results, exploring the overhead caused by our infrastructure as well as the impact of service failures on overall system throughput.

This paper is structured as follows: Section 2 presents the architecture of our infrastructure. In Section 3 we describe the process transformations performed at deployment time. Section 4 explains the interactions between the BPEL engine and our infrastructure. The customization points, binding policies and service selection strategies, are described in Section 5. We evaluate our approach in Section 6 and discuss related work in Section 7. Section 8 concludes this paper.

## 2 Architecture

Fig. 1 shows our system architecture. In this section we describe the responsibilities of the *Transformation Tool* and of the *Bind System*, which includes the *Service Manager* and the *Bind Manager*. Fig. 2 illustrates the transformations upon process deployment. Fig. 3 defines important concepts used throughout this paper. Some operations provided by the *Bind System* are presented in Fig. 4, as far as necessary for the discussion of our system.

The *Transformation Tool (TT)* is responsible for automatically transforming the process at deployment time to use our infrastructure, assuring complete transparency to the process developer. The results of the transformation are two processes, the *transformed process* and the *wrapper process*. The transformed

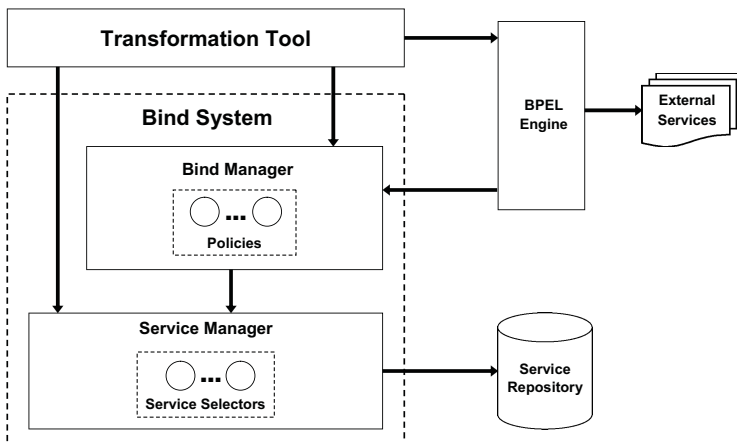


Fig. 1. System architecture

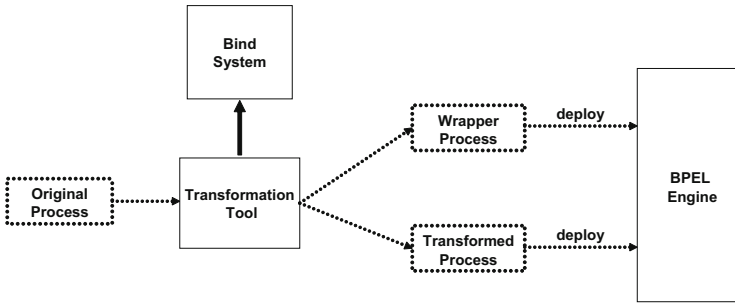


Fig. 2. Process transformation at deployment time

**Service Type (ST)** – Unique identifier of a group of equivalent services that can substitute for each other.

**Process Name (PN)** – Unique identifier of a process in the system.

**Process Identifier (PID)** – Unique identifier of a wrapper process instance in the system. Instances of a transformed process have the same PID as the wrapper process instance that has started them.

**Service Identifier (SID)** – Unique identifier of a service (service address<sup>4</sup>).

**Action on Failure (AOF)** – The action to be taken on service failure. It can take one of the following values:

- *retry* – Invoke a replacement service without restarting the process; used for stateless services or if the failure happens on the first invocation of a stateful service.
- *restart-process* – Automatically restart the process and invoke a replacement service; used for stateful services (if the failure happens after a successful first invocation).
- *throw* – Throw the original service fault which may be handled by the fault handlers defined in the original process.

Fig. 3. Definitions

process is always called from the wrapper process, which is invoked by the user. A detailed description of the *TT* is given in Section 3.

The *Service Manager (SM)* handles the classification and dynamic selection of services. The *SM* builds the service type namespace by matching equivalent services that can substitute for each other. A service is mapped to exactly one service type. All services of the same service type must offer the same interface. Our current *SM* implementation assumes the service type information to be specified by the service provider. The services are classified as stateful or stateless based on information from the service provider. By default, a service is considered stateful. Services are selected according to different ranking criteria implemented by *service selectors*. Service selectors are further explained in Section 5.

<sup>4</sup> <http://www.w3.org/Submission/ws-addressing/>

*Bind Manager* interface:

- **createPN(): PN** – Returns the PN for a new process (used by the *TT* upon process deployment).
- **createPID(PN): PID** – Returns the PID for a new instance of the process identified by the given PN; instantiates the binding policy for the process instance; allocates storage to keep track of the process instance's service mappings.
- **deletePID(PID)** – Discards PID and all associated data structures.
- **getService(ST, PID): SID** – If no service is mapped to the pair <PID, ST>, returns the SID corresponding to the service of the given ST to be bound by the process instance PID and maps the SID to the pair <PID, ST>; otherwise returns the SID corresponding to the existing mapping for the pair <PID, ST>. If no service exists, an exception is thrown.
- **getAOF(SID, PID): AOF** – Notifies the *BM* about the failure of the service SID that was invoked by the process instance PID; returns the AOF according to the binding policy; deletes the mapping for the pair <PID, ST>; adds the service SID to the set of failed services.

*Service Manager* interface:

- **classifyService(SID): ST** – Classifies the service SID and returns the corresponding ST.
- **isStateful(ST): boolean** – Returns *true* if services of the given type ST are stateful.

*Service Selector* interface:

- **selectService(ST, Set{SID}): SID** – Returns the SID of the best ranked service of type ST that is not in the given Set{SID}, or *null* if no such service is available; invoked by the *BM*, passing the current set of failed services of type ST.

**Fig. 4.** Parts of the *Bind Manager*, *Service Manager*, and *Service Selector* interfaces

The *Bind Manager* (*BM*) constitutes the interface to the process. The main component of the *BM* is a web service. All interactions of the process with the infrastructure are web service invocations, thus assuring compatibility with any BPEL engine. The *BM* has the following responsibilities:

- PN creation upon process deployment (the acronyms are defined in Fig. 3).
- PID allocation upon process start.
- Binding policy: registration, mapping to PN, and instantiation upon process start.
- Service bindings per process instance.
- Provision of the action on failure (AOF) based on the policy corresponding to the process instance.
- Keeping track of failed services. A service is added to the set of failed services upon failure and removed from the set after a configurable period of time. The set is used to exclude failed services from selection for some time.

*Binding policies* control failure recovery and choose service selectors; they are detailed in Section 5.

### 3 Transformation Tool

Fig. 2 on page 244 shows the processes resulting from the transformation that are deployed in the BPEL engine.

The transformed process contains the business logic augmented with code for dynamic service binding and for failure recovery. The code within the transformed process allows replacing a failed service without process restart (for stateless services and stateful services that fail on the first invocation).

The wrapper process has a predefined structure, replaces the original process, and contains code for automated process restart upon failure of a stateful service. If a stateful service fails on or after the second invocation from the transformed process, the transformed process is brought to the initial state (i.e., it is restarted) by the wrapper process and the service invocations are repeated using a replacement service. PIDs are created only for instances of the wrapper process.

In a first step, the *TT* registers the process to the *Bind Manager* (`getPN()`) and replaces every service used in the original process with the service type provided by the *Service Manager* (`classifyService(SID)`). In a second step, the following transformations take place in the original process (yielding the transformed process):

- Every receive activity that starts a process instance is replaced with a receive activity that takes the PID as an additional input parameter. We will refer to the replacing receive activity as *extended-receive*.
- Every reply activity is replaced with a reply activity that provides an additional parameter along with the original output parameters, the *exit-code* of the transformed process. The exit-code dictates the activity to be performed by the wrapper process; it can be either *success* or *restart-process*. **Reply(output)** in the original process becomes **reply(output, “success”)** in the transformed process.
- Every invocation of a service **output←invoke(input,ST,operation)** is replaced with a code pattern illustrated by the pseudo-code in Fig. 5. Tuples are represented as  $\langle data1, data2 \rangle$ . A reply activity ends the process instance, while a rethrow activity passes control to the fault handlers defined in the original process. If no fault handler is defined, the transformed process is terminated and the wrapper process deletes the PID before re-throwing the exception.

The wrapper process replaces the original process and copies every receive activity that starts an instance as well as the corresponding reply activity. The pseudo-code in Fig. 6 presents the code within the wrapper process for a receive-reply activity. **Receive(input)** and **reply(output)** are the BPEL activities copied from the original process, and *extended-receive* is the corresponding replacing receive activity in the transformed process.

```

retry ← true ;
while retry do
  retry ← false ;
  SID ← invoke(< ST, PID >, BindManager, getService);
  try begin
    output ← invoke(input, SID, operation);
  end
  catchAll(exception)begin
    AOF ← invoke(< SID, PID >, BindManager, getAOF);
    switch AOF do
      case "restart-process": reply(< -, "restart-process" >);
      case "throw": rethrow exception;
      case "retry": retry ← true ;
    end
  end
end
end
end

```

**Fig. 5.** Transformation of a service invocation  $\text{output} \leftarrow \text{invoke}(\text{input}, \text{ST}, \text{operation})$

```

receive(input) begin
  PID ← invoke(PN, BindManager, createPID);
  retry ← true ;
  while retry do
    retry ← false ;
    try begin
      < output, exit-code > ←
        invoke(< input, PID >, transformedProcess, extended-receive);
    end
    catchAll(exception)begin
      invoke(PID, BindManager, deletePID);
      rethrow exception ;
    end
    switch exit-code do
      case "success":
        invoke(PID, BindManager, deletePID);
        reply(output);
      case "restart-process": retry ← true ;
    end
  end
end
end
end

```

**Fig. 6.** Wrapper process for a receive-reply activity

## 4 Interactions at Execution Time

Fig. 7 illustrates the interactions between the BPEL engine and our infrastructure at runtime for an example process with two invocations of the same stateful service, both in case of service failure and service normal completion. Below we outline the interactions according to the numbered steps in Fig. 7. The BPEL engine side is represented by the wrapper process (*WP*) and the transformed process (*TP*); our infrastructure is represented by the *Bind System* (*BS*), and the two services, *S1* and *S2*, are services of the same type *st1*. The internal interactions within the *BS* are further detailed in Fig. 8 for an example binding policy. For better clarity, only the most important functionalities performed in the *BS* are presented in the diagram.



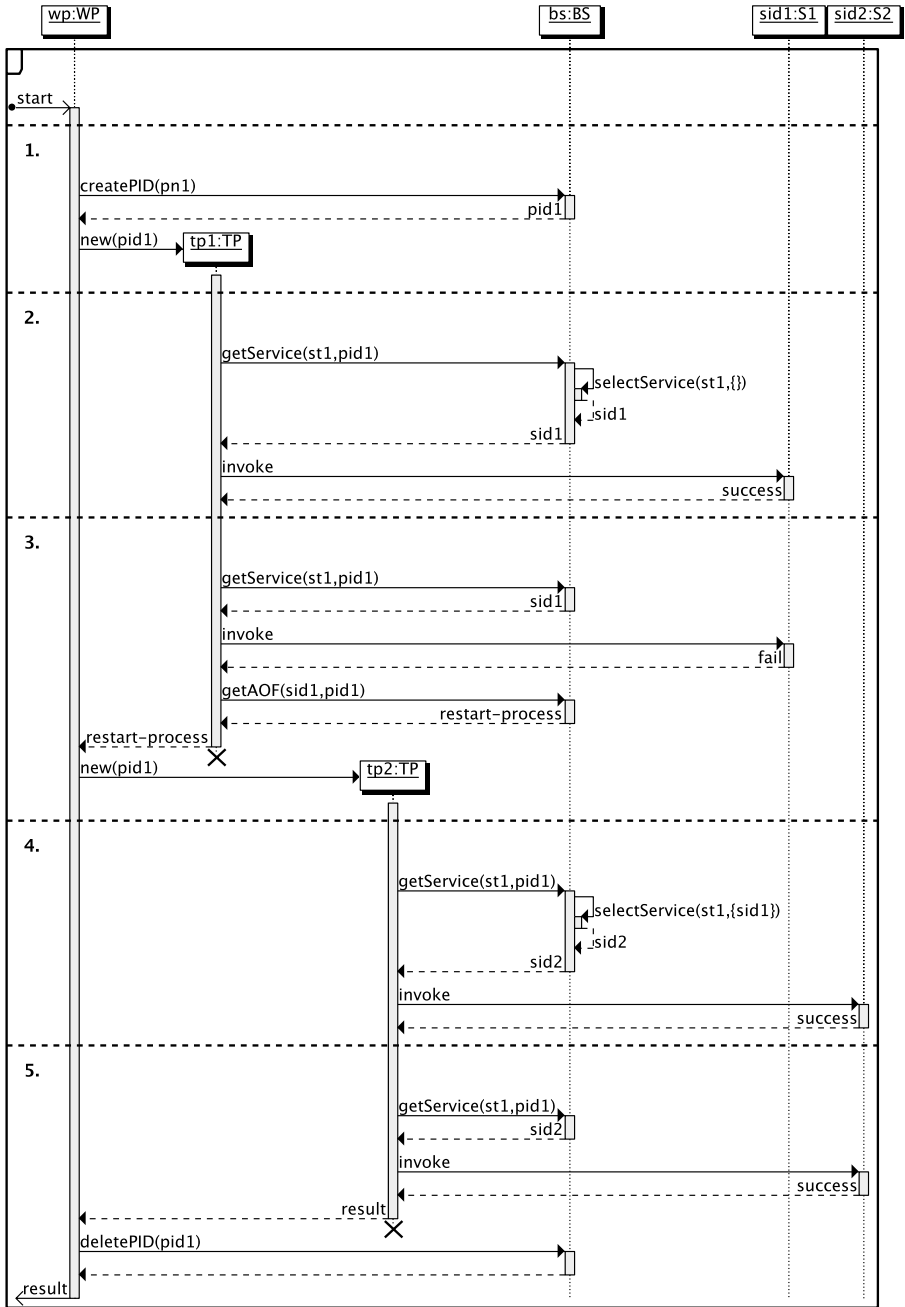


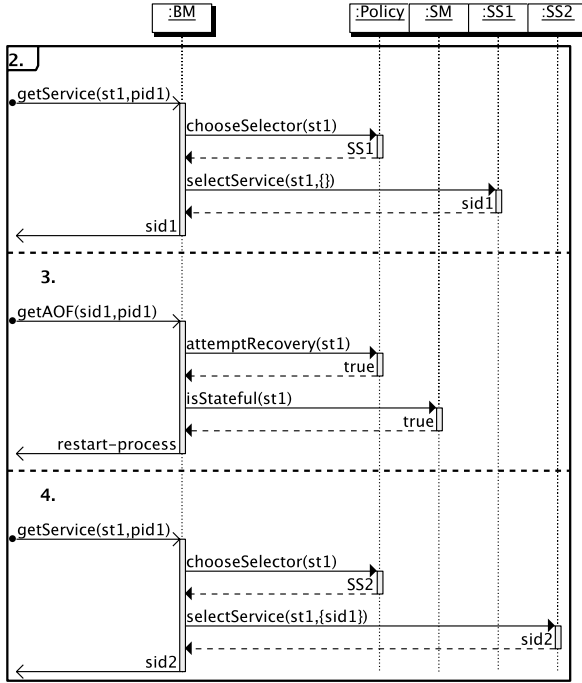
Fig. 7. Sequence diagram for stateful service invocations

1. Process request.  
When a request for the process arrives, a new instance of the *WP* is created. The *BS* provides *pid1*, allocates storage for the service bindings of the process instance, and instantiates a binding policy. The *WP* creates a *TP* instance as illustrated in Fig. 7 by `new(pid1)`. A PID is allocated only for the *WP* instance.
2. First successful invocation of a service of type *st1*.  
The *BS* checks whether there is a service mapped to the pair  $\langle pid1, st1 \rangle$  and does not find any. The *BS* selects *sid1* of type *st1* using the service selector specified by the binding policy and maps the service *sid1* to the pair  $\langle pid1, st1 \rangle$ .
3. First failed invocation of a service of type *st1*.  
The *BS* looks up the service mapped to  $\langle pid1, st1 \rangle$ . As *sid1* fails, the *BS* adds *sid1* to the set of failed services and deletes the mapping of  $\langle pid1, st1 \rangle$ . The *BS* checks the policy for *pid1* and, as *sid1* is stateful, returns AOF=*restart-process*. The *TP* instance signals the *WP* that it failed with exit-code *restart-process*. The *WP* creates a new *TP* instance.
4. First successful re-invocation of a service of type *st1*.  
The *BS* selects *sid2*, excluding from selection the services of type *st1* that are considered failing at the moment (*sid1*). The *BS* maps *sid2* to the pair  $\langle pid1, st1 \rangle$ .
5. Second successful re-invocation of a service of type *st1*.  
The *BS* retrieves *sid2* which has been mapped to  $\langle pid1, st1 \rangle$ . The *TP* instance returns the computed result to the *WP* along with exit-code *success* and completes execution. The *WP* notifies the *BS* that *pid1* has finished. The *BS* deletes all data associated with *pid1*, including the binding policy instance. The *WP* forwards the result as response to the process request.

## 5 Dynamic Customizations – Service Selectors and Binding Policies

In the following we address customizable service selectors and binding policies.

Dynamic service selection is based on ranking criteria implemented in service selectors. Service selectors are deployed in the *Service Manager (SM)* and use the service type classification provided by the *SM*. The ranking criteria implemented by a service selector typically relate to non-functional properties, such as Quality-of-Service (QoS) parameters (e.g., response time), which can be provided by monitoring mechanisms. Service selectors may use any external service discovery mechanism. Our infrastructure eases the integration of different selection mechanisms, e.g., based on QoS selection models [3], or reputation mechanisms [45]. A service selector is registered in the system under a unique identifier (**SelectorID**). The *SM* must provide at least one service selector which is used by default if no other selector is specified. The selector to be used is specified by binding policies. When a request for a service binding arrives, the *BM* checks the policy for the selector to use. A new service selector can be registered in the system at any time and can be used in updated and new policies.



**Fig. 8.** Interactions between *BM*, binding policy, *SM*, and service selectors *SS1* and *SS2*, showing details for the steps 2, 3, and 4 of Fig. 7

The binding policy has two responsibilities, it controls the action on failure (AOF) and chooses a service selector. A binding policy must provide the following interface:

- **attemptRecovery(ST): boolean** – Returns *true* if the system should attempt recovering from a recently occurred failure of a service of type *ST*.
- **chooseSelector(ST): SelectorID** – Returns the identifier of the service selector to be used. A return value *null* chooses the default selector.

Binding policies are typically stateful objects keeping track of (and limiting) the number of failure recovery attempts. A policy may choose failure recovery and service selectors depending on the service type, but it is not required to take the service type into account. According to the default policy, recovery is attempted only once, upon the first service failure, independently of the service type; hence, the process is restarted at most once.

A binding policy can be mapped to process names. When a PID is created, an instance of the policy mapped to the PN is created for that PID. In this way, policies can be changed at runtime without affecting the running instances of the process. New process instances will use the changed policy, while the running process instances will be still using their instances of the previous policy.

As an example for a binding policy, consider a system with two service selectors, *SS1* ranking services according to service response time, and *SS2* ranking services according to availability. As long as no failure has occurred, the policy chooses the selector *SS1* so as to minimize response time. However, after a failure, upon subsequent **chooseSelector(ST)** requests, the policy returns *SS2*, because the successful execution of the process is more important than the response time. Fig. 8 presents the interactions between the *BM*, the policy instance, the *SM*, and the service selectors on a first request for a service binding and on service failure.

## 6 Evaluation

In this section we evaluate the implementation of our infrastructure in two different settings. In the first setting, external services never fail, and we explore the overhead caused by our infrastructure. In the second setting, invocations of external stateful services fail with a given probability, and we investigate the resulting decrease of system throughput due to process restart upon failure.

Our implementation uses Java 5, Apache Axis 1.4, and BPEL 2.0; as BPEL engine we use ActiveBPEL 4<sup>5</sup>; both the infrastructure and the engine are deployed in an Apache Tomcat 4.1.24 installation. Our measurement machine is an Intel Core 2 Duo (2.4GHz, 2GB RAM) running Mac OS X v10.4. We avoided any unnecessary background processes. All measurements were repeated 15 times and we report the median of these measurements.

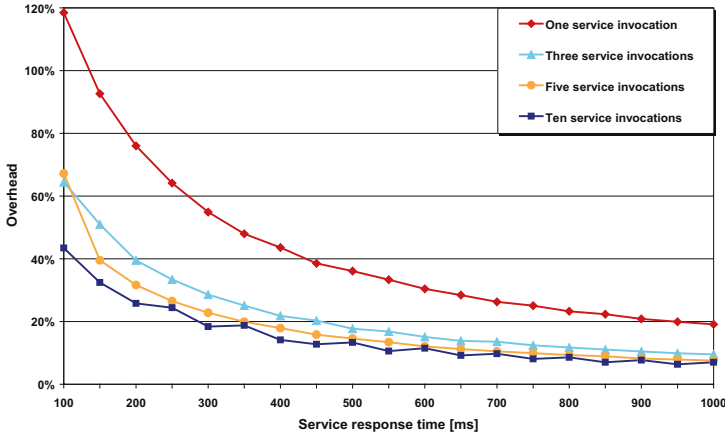
Our evaluation is based on four processes of increasing complexity. These processes interact with stateful services 1, 3, 5, resp. 10 times. In order to simulate possibly unreliable external services, service response time and failure probability are configurable.

Fig. 9 illustrates the results of our overhead evaluation. The overhead is presented relative to the execution time of the process in the absence of our infrastructure (i.e., the reference value is the execution time of the original process without any transformation). We consider the relative overhead for a varying service response time between 100ms and 1s.

The overhead shown in Fig. 9 is caused primarily by the interactions of the transformed process with the *BM*; before each invocation of an external service, the bound service is retrieved from the *BM*. In addition, the wrapper process contributes to the overhead, too, because it calls the *BM* to obtain a PID and because it has to start the transformed process. However, in the absence of service failures, the overhead induced by the wrapper process is constant. The highest overhead is observed for a process with a single service invocation, since the overhead due to the wrapper process preponderates. Moreover, the first invocation of a service of a given type causes higher workload within the *BM* because of dynamic service selection.

We assume that the *Bind System* is deployed on the same machine as the BPEL engine. Hence, interactions with the *BM* are local and relatively efficient

<sup>5</sup> <http://www.activevos.com/>



**Fig. 9.** Infrastructure overhead for 4 processes and varying service response time; services never fail

when compared with remote invocations of external services. For a service response time of 300ms, the overhead caused by our infrastructure is about 50% for a process with a single service invocation, and below 30% for more complex processes. As expected, the overhead further decreases with an increasing service response time. For a response time of 1s, the overhead is below 20% for a process with a single service invocation, and below 10% for more complex processes.

Fig. 10 illustrates the decrease in system throughput due to service failures, requiring process restart. The measurement setting corresponds to a worst-case scenario, where all services fail with a given probability and failed services are never excluded from selection. We used a binding policy that does not limit the number of recovery attempts in case of failure. Throughput was measured by considering the total elapsed time for 100 process invocations (the workload was created by 10 concurrent threads in a Java Virtual Machine, each invoking the process 10 times). Fig. 10 presents the measured throughput relative to a setting without service failure.

The impact of service failure on system throughput largely depends on the process complexity. The more complex a process is, the more work may have to be redone upon process restart. While for a simple process with a single service invocation, a service failure probability of 10% reduces the system throughput by less than 10%, the same service failure probability halves the throughput for a more complex process with 10 invocations of a stateful service. Fig. 10 shows that our approach to transparent recovery from failures of stateful services can cause a significant throughput reduction in the case of complex processes and frequent service failures. However, note that in a practical setting, a failed service would be excluded from selection for a certain period of time. Furthermore, rebinding of failed stateless services causes significantly less effort, because the process need not be restarted.

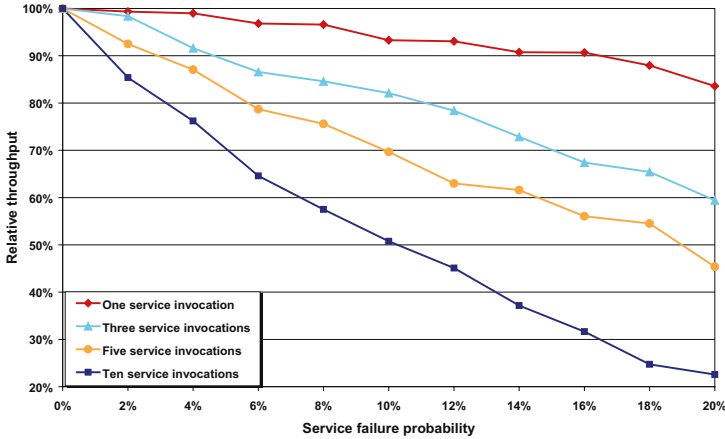


Fig. 10. Relative throughput for 4 processes and varying service failure probability

## 7 Related Work

While most existing solutions for dynamically binding services are modifying the BPEL engine [6,7], the approaches that are transparent to the engine [8] or extend the BPEL standard [9] offer limited dynamic customizability for service selection and binding policies. Our solution, in contrast, works with any BPEL engine and allows for dynamic customizability. Furthermore, solutions that also cater to fault handling deal only with stateless services [8,7], whereas our approach specifically supports stateful services.

VieDAME [6] is an aspect-based service monitoring and selection system that intercepts SOAP messages and dynamically replaces services used in the process. It monitors process execution and gathers information on the QoS of services that is stored in a database for future use. The services are selected based on defined selectors and can be adapted to replace services that implement a different service interface. The system requires that the services are registered in repository and that the services to be monitored and eventually substituted are marked as replaceable. Unlike the VieDAME system that uses an engine adapter to extend engine functionality, our infrastructure is completely transparent to the engine. Furthermore, VieDAME focuses on service monitoring and dynamic selection of services, but does not explicitly address fault handling. Our system handles failure recovery for the selected services.

Baresi et al. propose an aspect-oriented engine extension of ActiveBPEL that provides self-healing capabilities to the process and demonstrate it in their *Dynamo* system [7]. Using two domain specific languages (WScCoL, the *Web Service Constraint Language* and WSReL, the *Web Service Recovery Language*), the *Dynamo* framework allows for defining recovery strategies by specifying monitoring and recovery rules. Our infrastructure allows for the best available service to be selected for every process instance by integrating existing mechanisms for dynamic service selection based on QoS parameters.

A solution that proposes an extension of the BPEL standard to achieve dynamic binding of services on a per-instance basis is the *find-and-bind* mechanism [9]. The service to be bound is searched at runtime. This solution provides the developer with the option of deciding on the services he wants to be adaptable, but requires him to provide the fault handling constructs. Our approach separates service selection and fault handling from the process business logic.

TRAP/BPEL [8] makes use of a generic proxy to discover alternative services on failure for services that the process developer marks for monitoring. In contrast, our infrastructure is completely transparent to the user. Differently from our approach, in TRAP/BPEL all service invocations are done by the proxy, which acts as an indirection layer and allows for monitoring of the invoked services. While the TRAP/BPEL approach is similar to ours with regard to the separation of fault handling from process business logic, it does not provide support for stateful services and requires restart of the proxy on policy change.

There are other more formal approaches to exception handling in processes, such as Recovery Nets [10,11] that create models for exceptions at design time which are then used to recover from failure. Recovery Nets leverage an extended Petri net model, which incorporates customizable recovery policies to handle exceptions at runtime. Recovery Nets' aim at improving the reliability of business processes is common to ours, but Recovery Nets focus on formal exception modeling and on the development of recovery policies.

## 8 Conclusion

In this paper we have introduced a novel infrastructure for fault-tolerant execution of BPEL processes, which completely separates the process business logic from service binding and failure recovery strategies. Our infrastructure completely hides service failures from clients, addressing failures of both stateless and stateful services. It consists of two major parts, the *Bind System* to manage service bindings for process instances, and the *Transformation Tool* to enhance processes for automated failure recovery, interacting with the *Bind System*. Our infrastructure is transparent to the process developer and compatible with any standard BPEL engine. In addition, the *Bind System* enables dynamic changes of service selection and fault-handling strategies without requiring any redeployment of existing processes. Thus, our system can be tuned and reconfigured at runtime, increasing its availability. Performance evaluations have shown that our infrastructure causes only moderate overhead if external services have an average response time of 300ms or more.

Regarding limitations, our infrastructure currently does not support processes that provide asynchronous operations, because upon failure of a stateful service, the process is restarted and the reference to the operation consumer is lost. Furthermore, a failure of the *Bind System* will disrupt all transformed processes. However, the *Bind System* is not an external service (it is typically deployed on the same machine that hosts the BPEL engine); our goal is to enhance fault tolerance with respect to external services. As another limitation, our current

classification of services into service types is restrictive, because a service is mapped to exactly one type.

With respect to ongoing research, in addition to supporting more flexible service classification, we are working on process decomposition algorithms so as to restart only the affected subprocess upon service failure, therefore reducing the impact of service failure on the overall system throughput. To this end, we are applying techniques such as SESE decomposition [12].

## References

1. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: *Web Services - Concepts, Architectures and Applications. Data-Centric Systems and Applications*. Springer, Heidelberg (2004)
2. Dobson, G.: Using WS-BPEL to Implement Software Fault Tolerance for Web Services. In: *EUROMICRO 2006: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 126–133. IEEE Computer Society, Washington (2006)
3. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A QoS-Aware Selection Model for Semantic Web Services. In: *ICSOC*, pp. 390–401 (2006)
4. Bianculli, D., Jurca, R., Binder, W., Ghezzi, C., Faltings, B.: Automated dynamic maintenance of composite services based on service reputation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007. LNCS*, vol. 4749, pp. 449–455. Springer, Heidelberg (2007)
5. Jurca, R., Binder, W., Faltings, B.: Reliable QoS monitoring based on client feedback. In: *16th International World Wide Web Conference (WWW 2007)*, pp. 1003–1012. ACM, Banff (2007)
6. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for WS-BPEL. In: *WWW 2008: Proceeding of the 17th international conference on World Wide Web*, pp. 815–824. ACM, New York (2008)
7. Baresi, L., Ghezzi, C., Guinea, S.: Towards Self-healing Composition of Services. In: *Contributions to Ubiquitous Computing*, pp. 27–46. Springer, Heidelberg (2007)
8. Ezenwoye, O., Sadjadi, S.M.: TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services. In: *WEBIST-2007. International Conference on Web Information Systems and Technologies* (2007)
9. Karastoyanova, D., Houspanossian, A., Cilia, M., Leymann, F., Buchmann, A.: Extending BPEL for Run Time Adaptability. In: *EDOC 2005: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pp. 15–26. IEEE Computer Society, Washington (2005)
10. Hamadi, R., Benatallah, B., Medjahed, B.: Self-adapting recovery nets for policy-driven exception handling in business processes. *Distrib. Parallel Databases* 23(1), 1–44 (2008)
11. Hamadi, R., Benatallah, B.: Recovery Nets: Towards Self-Adaptive Workflow Systems. In: Zhou, X., Su, S., Papazoglou, M.P., Orłowska, M.E., Jeffery, K. (eds.) *WISE 2004. LNCS*, vol. 3306, pp. 439–453. Springer, Heidelberg (2004)
12. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Through SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007. LNCS*, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)



# Organizational Constraints to Realizing Business Value from Service Oriented Architectures: An Empirical Study of Financial Service Institutions

Haresh Luthria and Fethi Rabhi

Information Systems, Technology & Management, The Australian School of Business  
The University of New South Wales, Sydney, Australia  
{h.luthria, f.rabhi}@unsw.edu.au

**Abstract.** Service-oriented architectures (SOAs) are gaining popularity as an approach to provide flexibility and agility, not just in systems development but also in business process management. Studies of the practical business impacts of SOA are crucial as the number of SOA implementations grows, and are required for a better critical understanding of this popular architectural concept that is being rapidly adopted by industry organizations. Although there is a significant amount of ongoing research related to technology implementations of SOAs, there is a paucity of research literature on the factors affecting the adoption of service-oriented computing and the realization of business value in practice. This paper empirically examines the adoption of service-oriented computing (SOC) as an enterprise strategy across fifteen firms, and discusses the organizational constraints that influence the enterprise adoption and implementation of SOA. In doing so, this paper fills a crucial gap in the academic literature about the practical use of SOA as an enterprise strategy for agility, and lays the groundwork for future work on SOA alignment with organizational strategy. The paper also provides practitioners with guidelines for the successful implementation of SOA to achieve business value.

**Keywords:** Service Oriented, SOA, SOC, Business Value, Organizational Constraints, Technology Adoption, Technology Diffusion.

## 1 Introduction

In response to dynamically changing market conditions, financial institutions are increasingly looking for avenues of organizational agility [2]. By virtue of being the underlying enabler of the core business processes, information technology is very critical to achieving this agility [13]. Technology infrastructures built on service oriented computing principles appear to facilitate business process and, subsequently, organizational agility [17]. The paradigm of Service Oriented Computing (SOC) views whole business functions as modular, standards-based software services. The associated Service Oriented Architecture (SOA) establishes a defined relationship between such services offering discrete business functions and the consumers of these services, independently of the underlying technology implementation [23].

There is a great deal of enthusiasm in the industry about this concept but the adoption of SOA by end-user organizations is still in a relatively early stage [24]. Therefore, there is a scarcity of critical research addressing the ability of organizations to realize business value from the adoption of SOA. From a pragmatic perspective, there is widespread recognition of the fact that various organizational issues need to be addressed for the successful implementation of any information technology [14]. What is needed beyond the current research on the technology implementations of SOA, is a focus on the study of the real-world adoption of SOA across the enterprise and the factors that aid or impede such adoptions. This understanding becomes even more critical in the context of financial services institutions since the strategic impact of information technology is very high for financial institutions, and the industry sector is at the leading edge of the adoption curve for innovative technology solutions [20].

This paper empirically examines the use of SOA across fifteen firms – a mix of banks, insurance firms, and service providers - and as part of a broader study, specifically investigates the best practices promoted by service providers, and the organizational constraints and challenges experienced by firms considering the enterprise-wide implementation of SOA. The results provide insights into the factors that impact the real-world adoption of SOA, thus filling a crucial gap in academic literature. The paper also aligns organizational constraints with advocated best practices, thus providing practitioners with a guide to maximizing business value from their SOA implementations.

Section 2 describes the empirical study of SOA adoption across fifteen firms and the data collection and analysis processes used. Section 3 describes the observed trends in the results of the use of SOA across these firms. Section 4 distils the results of the empirical study by comparing suggested best practices for SOA adoption with the organizational challenges faced on the ground. Finally, Section 5 summarizes the research contribution and the business impact of the paper.

## **2 The Empirical Study – Data Collection and Analysis**

A case study approach was chosen as the research methodology to study the alignment and adoption of SOA across the enterprise because, according to Benbasat *et al* [3], case studies are “well-suited to capturing the knowledge of practitioners and developing theories from it”.

Fifteen firms – a mix of both financial service institutions in the banking and insurance sectors, and software service providers that had a significant number of clients in the financial services industry - were approached to understand their position on SOA. Most of these firms were chosen based on their involvement in industry conferences on SOA which was an indication of their interest in adopting SOA. A few, however, were chosen on an opportunistic basis leveraging a network of contacts. Table 1 describes the industry sector and profile of the firms interviewed as well as the designation of the interviewees.

**Table 1.** Summary of Firms Interviewed

Firm	Sector	Interviewee	Profile
1	Bank	Head of Strategy	Large Australasian private bank
2	Bank	Business development executive; Technical Architect	Large U.K. based private bank
3	Bank	Business development executive	Large Europe based private bank
4	Bank	CIO	India's second largest private bank
5	Bank	Enterprise Architect	Mid-sized Australasian public sector bank
6	Bank	Enterprise Architect	Large Australasian private bank
7	Insurance	2 x Technology manager / Architect	Mid-sized Indian private general insurance firm
8	Insurance	CTO	Large Indian public sector general insurance firm
9	Insurance	CIO	Large Australasian private insurance firm
10	Insurance	Enterprise Architect	Large Australasian public sector insurance firm
11	Product & Services	CTO; VP of Strategic Accounts	Small India-based ERP solutions firm
12	Product & Services	Technical architect	Large European ERP solutions firm
13	Product & Services	Technical architect	Large U.S. based software and services firm
14	Services	2 x Technical architect; Product manager	Large India-based software services and consulting firm
15	Services	Principal	Large multi-national consulting firm

Semi-structured interviews were conducted with business managers, enterprise architects, and CIOs/CTOs of 13 (thirteen) of these firms. A broad set of questions addressing specific areas of discussion (implementation details, challenges and concerns, benefits realized, lessons learned) was used to guide the interviews. Wherever possible, the interview data was augmented by documents provided by the interviewees. Each of the individual interviews lasted an hour with the exception of the interview with Firm 5, which lasted 30 minutes.

Communications with Firms 10 and 15 were limited to electronic communication. Firm 10 indicated that their firm did not have an explicit SOA strategy, but they were pursuing SOA practices at a technical level by "following reasonable SOA practices in terms of trying to keep things abstracted through the use of messaging middleware and a messaging portal". Firm 15 was able to supply documents describing its SOA strategy at the business and technical levels, and provide specifics of a case study of a

large financial services firm. Firms 10 and 15 are both included in the analysis not as primary data but more as an emphasis to the findings from the data gathered in the interviews with the other firms.

Fourteen of the firms interviewed were in various stages of implementing SOA, most of them already having migrated targeted business functions to a service based deployment. The firms were able to provide some insight into the anticipated and observed benefits of the migration to a service-oriented approach. Firm 6 did not have an SOA strategy and had tried unsuccessfully to migrate to a service based infrastructure. The interview provided a valuable insight into the challenges of building a business case for SOA adoption. The product and software service providers (Firms 11-15) were able to provide an insight not only into the business drivers for their product offerings but also their perception of the business drivers for their clients.

Transcriptions of the individual interview data were analyzed using a two-pass method. The first pass of the analysis used thematic coding to identify broad categories of organizational issues. The second pass of analysis was performed using axial coding and major factors were identified using meta-codes. The meta-codes were then used to identify similar patterns across the data from the multiple firms interviewed. The following section details the results of the data analysis identifying the major themes of suggested best practices and organizational factors affecting the implementation, and the cross-firm patterns observed within these themes.

### **3 The Empirical Study - Results**

Despite the many potential benefits of information technology innovations, organizations have generally found it very difficult to achieve the promised benefits [43], and the successful implementation of SOA appears to have its fair share of challenges. It is interesting to note that even in the limited scholarly work on the use of SOA in the business domain, an empirical study of two European banks indicated that the “[business] service concept was difficult to define in practice” [1].

We were able to get a significant understanding of the constraints impacting the organizational adoption of SOA running the gamut from funding at the corporate level to performance challenges at the implementation level. These constraints were compared with the best practices suggested by service providers to get an understanding of how firms were actually implementing these practices. The best practices are discussed in Section 3.1, and the constraints that impact each best practice are detailed in the following sections (Sections 3.2 – 3.8) and summarized in Table 2 at the end of this section.

#### **3.1 Best Practices Suggested by Service Providers**

Service providers, Firms 11-15, were able to provide us an insight into what lessons they thought they had learned from their experiences, and more importantly, mistakes, and what they would consider best practices for a successful enterprise SOA implementation. These experiences were collectively analyzed and distilled by thematic

categorization into a set of proposed best-practices for successful enterprise-level SOA adoption. These best practices are summarized below:

1. Get commitment at the board level.
2. Manage expectations - Invest in SOA for the long term.
3. Align the entire organization along the SOA strategy.
4. Change the mindset of people – SOA is not about technology, it is about transforming the business process.
5. Governance is critical.
6. Focus on training.
7. Leverage existing resources.

SOA is all pervasive, according to the Development Architect of Firm 12, and you need to get commitment at the board level if the adoption of SOA at the enterprise level is to be successful. Along with commitment at the board level, the expectations of people across the organization need to be managed. There needs to be an acknowledgement that the governance and training associated with service orientation could possibly end up costing much more than the development of the services.

SOA should be viewed as an evolving process and not a silver bullet. There needs to be a clear understanding at the senior management level that SOA is a long-term investment in time and resources, according to Firms 12 and 14. Success is more probable when starting out with a small project to show business value before rolling out service orientation on a larger scale, according to Firm 15.

The Development Architect of Firm 12 indicated that the commitment at the board level needs to permeate the organization to ensure that the entire organization is aligned with the SOA strategy. Organizations need to get “everyone on the same page on the SOA strategy”. The theme underlining this enterprise wide alignment needs to be that SOA is not about the technology, it is “a way of thinking”. This requires that people’s mindset needed to be changed to ensure that the focus of SOA adoption should be on transforming business processes (Firms 13 and 15).

A majority of the effort in implementing SOA appears to be implementing an appropriate governance framework in place (Firm 12), and appropriate training (Firms 12-14). Business people need to be trained in Business Process Modeling and technical people in the business aspects of services in addition to the appropriate technical training on the use of tools (Firm 14). Firm 15 indicated the need to be cognizant of when SOA may not be appropriate. Firms 12, 13, and 15 emphasized that both business and technology teams need to understand that SOA is not about creating new functionality but leverage existing resources more effectively.

These empirically gleaned best practices, it may be argued, are fairly sound generic software adoption guidelines and match up with what existing analytical SOA-related literature suggests. So what does this really mean in the real world? The input from individual firms as well as service providers’ experiences with client implementations was analyzed independent of suggested practices to better understand how the SOA implementations actually fared with the rollout of SOA across the enterprise. The following sections describe this analysis in detail.

### 3.2 Get Commitment at the Board Level

At the organizational level, there appears to be no direct business case for SOA, as indicated by Firms 1, 4, 6, 7, 9, and 10. The general approach to SOA was captured pithily by the CIO of Firm 9, who said “Using SOA increases IT value...[but] we are implementing SOA by stealth. We have no business case for SOA.”

The data also suggests that at the very high business level or customer facing level, the enterprise offerings are treated as service offerings by business units of most firms, including Firms 2, 3, 4, 8, and 9, with the business executive at Firm 3 using the term services interchangeably with applications. Business units view their offerings as a set of services, according to Firm 3, and so cannot understand why the IT teams are not already service oriented. At a business process level, however, there is no service oriented thinking and it is left to the technology teams to push service thinking up from the technology infrastructure implementing the business processes.

### 3.3 Manage Expectations – Invest in SOA for the Long Term

Funding was an issue for Firms 2, 6, and 9, since SOA needs a significant investment and business users are not willing to spend money for something intangible that may only be achieved in a few years time. The move to SOA requires significant investment in time and resources for longer-term benefits, but the technology teams in Firms 2, 4, and 9 were faced with the difficulty of defining what the return-on-investment (ROI) would be for SOA. According to the VP of Technology of Firm 2, “business deadlines do not change. How do you convince business units to spend money and time? Business units want it now. They don’t want to spend money for something three years down the road. We had to couch [the SOA implementation] in some other terms like infrastructure updates”.

From an implementation perspective, model based development was critical to the success of SOA implementations according to the service providers we spoke with, specifically Firms 12 and 13. However, there are no mature tools to either directly orchestrate business services to create an application or to translate the business process models to technical services as an intermediate step. Firm 12 uses models to elicit business requirements, but manually maps them to technical infrastructure requirements. According to Firm 14, which had unsuccessfully tried model based development on client projects, “Changes in the business process are hard to reflect in technical services. So [you end up with] two flows – business process modeling and technical process modeling. The mapping is a manual effort. Ideally we would have liked to model business processes, to technical services, to implementation. But changes cycling back cause problems because [the available] tools don’t support this”. The technical architect at Firm 2, relating a similar experience regarding their attempted BPM adoption effort said “BPEL has inherent problems. The [software] vendor says it all works – the model translates to the system. The reality is that the business unit can model its process and simulate BPR scenarios, but once the development is done, if you implement any technical change then things fall apart. BPEL has a round trip problem as a language in supporting this real-world development process. So now we define a business process model, and then manually translate it to Business Requirements and Use Cases. The business analyst draws up the business

process. The techie looks at the diagram and imports into a BPM tool. Modifications even at the process level were a problem. The processes were modeled mostly on paper – and then translated to BPM the tool. It served as documentation for future maintenance but using the models is a challenge.”

Although major software vendors have rallied around the concept of service-oriented computing [24], there still is no single unified view of the basic communications standards involved across the board [15, 19]. So despite the integration of systems, both internal and external, being a business driver for the adoption of SOA, integration across domains continues to be a challenge that goes beyond service thinking. While executives at Firms 1 and 4 were skeptical of the promise of plug-and-play, Firms 2 and 6 were experiencing problems integrating external systems because of differing standards. Firm 6, in facing the integration effort after an acquisition, found that even adopting a standard like IFX<sup>1</sup> still did not help since the contextual or semantic relevance of the data varied across the two systems being integrated. Service providers like Firms 12 and 14 also indicated that the kinds of integrations they were seeing with clients were all one-off or custom integrations. This makes it harder to sell SOA as a plug-and-play infrastructure.

Standards notwithstanding, by virtue of crossing administrative domains with potential loss of visibility and control, the new cross-organizational business models put an increased emphasis on non-functional business requirements (generally referred to as service quality attributes) such as performance, reliability, transactional integrity, and security [21, 23, 28, 29]. Additionally, with the increasing number of interfaces in a typical inter- and intra-enterprise service-oriented implementation, addressing these systemic issues in an environment of multiple administrative domains, straightforward in theory, becomes a complex problem in practice spanning technical and business arenas [19, 26]. In practice, many of the firms we spoke with were struggling with the same issues, security and performance, being of the highest concern. Firm 4 found that security could grow to be a challenge not just across external domains because, according to the CIO, a “loose confederation of services creates access and security issues”, but also across internal domains. Reuse of services by different users or higher-order services could potentially compromise security as well by changing the context in which the service is used. While Firms 2, 4, 5, 7, and 11 had experienced issues with performance of services because of fine grained services across networks, Firm 14 was grappling with problems of maintaining transactional integrity across services. What was interesting was that all of these issues were approached from a perspective of granularity, with the firms eschewing too fine grained services for performance, security, and transactional integrity, resulting in a trade-off between granularity and service quality attributes.

The use of services also involves a trade-off between granularity and the potential for reuse according to Firms 4, 5, 7, 9, and 11, since the coarser the service, the less the chance to reuse the service in different business contexts or domains. Firm 5, which appeared to have a significant and relatively successful implementation of service-oriented systems across the enterprise compared to the other firms we interviewed, was strongly concerned about the proliferation of services due to the lack of reuse. Firm 7 initiated a service-oriented infrastructure project to address the business

---

<sup>1</sup> <http://www.ifxforum.org>

requirement for a consistent user experience (“360 degree view of [the] customer”) across their three main applications dealing with health insurance, travel insurance, and automobile insurance. The major thrust of this project, called Unified Customer View, was viewed as “a de-duplication effort across all the three systems”. Each system had its own web service to retrieve customer data from a back-end transactional systems/applications. So the data had to be retrieved from all three systems and merged to get a complete picture. This removal of redundancy worked well for consistency in user experience, but caused a performance bottleneck since it was a single service addressing queries from three distinct customer bases. As a compromise, the functionality was then split across the single new service and the three older application specific services. Basic customer details were managed via the higher level service and the individual services were invoked only if further customer details were required.

### **3.4 Align the Organization Around the SOA Strategy**

One of the biggest challenges in SOA adoption is understanding which business functions can actually be viewed as business services, and how this set of granular services can be used to create a service framework [9]. The business units of Firms 2, 6, 7, 9, and 10 did not understand SOA and were more focused, understandably, on business requirements being met with “consistency, reliability, and uptime”. The VP of Engineering for Firm 2 found that the organizational challenge was clearly articulating to the business what SOA could help achieve. “The big problem in my personal opinion is that if something is so great and you cannot explain it – is it really do-able? Vendors say it is a silver bullet but cannot explain SOA in the same way to business owners, techies, and others across organization. People are trying very hard to define SOA, both vendors and end-users, rather than addressing business problems and realizing business value. You need to sell it to fund it and so people are getting lost in defining SOA”.

At the technical infrastructure level, the technology teams are actively adopting service-oriented practices for the most part independently of the business units.

### **3.5 Change the Mindset from Technology to a Business Process Focus**

Despite the service mind-set of the business units, business process modeling as a strategy was not adopted by the business units as evidenced by Firms 2 and 6. The translation of the business service requirements to a set of technical services and flows is done by the technology teams using a variety of modeling techniques. In most firms, according to Firms 2, 5, 6, and 12-14, the business services are modeled by the technology teams or representatives of the technology teams, and if needed manually mapped to the technical service architecture. This is consistent with the one other empirical study of the adoption of SOA [1] which found that business processes are not included in the service definitions, which tend to be technical service implementations of the business process flow.



Table 2. Summary of Observed Constraints to Enterprise SOA Adoption

Best Practice	Constraint Faced	Firms	Typical Quote(s)
Get commitment at the board level.	No business case for SOA	1, 2, 4, 6, 7, 9, 10	We are implementing SOA by stealth. We have no business case for SOA.
Manage expectations - invest in SOA for the long-term.	Businesses view themselves as being organized as services already	2, 3, 4, 8, 9	Each business channel or service has a development team. There is an infrastructure team supporting these services and an operations team.
	Funding for SOA	2, 6, 9	Funding is a big issue. There is no perceived business value [of SOA]. What IT is trying to do and what the software can accomplish is not understood.
	Lack of supporting tools	2, 12, 13, 14	Vendor tools are complex. Hard to learn and use. Reality is not like the hype for the tool.
	Business mapping to technical services is a manual effort.	2, 5, 6, 12, 13, 14	Changes in the business process are hard to reflect in technical services. So two flows are maintained – business process modeling and technical process modeling. The mapping is a manual effort.
	Lack of standards	1, 2, 4, 6, 12, 14	[There are] too many standards – not really a Web Services stack but a mix of technologies and standards. The core is okay (SOAP, WSDL) but standards are bad. There are lots of politics in standards with the vendors.
Trade-off between service granularity and quality attributes	Trade-off between service granularity and quality attributes	2, 4, 5, 7, 9, 11, 14	Now things are okay, reuse is not very high. If reuse increases then resources hosting these assets will become hotspots. It becomes a choice between performance oriented architecture versus SOA – and the granularity you choose determines where you land.
	Trade-off between granularity and reuse	4, 5, 7, 9, 11	Reuse is a double-edged sword. We cannot reuse changing software modules.
	Business understanding of SOA	2, 4, 7, 9, 10	I doubt whether anyone in the business would even understand what "SOA" is about, or get excited when you mention "Architecture" although they would be able to relate services-oriented model.
Align the organization along the SOA strategy.	SOA implemented within technology domain only	2, 3, 4, 8, 9	The business community has no understanding of SOA. They are looking for consistency, reliability, and uptime.
	Overship of services	2, 4, 9, 14	It is hard enough to identify the business owner of an account or application. Imagine doing that for a business service!
Change the mindset	Business processes modeled by technology teams.	2, 5, 6, 12, 13, 14	The business analyst draws up the business process. The techie looks at the diagram and imports into a BPM tool.
	Knowledge management	2, 4, 5, 12, 14	We will always need the person who created the services.
Governance is critical	Proliferation of redundant services	5, 9, 11	Our development team is not crazy about reuse.
Focus on training	Availability of skills and training	1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14	Local skill sets have not absorbed the nuances of SOA, but have been quick to absorb the hype.
	Confusion around use of legacy resources	1, 2, 4, 6, 7, 8, 12	SOA helps legacy integration. Business benefits were harvested using messaging benefits. It is difficult to move away from legacy systems. To move to what [vendor name] calls SOA is too far too quick.

An added potential complication was identified by Firm 14 which found that “breaking application into services causes ownership issues”. If a service is created for a business unit and can be used by other units, there is considerable debate of how the service needs to be maintained going forward. Will the business unit that created the service (or the business need for the service), own the service or should the common IT infrastructure team own it? If the needs of a specific user changes should the infrastructure team change the common service or customize another incarnation of the service? Such issues were echoed by the CIOs of both Firm 4 and Firm 9, and the VP of Technology for Firm 2. According to the CIO of Firm 4, identifying business ownership is critical and needed for compliance to industry regulations. “It is hard enough to identify the business owner of an account or application. Imagine doing that for a business service!”

### 3.6 Governance is Critical

The focus of service governance appears to be on knowledge management achieved by “extensive documentation”, less on which functions need to be converted to services and more on how the services were going to be created and maintained (Firm 12). The IP related to services – i.e., the knowledge of how the services work and what the inter-dependencies are - was “primarily person-based” and managed via documentation, and this knowledge management of service function and impact was identified as a big concern (Firms 2, 4, 5, 12, and 14).

A critical part of governance is vetting the creation of services to encourage reuse. In practice, however, governance committees are not looking at whether a service needs to be developed but how it will be developed. This lack of reuse results in a glut of redundant services, according to Firm 5, that causes versioning and integrity issues. Firm 9 indicated that service reuse does not work for them for a couple of reasons. The first of these reasons was that some non-critical services had been reused in the past in mission critical applications causing service downtime, and hence dissatisfied customers, when the non-critical services were taken offline for maintenance. The second reason was that the developers preferred to develop something on their own rather than take the time to learn about an existing service and its interdependencies. This was attributed to potential clashes in levels of criticality, pressures of business delivery, and sheer developer propriety. In the CIO’s words, “The concern is the speed of reaction to business requirement limits the use of SOA we would like to. We have no vision of reuse. Given the need to react quickly we are concentrating only on rebuilding for use.”

### 3.7 Focus on Training

The dependency on individual people for their implicit knowledge was also tied to a concern about availability of the right skills and training. In dealing with business units and strategic partners of businesses, general SOA awareness training and the move to a service mindset was critical, according to Firms 4 and 12. According to Firm 12, 90% of the effort in implementing SOA is on governance and process engineering, so technical people are not too keen on implementing SOA at an enterprise level. Training business people in using business process modeling tools, was also

found to be a challenge, according to Firm 2, and so the responsibility of process modeling falls on the technology teams. This requires technical staff to have the requisite “implementation and practical skills”. This requires appropriate training for technical people on the business aspect of services and the appropriate technical training on the use of tools. Even at a purely technical level, the sheer complexity of the vendor tools available for the development and management of services required extensive training according to Firm 2, which had unsuccessfully attempted to roll out business process modeling across the enterprise six years earlier. Thus, adopting a service mindset and grooming service-savvy talent on both the business and technical sides was a big concern universally for Firms 1, 2, 4-10, and 12-14.

### 3.8 Leverage Existing Resources

Our cross-firm data also appeared to highlight a lack of understanding of how legacy systems could be leveraged in a service-oriented environment. Firms 4 and 12 strongly advocated finding ways to expose existing legacy resources as services instead of building new functionality. Firm 12 was able to cite the example of a client bank that had taken three years to build a new service based loan system and “scrapped its old loan system...[that] had been in use for 12 years”. The lesson learned was that key business functions provided by the existing and proven legacy system should have been exposed as services over time one by one, thus providing a safer and less expensive migration path to a service-oriented infrastructure. Firm 14 also indicated that many firms were attempting to leverage their legacy systems by replacing their Enterprise Application Integration (EAI) frameworks by an Enterprise Service Bus (ESB) and plugging legacy systems into the ESB with service wrappers. Firm 2 was grappling with how to use legacy assets – assessing ways to do this ranging from lightweight approaches like wrappers to attempting to create a full-blown service component architecture which would result in a more complex implementation. Firm 7 was converting its legacy base iteratively to Web Services while Firm 6 could not move away from its legacy systems because the abstraction was complex. Across the firms interviewed, there appeared to be no clear or generally accepted way to leverage proven legacy systems.

## 4 Related Work

There are a few existing studies that empirically investigate the real-world implementations of SOA. Among the studies that look at organizational impacts, two vendor-sponsored studies provide insights from client engagements. The first one by Fricko [10], identifies the importance of addressing the organizational culture, specifically with respect to business and technology team interaction, and reuse in the context of a specific IBM project. The second study by Bieberstein *et al* [4] prescribes a spectrum of guidelines from organizational structure to technology architecture based on the vendors solutions.

From an adoption perspective, Yoon and Carter [30] use publicly available secondary data to study the diffusion of SOA in organizations, while Ciganek *et al* [7, 8], in

a study more closely related to the issues being considered in this paper, examine the challenges of adopting Web services across four financial service firms.

The main focus of most other studies is on the value of SOA as an integration strategy. Baskerville *et al* [1] focus on the strategic benefits achieved from implementing SOA through the lens of the four architectural challenges faced by banks – internal application integration, integration with partners, integration in the context of mergers and acquisitions (M&As), and agile development. The use of SOA as an integration framework in the context of M&As is also examined by Henningson *et al* [12]. The study reviews five companies across industry sectors, and concludes that SOA can be used to effectively integrate disparate systems. Another study by Legner and Heutshi [16] also examines the use of SOA as an integration mechanism in four firms across industry sectors. While the main thrust of the results is the design of SOA for effective systems integration, a by-product of the analysis is a suggested set of three major activities for SOA adoption – (i) introduction of new organizational roles and processes for governance, (ii) creation of architectural guidelines, and (iii) the use of SOA for infrastructure projects.

## 5 Contribution and Business Impact

Research in the area of information technology diffusion indicates that the successful adoption of new technology requires organizations to take an integrated approach to organizational and technical changes introduced by the technology [19]. The technical aspects of SOA appear to have appropriate research efforts and guidelines [23], but there is a lack of similar structure for the examination of the pragmatic impacts of the adoption of SOA. There is a growing understanding of the organizational processes and characteristics that influence the adoption and implementation of technology [5, 14], but there is little understanding of how these organizational constraints may impact the organizational adoption of SOA [18].

Our study adds to current knowledge by reviewing organization-wide challenges to SOA adoption across multiple firms with a fairly broad representation within the financial services industry – banks and insurance firms, which researchers have identified as having high dependence on technology [13, 20] - and service providers with clients in the financial services industry. Our study also has the advantage of having a blend of company profiles to add depth to the investigation. The study looks at a mix of banks and insurance firms, service providers and client firms, large and small firms, within an industry but across various countries and across public and private sectors. Firms 2, 3, 6, 12, 13, and 14 indicated that the SOA infrastructure deployment and management characteristics were comparable across countries since the business processes were identical. Contextual variations were limited to business rules catering to local regulations. The profiles of the people interviewed vary from the Head of Corporate Strategy to the Technical Architect, providing a rounded perspective of the implementation of SOA. The analysis also includes firms which ranged from those that considered their attempts at SOA unsuccessful to others that had achieved tangible business benefits from their SOA implementation. The interview data was thematically coded to glean what challenges the firms faced in the process of implementing SOA, and our findings were fairly consistent across the firms interviewed.

Studies of the practical business impacts of SOA are crucial as the number of SOA implementations grows, and are required for a better critical understanding of this popular architectural concept that is being rapidly adopted by industry organizations. These studies could well provide frameworks, guidelines, and best practices for the effective adoption of SOA as an enterprise strategy, and more importantly what challenges to expect in trying implement these practices.

The business opportunity created by SOA revolves around the reorganization of enterprise information resources as independent, reusable services [27], moving away from viewing corporations as a building block of processes, and re-inventing the corporation to be more a collection of services focused on comparative advantage [11, 22]. The automation of these services creates a new kind of business model, facilitating an integrated process across the enterprise ecosystem to include partners, suppliers, and customers [27]. This makes it critical to have commitment across the organization starting at the board level of the firms.

The evolution to the service paradigm is equally a business and IT transformation [27], and the key to effectively deploying SOA across the enterprise, is to recognize that it is an architecture that transcends technologies and could actually be independent of the underlying technologies that implement it [6]. Not many business people, however, are familiar with the term 'SOA', and many firms whose SOA implementations have fallen well short of expectations possibly did not include the business aspects of the move to a service-based deployment [25]. This risk may be mitigated by training both business and systems people to understand this new model, getting the organization aligned along this model, and changing the mindset of the organization to not only work differently but also leverage existing legacy systems.

Even as SOA is now widely recognized as having the potential to improve the responsiveness of both business and IT organizations, it seems that most organizations that are adopting SOA do not fully understand the business potential of SOA, focusing on technical implementation issues instead of the broader business service view [27]. In this context it is imperative to bear in mind the need for skills training, appropriate governance controls while being cognizant of the slowly maturing technology environment to support the migration to service orientation.

The understanding of how SOA is actually implemented on the ground, the associated implementation issues, the true business value realized, and best practices learned are all areas in which the academic and practitioner literature could be enhanced. In investigating the issues impacting a services implementation, this study

- (i) fills a crucial knowledge gap because there is little empirical evidence of the practical use of SOA across the enterprise,
- (ii) provides direction for future research, and
- (iii) provides a set of guidelines to help practitioners implement SOA successfully across the enterprise.

The findings of this study are part of a larger research effort to leverage the data from the fifteen firms to understand how the enterprise SOA strategy can be aligned with the organizational strategy. The next phase in this research effort involves a continued analysis of the data to develop a framework for SOA implementations.

## Acknowledgements

The generous scholarship provided by the DEST-funded project ADAGE at UNSW is gratefully acknowledged.

## References

- [1] Baskerville, R., et al.: Extensible Architectures: The Strategic Value of Service-Oriented Architecture in Banking. In: Thirteenth European Conference on Information Systems, Regensburg, Germany (2005)
- [2] Beidleman, C., Ray, M.: The agility revolution. In: Cortada, J.W., Woods, J.A. (eds.) *The Quality Yearbook 1998* (1998)
- [3] Benbasat, I., Goldstein, D.K., Mead, M.: The Case Research Strategy in Studies of Information Systems. *MIS Quarterly* 11(3), 369–386 (1987)
- [4] Bieberstein, N., et al.: Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal* 44(4), 691–708 (2005)
- [5] Broadbent, M., Weill, P.: Improving business and information strategy alignment: learning from the banking industry. *IBM Systems Journal* 32(1), 162–179 (1993)
- [6] Channabasavaiah, K., Holley, K., Tuggle, E.M.J.: Migrating to a service-oriented architecture. In: On demand operating environment solutions, IBM (2004)
- [7] Ciganek, A.P., Haines, M.N., Haseman, W.: Horizontal and Vertical Factors Influencing the Adoption of Web Services. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences HICSS 2006, p. 6 (2006)
- [8] Ciganek, A.P., Haines, M.N., Haseman, W.D.: Challenges of Adopting Web Services: Experiences from the Financial Industry. In: Proceedings of the 38th Hawaii International Conference on System Sciences (2005)
- [9] Erlanger, L.: Making SOA Work. In: *InfoWorld*, pp. 45–52 (2005)
- [10] Fricko, A.: SOAs Require Culture Change and Service Reuse. In: *Business Communications Review*, pp. 58–64 (2006)
- [11] Hagel, J.I., Brown, J.S.: Your Next IT Strategy. *Harvard Business Review*, 105–113 (2001)
- [12] Henningsson, S., Svensson, C., Vallen, L.: Mastering the integration chaos following frequent M&As: IS Integration with SOA Technology. In: Hawaii International Conference on System Sciences 2007, IEEE Computer Society, Big Island (2007)
- [13] Jarvenpaa, S.L., Ives, B.: Information technology and corporate strategy: a view from the top. *Information Systems Research* 1(4), 351–376 (1990)
- [14] Lai, V.S., Guynes, J.L.: An assessment of the influence of organizational characteristics on information technology adoption decision: a discriminative approach. *IEEE Transactions on Engineering Management* 44(2), 146–157 (1997)
- [15] Leavitt, N.: Are Web Services Finally Ready to Deliver? In: *Computer*, pp. 14–16 (2004)
- [16] Legner, C., Heutschi, R.: SOA Adoption in Practice-Findings from Early SOA Implementations. In: Österle, H., Schelp, J., Winter, R. (eds.) Proceedings of the 15th European Conference on Information Systems, St. Gallen, Switzerland, pp. 1643–1654 (2007)
- [17] Luthria, H., Rabhi, F., Briers, M.: Investigating the Potential of Service Oriented Architectures to Realize Dynamic Capabilities. In: Asia-Pacific Service Computing Conference, The 2nd IEEE (APSCC 2007). IEEE Computer Society, Tsukuba (2007)

- [18] Luthria, H., Rabhi, F.A.: Service Oriented Computing in Practice - An Agenda for Research into the Factors Influencing the Organizational Adoption of Service Oriented Architectures. *Journal of Theoretical and Applied Electronic Commerce Research* (2008)
- [19] Margaria, T., Steffen, B.: Service Engineering: Linking Business and IT. In: *Computer (IEEE)*, p. 45 (2006)
- [20] McFarlan, F.W.: Information technology changes the way you compete. *Harvard Business Review* 62(3), 98–103 (1984)
- [21] Mukhi, N.K., Konuru, R., Curbera, F.: Cooperative Middleware Specialization for Service Oriented Architectures. In: *International World Wide Web Conference*. ACM, New York (2004)
- [22] Murray, W.: Implications of SOA on business strategy and organizational design. *The SOA Magazine* volume (2007)
- [23] Papazoglou, M.P., et al.: Service Oriented Computing Research Roadmap. In: *Dagstuhl Seminar* (2006)
- [24] Quocirca SOA: Substance or Hype? Quocirca Ltd. (2005)
- [25] Ricadela, A.: The Dark Side of SOA. In: *Information Week*, pp. 54–58 (2006)
- [26] Saunders, S., et al.: The software quality challenges of service oriented architectures in e-commerce. *Software Quality Journal* 14(1), 65–75 (2006)
- [27] Sprott, D.: Service-Oriented Architecture: An Introduction for Managers. *CBDJ Journal* (2004)
- [28] Stantchev, V., Malek, M.: Architectural translucency in service-oriented architectures. *IEE Proceedings - Software* 153(1), 31–37 (2006)
- [29] Tsai, W.T.: Service-Oriented System Engineering: A New Paradigm. In: *IEEE Workshop on Service-Oriented System Engineering (SOSE 2005)*. IEEE, Los Alamitos (2005)
- [30] Yoon, T., Carter, P.: Investigating the Antecedents and benefits of SOA Implementation: A Multi-Case Study Approach. In: *Americas Conference on Information Systems (AM-CIS)*. AIS Electronic Library, Colorado (2007)

# E-Marketplace for Semantic Web Services

Witold Abramowicz, Konstanty Haniewicz, Monika Kaczmarek,  
and Dominik Zyskowski

Department of Information Systems, Poznan University of Economics Al.  
Niepodleglosci 10, 60-967 Poznan, Poland  
{w.abramowicz,k.haniewicz,m.kaczmarek,d.zyskowski}@kie.ae.poznan.pl

**Abstract.** Automation of processes is a crucial factor for enterprises operating within a modern collaborative business environment. In order to ensure flexible operations, companies tend to build their IT systems in accordance with the SOA paradigm and take advantage of the Semantic Web technologies. The mentioned tendency especially in case of cooperating organizations requires support for automated service discovery and fast integration of discovered artefacts. Currently, one can easily find several initiatives that aim at automation of already pointed tasks. As a part of this work, we analyze a number of different frameworks that implement, support and facilitate interactions inherent to Semantic Web services and indicate their shortcomings. Having completed this survey, we propose a general model of SWS e-marketplace taking into account all important aspects that a featured model should provide. In order to achieve this goal, a set of features provided by the surveyed frameworks is compiled with a set of additional traits that were not considered before. Moreover, the model is enriched with economical requirements driven by service providers' and service requesters' needs.

## 1 Introduction

Automation of enterprise processes is a crucial factor for enterprises operating within a modern collaborative business environment. In order to ensure flexible operations, companies tend to build their IT systems in accordance with the SOA paradigm and take advantage of the Semantic Web technologies [1]. The mentioned tendency especially in case of cooperating organizations requires support for automated service discovery and fast integration of discovered artefacts.

As the conducted research and current initiatives showed, the service-oriented computing requires an infrastructure that provides a mechanism for coordinating between service requesters and providers [2]. Such a coordination mechanism may take different forms. Various research initiatives were undertaken in order to provide a fully-fledged platform that would enable the automated interactions between service requesters and service providers. However, majority of the research concentrates on selected aspects of Semantic Web services (SWS) usage in the context of service orientation. The most active areas include service composition, discovery, description methods or contracting.



Within this work we present an abstract model of SWS e-marketplace defined based on the observations of the SOA and Web services market evolution. We argue that the problem of Semantic Web services provisioning requires incorporation of business-oriented point of view and that an additional set of mechanisms should be taken into account in order to meet business expectations. Within our approach, we do not ignore the fact that SWS are also a kind of good that is provided and requested by some entities. Therefore, we pay special attention to a way of describing clients' needs and providers' services in order to assure possibly complete information on services offered. We also highlight the problem of service quality representation.

A reference model of a marketplace is given for the sake of clarity and building common understanding of concepts later discussed. The model is defined with aid of a set of crucial concepts that were designed to be straightforward and descriptive to convey all information needed for further reference. A study of adoption of the concepts enumerated is presented.

The article is structured as follows. First, a necessary introduction into a research domain and current trends is given, based on the experience and work done by scientists involved in various European and worldwide initiatives revolving around a notion of Web service and its semantically annotated counterpart. Then, a more detailed picture of the marketplace is drawn by taking into account initiatives centred on electronic commerce in general. This is of utmost importance for the work as it is impossible to imagine a working marketplace that does not implement a number of standards and features desired and expected to be in place. Core functionalities are later enhanced with ones that seem to be omitted either due to the concrete assumptions of reviewed initiatives or due to other unknown to the authors reasons. Further on, a presentation of reference model electronic marketplace is followed by a complete list of functionalities that characterize the featured electronic marketplace is given along with discussion of its soundness. The discussion of the soundness is set as a result of a comparison of other initiatives of electronic marketplaces. The article concludes in a summary of efforts and shows the directions of our future work.

## 2 Open Service Marketplace and Research Roadmap

As pinpointed by [3] the market of Web services evolves into a direction of the open Web services market, as depicted in [1]. This vision is also supported by the research roadmap defined by European experts in the domain of service oriented computing [4] as well as current trends e.g. [1] or [41]. The mentioned vision and evolution concerns the different stages (i.e. layers) of SOA development as well as application of Semantic Web technologies to achieve automation of certain interactions.

Within the vision of extended SOA [3], three main layers may be distinguished. The first one focuses on atomic services, their descriptions and basic operations such as: publication, discovery, selection and binding; producing or utilizing their description. This layer constitutes the SOA foundation. It has already been

implemented not only for the Web services but also for Semantic Web services (e.g. DIP<sup>1</sup> or ASG<sup>2</sup>).

Upper layers provide an additional support required for service composition and service management. In the second layer, the composition of services requires an existence of additional functionalities, namely: coordination of the composite service execution, monitoring as well as conformance that are to ensure the integrity of the composite services and finally assurance of the quality of the composite service.

In the top layer, the organizations responsible for performing management functions (such as QoS assurance, overall maintenance etc.) are situated. They are called service operators, which may be service clients or composite service creators. The aim of the third layer is also to provide a support for open service marketplaces.

The authors [3] argue that the purpose of an open service market is to create opportunities for buyers and sellers to meet and make business electronically, or aggregate service supply/demand by offering added-value services and grouping the buying power. The scope of such a service marketplace would be limited only by the ability of enterprises to make the offer visible to other enterprises and establish industry-specific protocols to conduct business.

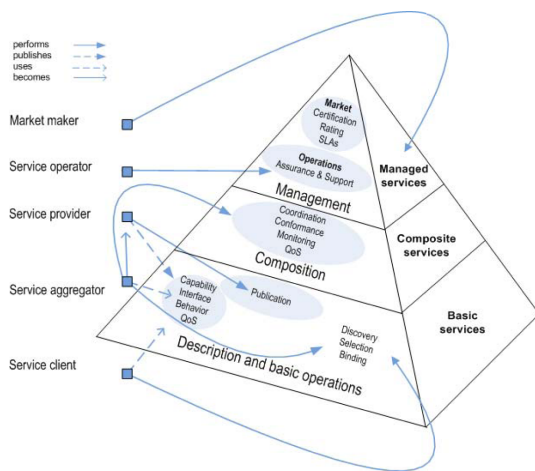


Fig. 1. Extended SOA [3]

It may be argued that we are currently situated in the third layer of SOA development, however, the question appears what form such a marketplace should take and which functionalities it should offer. As the conducted research showed, the service-oriented computing requires an infrastructure that provides a mechanism for coordinating between service requesters and providers [2]. Such a

<sup>1</sup> <http://dip.semanticweb.org>

<sup>2</sup> <http://asg-platform.org>

coordination mechanism may be implemented in various forms. Abstracting from the model that will be used, the platform where potential business partners can be discovered, prices can be ascertained and contracts may be signed is required.

The answer to the type of this platform may be found by reviewing the current initiatives and trends [1], [29]-[38] and taking into account the already mentioned research [4]. Based on current findings, we argue that the most appropriate architecture for inter-organisational collaboration should be an e-marketplace adopted to particular characteristics of SWS. Therefore, apart from typical SWS interactions like composition an additional focus is paid to industry community, business exchanges and institutional and governance aspects. The detailed description of SWS e-marketplace features is presented in the next section.

### 3 Electronic Marketplace of SWS - Requirements

Taking all of the already mentioned issues into account, it seems that the best form that the open service market may take is the semantics-based e-marketplace [17] [18] targeted at B2B interactions. As SWS are in fact the good that is may be traded on the ontology-based e-marketplace, thus, instead of a Web services e-marketplace a SWS e-marketplace should be addressed.

Bakos [8] defines an electronic marketplace as an inter-organizational information system that allows the participating buyers and sellers in some market to exchange information about process and product offerings. Other definitions stress that e-marketplaces are intermediaries that allow buyers and sellers to meet on an electronic platform that rests on the Internet infrastructure in order to exchange information about products/services (e.g. prices, specifications), conduct transactions online and adhere to other value-added services (e.g. settlement, distribution, integration, SCM) offered by the intermediary [9].

E-marketplace activity has been evolving from the early matchmaking models to more complex interactive and interconnected marketplaces. Following [10], four phases of e-marketplace evolution may be distinguished. It began with the transaction focus (the first phase) and evolved into the value-added marketplace that offers transaction support services (the second phase). In the third phase, the e-marketplace's services allow for not only information exchange but also for knowledge exchange facilitating cross-organizational collaboration. Finally, the ability to integrate the transaction exchange, the value-add services and the knowledge services moves the evolution of e-marketplaces into the fourth phase called Value Trust Networks (VTN).

In the recent years, e-marketplace proved to be sound solution to promote intra-organisational cooperation [11] [12] [13]. Moreover, collaboration-oriented e-marketplaces are cited as an emerging approach to support online business-to-business transactions [14]. As indicated in [15] the main goal of e-marketplaces in their formation phase was to bring different trading partners together. However, the requirements on e-marketplaces already increased within this phase. Companies demand additional features for lowering costs and for automation and optimization of their business processes. According to this, the aim for

e-marketplaces is to offer more automation and value add services, such as offering services for initiation, fulfilment, and completion of trading transactions including shipment, payment and logistic services [15].

The research in the area of e-marketplaces proved that the utilization of ontologies facilitates the processes of e-marketplace, from matchmaking, recommendation, to negotiation [16] and helps to achieve the desired level of automation [15]. In addition, ontology allows to solve some typical problems in e-marketplaces (see [16] for details). Taking all of the above issues in the account, it seems that the best form that the open service market may take is the ontology-based e-marketplace targeted at B2B interactions.

The Semantic Web services e-marketplace should incorporate most functionalities offered currently by online marketplaces like eBay, Amazon or auction portals [17]. Functionalities like discovery, personalization, payment, delivery, shipment tracking [19] are the must-have items on the functionality list of every modern web-based marketplace. The minimal set of functionalities that the SWS e-marketplace should support is as follows [14] :

- dynamic discovery of services and business processes,
- generation of reusable services and business processes,
- registering and advertising of available services and business processes in a proper structure.

However, we argue that the e-marketplace of Semantic Web services should possess also some additional mechanisms to meet specific requirements of the clients and providers. According to [32], the platform should allow multiple buyers and sellers to trade simultaneously and ensure an immediate reaction in case a suitable counterpart is found. The mechanism should support trading of heterogeneous services. In fact, a meaningful matchmaking of orders should be realized by the market infrastructure to allow matching of services based on the semantics of an order instead of their syntactical representation. Furthermore, services may differ in their quality characteristics and their policies, e.g. a stock quote service by its quote time; a billing service by its age restriction. As such, the mechanism should support services with many attributes.

In order to meet these requirements, an e-marketplace of Semantic Web services needs to provide a unified view of services, standard business terminology and detailed descriptions of composite services. It is important to assure that clients and service providers use the same vocabulary and language to describe offered or requested services so to assure that the appropriate domain ontology is used to annotate the selected by the e-marketplaces semantic representation of a Web service. The market maker should secure the management and maintenance of underlying ontologies as they provide the right means for specifying such semantics by featuring logic-based representation languages [32]. In order to allow meaningful matchmaking communication with the market has to take place on a semantic level. The usage of ontologies requires an introduction of reasoning functionalities to the e-marketplace. However, the mechanisms offered should be still tailored to both human user as well as machine to machine interactions as even while creating the e-marketplace for Semantic Web services,

one has to remember to find an appropriate balance between computer automation and manual involvement. One cannot focus only and solely on automating transactions as we cannot forget that although the SWS are making their way, it is still a human that takes the last decision.

Another problem is associated with different levels of granularity of service description (coarse grained business services versus fine grained objects). We need to take into account that various service providers will describe their services (using of course the ontology provided by the e-marketplace) on the different level of abstraction. For one service provider, for example, the right level of abstraction will be providing their service as e.g. payment service (that implicitly will have such functionalities as customer verification, credit card number verification etc.), whereas other service providers will provide only atomic functionalities as customer verification, others may provide such functionalities as taking input from customer. The mechanism on the e-marketplace should be able to deal with this problem.

Discovery is the key functionality required in every e-marketplace. Clients must have a tool or mechanism that helps them find a service (or their composition) according to the specification provided. Useful solution here is to categorize the services and/or to perform a clustering over them [21]. When it comes to traditional technical solutions the most popular are these using keyword matching algorithms and other text processing techniques. However, if we will consider the SWS e-marketplace, the algorithms need to operate on the semantic representation of a Web service. Therefore, the appropriate ontology-based SWS discovery mechanisms were developed [20] and are successfully utilized in various scenarios.

Personalization causes or tries to achieve the situation that every customer of the marketplace is treated uniquely. This mechanism, in its simplest form, is based on the system of user accounts. Such account keeps information about user, his personal data, interests, and other related to goods offered on the marketplace. It is very important that this data is secure. Issues with personal data can destroy trust which is later very hard to regain. The majority of marketplaces takes advantage of the system of user accounts. When user logs in, the personalized website opens for him, with tailored content as specified already in the user profile. To achieve the personalization the SWS filtering system needs to be implemented on the e-marketplace. Its main aim is to gather the customer profiles (either the human user or acting on his behalf software agent) and using semantics-aware algorithms to filter the incoming stream of new SWS in order to find the relevant ones. The Semantic Web services clustering and filtering [22] are very helpful mechanism during the personalization of the offer as only services fulfilling user needs are then presented.

The issue of trust, which was until now only marginally mentioned, is in fact extremely important. Previous SOA solutions were successfully exploited internally in enterprises. These implementations were coupling only internal Web services or at most Web services only from a few trusted partners. With more trustworthy Web services, one will be more enthusiastic about using services of other parties in his own applications. Higher trust among parties effects in

raised competition among service providers. Stronger competition means larger number of offered Web services. Moreover, in electronic commerce, trust between trading partners is considered to be just as important as in offline transactions and in some respects more important because of the nature of the channel.

Appropriate feedback mechanisms built into the e-marketplace, which allow participants to publicise their experiences, could improve the levels of trust between buyers and sellers and allow to update and the SWS characteristics. An exemplary functionality that could be applied here is provided by a service profiling [23]. The profiling is used to determine the values of non-functional properties of services based on their execution history data. It allows to verify the correctness of the description provided by service providers. Moreover, as pinpointed by [24] in order to enable a true e-marketplace for services, there is a need for service clients to share their knowledge so as to help each other improve the quality of their decisions and learn from the previous interactions. In addition, e-marketplace requires ratings to attract new market opportunities and competition thus supporting dynamic real-time advertising [24].

The service composition is a mechanism that creates value added to SWS e-marketplace not only because it allows to fulfil clients' requirements (through composition of new applications), but also as new compositions become again services offered on the market. So in this way the number of available services is increasing all the time what makes the SWS e-marketplace more attractive to the users. With a use of sophisticated composition algorithms and methods [25] service chains may be created in order to fulfil complex user requirements. Carrying out the composition of SWS automatically is not a trivial issue. There are however many initiatives in this area. At first, enabling composite services has largely been an ad hoc, time-consuming and error prone process involving repetitive low-level programming. Then, the ontology-based frameworks for the automatic composition of Web services were introduced. A few different approaches/algorithms to automate service composition, most of them being an adaptation of planning algorithms, are used (e.g. ASG platform). Thus, there are still many issues that need to be investigated before the SWS composition algorithms will be taken fully advantage of by companies.

Semantic Web services e-marketplace must provide a possibility of contracting between customers and sellers. Web services are the kind of intangible good, whose main features, besides functionality, are in the performance and quality aspects. That is why the contracting infrastructure should be available on such a marketplace. Service level agreements should be defined with use of specific template, provided by the marketplace authorities/owners. The usage of formalized contracts prevents both sides (buyers and sellers) from any troubles in SLA interpretation in case of conflicts. An ideal situation would be if a marketplace was somehow connected with an infrastructure on which offered services are run. In that case it would be easy to monitor the performance of contracted services. Nevertheless, the storage of SLA instances in the marketplace repository is helpful, when we take into account that both buyers and sellers measure

the performance of contracted services. When any violation is detected they can refer to the agreement stored in the marketplace platform. The template of SLA should be constructed with regard to possible parameters to be measured, payment methods, discounts, fines, obligations and rights of both sides. The definition of such a template would be on a very general level, but to every domain of services there would be a need to create more detailed SLA templates. The contracting is heavily elaborated on in a number of publications and standards (e.g. [26] [27] and WSLA<sup>3</sup>).

The process that is an important point of every deal is terms/price negotiation. Semantic Web services are ideally designed to be negotiated. This results from the basic idea of Semantic Web, where intelligent agents can act on human's behalf. So, it is possible to equip both buyer and seller with preferences related to service delivery terms. The output of the negotiation process could be formed as an already mentioned service level agreement. An algorithm that does not give any advantage to buyer/seller implemented on the marketplace would enable the negotiation between agents. The negotiation strategies and detailed discussion on algorithms [28] are out of scope of this article.

Payment is another fundamental mechanism that every e-marketplace must have. We can divide payment services into two categories: buyer-seller based and third party based. First group of transactions causes the money flow between the buyer and seller without engagement of another body. Usually, when the agreed amount of money appears on the seller's account, the transaction is finalized. Second category covers these with third party intermediaries. Services such PayPal play a role of securers of the transactions. The latter category is more popular on international marketplaces, whose users come from many different countries. In case of SWS e-marketplace the payment mechanism is quite similar. However, it needs to be adapted to be used in the automatically carried out interactions.

Delivery of goods and deal tracking are crucial on the markets of tangible goods. They have a little bit different meaning on the market of Web services where the service may be either consumed using the Internet or the physical delivery needs to take place. When the deal is completed immediately, the thing to track or to be more specific - monitor, is the Web service execution itself (especially when it comes down to long-running Web services). On the other hand, when the delivery of a good takes more time and engages other parties (e.g. forwarding companies) it is very valuable for a customer to have the possibility of checking what is currently happening with the ordered item.

In consequence the following high level architecture of the SWS e-marketplace emerges as shown in figure 2. The main layer on which the e-marketplace is based is the adequate Semantic Web services representations along with the adequate reasoning possibility. The e-marketplace should support at least the following mechanisms that should operate on the semantics: publishing/registering, discovery and filtering, profiling, negotiation and contraction, composition, execution and monitoring, as well as trust and security and financial mechanisms.

---

<sup>3</sup> Web Service Level Agreements Project, <http://www.research.ibm.com/wsla/>

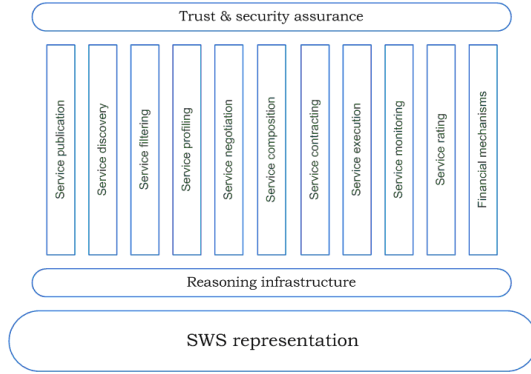


Fig. 2. Required functionalities of SWS e-marketplace

## 4 SWS E-Marketplace Reference Model

Within this section an abstract model of the SWS e-marketplace model is presented. We define SWS e-marketplace as an intermediary that allows service requesters and service providers to meet on an electronic platform that resides on the Internet infrastructure in order to exchange information about services (e.g. service description), conduct transactions online and adhere to other value-added services offered by the intermediary. All artefacts and mechanisms on this e-marketplace are semantic-enabled. Let SWSEM denote Semantic Web services e-marketplace. It may be defined as follows:

$$SWSEM = C, SP, S, M, PR, SA$$

Where:  $C$  - denotes e-marketplace clients, for  $i = 1 \dots k$ . A service client is an entity that requests a service from a service provider (by invoking the service) and eventually consumes the service (sending request data and/or receiving the results). The term service client may be exchangeable used with terms service consumer and service requester.

The discussed e-marketplace is targeted at enterprises of all shapes and sizes or for the company internal purposes (especially those virtual ones having multiple branches in multiple locations to be used by employees/departments etc.). On the SWS e-marketplace organizations are treated as business users. Business users want to streamline or enhance their business processes by using best available pieces of software (in this case - Semantic Web services). They manifest their needs in a form of client profile. Within our model we envision a presence of software agents acting on behalf of clients. Such an agent must be provided with the profile of a user, negotiation policies and constraints defining acceptable terms of service provisioning.

$SP$  - denotes service providers, for  $i = 1 \dots p$ . Service providers may be divided into two groups ASP (atomic service providers) and CSP (composite service providers). Some providers may belong to two groups as they may offer both atomic as well as composite services. One may say, that the special case of CSP



is an e-marketplace itself as it performs service composition and may offer to clients composite services. Service providers are also represented on the SWS e-marketplace using appropriate profile informing about the contact details as well as Semantic Web services they provide and their quality. Application developers are sophisticated participants of SWS e-marketplace. They are familiar with technical aspects of SWS and exactly know what they need. They are prepared to describe precisely which SWS is needed in their application. Developers need rather one-time solutions from the e-marketplace, in order to finish and run their application. However, they may update their application if a better service appears.

Brokers have more transactional power than single participants. We can distinguish brokers acting on providers' and customers' side. More popular are customers' side brokers. They may negotiate better terms with providers as they usually buy. Next, brokers may resell these services with some margin. Providers' side brokers are rare, but may appear when powerful customer emerges on the market and his requests may not be served by single providers. However, it should not be a case in the domain of SWS where only the scalability of computational infrastructure constitutes the limit.

$S$  - denotes a set of services (both atomic and composite ones, also those composed by the e-marketplace itself), for  $i = 1 \dots s$ . Atomic service is a service that does not rely on other services during execution. In case of composite services it is quite the opposite - they do rely on other services during their execution. Please note, that for the e-marketplace as well as clients there is not much difference whether it is a composite or atomic service that is offered by the service provider. Both services are black boxes and we cannot see their internal structure (the internal logic of the composition) as it is hidden to avoid being copied by other parties and losing the competitive advantage by the service provider. However, the difference lies in that the composite service may be improved by relying on different services during execution. The improvement of an atomic service requires changing the implementation of the service (new algorithms, new logic, new hardware etc.). The repository that would store the services description on both the functional as well as non-functional properties is also a part of the SWS model. On the high level it is not important which particular language is used to describe a service. However, the market maker needs to ensure that all services are described using the same ontology as well as that the domain ontologies are known to e-marketplace actors.

The services offered on SWS e-marketplace may be divided in two groups: information services and real world services. The difference between these two types lays in the nature of results obtained as a result of their execution. In case of information services a piece of information is both an effect and an output of a service execution. As an example may serve services responsible for some computation and all those that only operate on virtual objects. In case of real world services SWS are only the interface to real functionality that must be provided in a real world. So, we can use SWS that is an interface to ordering a book, but as an output we obtain a digital confirmation of a purchase, but the

effect is the real book that is shipped few days later. In our understanding of Semantic Web services we pay special attention to their description. We advocate that the description of services offered on the SWS e-marketplace must be as informative as possible and should cover aspects of their functionality as well as quality.

*M* - denotes mechanisms (functionalities offered by the marketplace) and tools provided to clients and service providers (for example: composition, selection, profiling, contracting, monitoring); for  $i = 1...m$ . They were already discussed within the previous section.

*PR* - denotes participation rules and a business model - all the rules that are to organize the provisioning, supply and demand matching process, for  $i = 1...r$ . E-marketplaces should provide their users with institutional infrastructure that encompasses issues related to contract law, dispute resolution, and intellectual property protection. These business rules must be enforced and monitored on the e-marketplace. These aspects, as is shown within the next section, are disregarded by most if not all of the current initiatives.

*SA* - supporting artefacts - all artefacts - SLAs, domain ontologies, extended OWL-S etc. This relates to the indispensable data structures, ontologies and languages used in the e-marketplace to process information about services, enable transactions and support all the functionalities offered by the e-marketplace.

## 5 Comparison of SWS Frameworks

In order to compare proposed model with current initiatives in the Web services provisioning, please take a look at the figure 3. This table presents the most important aspects and their coverage by selected (S)WS market models.

Initiative	Focus	Targeted at/Actors	Semantics	NFP and SLA	Discovery	Composition	NFP-based selection	Profiling	Execution	Character
ASG	Adaptive service provisioning	EU, C	W	Cost, duration, SLAs are considered	+	+	+/- only local level selection	+	+	Open marketplace platform/framework
WSMX [30]	Providing execution environment for discovery, mediation, composition and mediation	Any	W	Offers support for WSMO NFPs, no SLAs used	+	+	Not considered	-	+	framework
C-Cube [31]	Corporate intranets	C	DL	Generic QoS attributes, no SLAs	+	+	Not considered	-	+	marketplace
Lamparter [32]	Pricing mechanisms	BU	DO	Only price and security are considered	-	-	(Pricing and auctioning algorithms)	-	-	marketplace
Lamparter [33]	Policies, contracting. Market plays only a role of service repository.	BU	DO	Generic QoS attributes, SLA is considered	-	-	Not considered	-	-	marketplace
DIANE [34]	Matchmaking - centered	n/a	DSD	Do not distinguish between functional and non-functional service properties, no SLA is considered	+	+	+/- (It is integral part of service discovery)	-	+	framework
Dependencies [35]	based Dependencies as a modeling concept to describe service requests and service offers	n/a	OWL	Not considered	+/-	+/-	-	-	-	framework
Towards [36]	competitive Composition of services in competitive market of services	n/a	-	Not considered	-	+	-	-	-	marketplace
SWEET [37]	Creation of SWS applications	EU	W	Not considered	+	+	+	-	-	framework
METEOR-S [38]	Discovery and composition	BU	DS	Generic QoS attributes, SLA not considered	+	+	-	-	+	framework
SWS-EM	Modelling of requesters' needs, extensive information on SWS, automation of interactions	BU	OS	Expressive NFP ontology, SLA for both atomic and composite services	+	+/-	+(based on business context of requester)	-/+	-/+	formalized model of e-marketplace

Legend for columns: EU - end users, C companies, BU - Business users; W - WSMIL, DL - Description logic based formal languages; DO - DOLCE ontology; DS: DAML-S; DSD - Diane Service Descriptions; OS - OWL-S

Fig. 3. Comparison of SWS frameworks

The important conclusion is that these approaches have usually different research goals. Most of them are composition- or discovery-oriented, whereas some address more economical issues of SWS provisioning. Our SWS-EM model is more economically oriented, but we do not ignore the technical aspects. Therefore, some elements are marked +/- which means that the reference model considers them, but our research goals and detailed analysis are now aimed at missing aspects.

What is also interesting is that most approaches use their own languages for describing services. This may be the reason that they lack generality and their potential usage by wider communities is doubtful. Recent research [40] clearly shows that the most popular semantic language is OWL. Therefore, the use of OWL based service descriptions seems to be a right choice.

As also may be concluded from the table, the described functionalities of the Semantic Web services e-marketplace do not have to be implemented from scratch. The enumerated interactions along with several algorithms implementing them [39], are already applied in business scenarios [29] or DIP. However in order for the SWS marketplace to be successful lessons learned need to be taken into account. In the first generation of Web services the target user was a programmer. However, right now the target users of the e-marketplaces (and also SWS e-marketplace) are domain experts, consultants and business specialists implementing business processes through service composition [14]. The current description stack and interactions are tailored to the needs of the developers and not the business people. WS and SWS should not be perceived only as programmable components. They are also a kind of commercial commodity provided via the Internet with a remarkable trait of combining them in functional workflows fulfilling the user's need of resolving their non-trivial problems. The mechanisms of the SWS e-marketplace should take advantage of the already existing achievements in the area of SWS interactions, however all the mechanisms need to be adjusted to the new target user and extended with business aspects.

## 6 Conclusions and Future Work

The idea of the SWS e-marketplace presented in this article was developed as a result of research carried out in the area of SOA and Web services technology. Fully-fledged SWS e-marketplaces should provide a large set of service support and other functionalities, e.g.:

- providing to its participants a unified view of services, standard business terminology and detailed composite services descriptions,
- a comprehensive range of functionalities supporting trade, negotiation, financial settlements, service certification and quality assurance, rating services and service metrics and manage the negotiation and enforcement of SLAs,
- the market maker takes the responsibility of marketplace administration and performs maintenance tasks to ensure its quality and reliability.

The SWS e-marketplaces should be created with the cutting-edge information technology. However, to be truly successful, they must exceed their technological

roots and offer a support to the relationship management, personalization, one-to-one marketing as well as trust and security.

Solutions available at the moment hardly meet the enumerated postulates. Current e-marketplaces are basically simple websites storing information on available services. There is a place for marketplace with described features as it would create a brand new value for the users. The amount of features that the marketplace is to be equipped with, allows for effective use, thus creating a feedback for the new potential users. This statement is made upon the experiences of marketplaces of the class of eBay.com and community portals that share some characteristics with the marketplace.

We have to clearly state that the mechanisms of the featured framework can have some issues that can spoil all the benefits enumerated. We are aware of the obstacles and challenges that one has to tackle when creating the e-marketplace for Semantic Web services. The future work is to investigate in details the mechanisms presented in this article, elaborate on the issues of trust and security on the SWS e-marketplace and then provide a prototypical implementation of the proposed solution.

The current trends in the software domain should be closely watched. Currently we observe also a change in the way of the software distribution and provisioning. There is a shift towards the software as a service model and services and their non-functional characteristics are placed in the centre of interest. In this situation the mechanisms offered by the SWS e-marketplace will provide users with a required support on the global market of the future.

## References

1. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: Proceedings of the IEEE ICEBE 2005, Beijing, China, October 18-20, pp. 535–540 (2005)
2. Lamparter, S., Agarwal, S.: Specification of Policies for Automatic Negotiations of Web Services. In: Hendler, L.K.A.T.F.A.J. (ed.) Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, Galway, Ireland (2005)
3. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing. Introduction to the Communications of the ACM 46(10) (October 2003)
4. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing Research Roadmap. European Union Information Society Technologies (IST), Directorate D
5. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary (May 2003)
6. Bussler, C., Maedche, A., Fensel, D.: A Conceptual Architecture for Semantic Web Enabled Web services. ACM Special Interest Group on Management of Data 31(4) (December 2002)
7. Deng, S., Wu, Z., Li, Y.: ASCEND: a framework for automatic service composition and execution in dynamic environment. In: Proceedings of International Conference Systems, Man and Cybernetics, pp. 3457–3461 (2004)

8. Bakos, J.Y.: Reducing buyer search costs: Implications for electronic marketplaces. *Management Science* 43(12), 1676–1692 (1997)
9. Hadaya, P.: Determinants of the future level of use of electronic marketplaces: The case of Canadian firms. *Electronic Commerce Research* 6(2), 173–185
10. Raisch, W.D.: *The eMarketplace: Strategies for success in B2B ecommerce*. McGraw Hill, New York (2001)
11. Adams, J., Koushik, S., Vasudeva, G., Galambos, G.: *Patterns for e-business - a strategy for reuse* (2001)
12. Feldman, S.: E-business: Electronic Marketplaces. *IEEE Internet Computing*, (4) 93–95 (2000)
13. Grey, W., Olavson, T., Shi, D.: The role of e-marketplaces in relationship-based supply chains: a survey. *IBM Systems Journal* 44(1) (2005)
14. Hidalgo, A.N., Zhao, L., Falcone-Sampaio, P.R.: Leveraging e-marketplaces models for Web service-based application development. In: Pages-Casas, L. (ed.) *Web services*, vol. VII (2006)
15. Mueller, I., Braun, P., Rossak, W.: Integrating Mobile Agent Technology into an e-marketplace solution: The InterMarket Marketplace: Friedrich-Schiller-University Jena (2002)
16. Chiu, D.K.W., Poon, J.K.M., Lam, W.C., Tse, C.Y., Su, W.H.T., Poon, W.S.: How ontologies can help in an e-marketplace. In: *13th European Conference on Information Systems, Information Systems in Rapidly Changing Economy (ECIS 2005)*, Regensburg, Germany, May 26–28, 2005 (2005)
17. Lamparter, S., Schnizler, B.: Trading services in ontology-driven markets. In: *The Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1679–1683. ACM Press, New York (2006)
18. Li, Z., Zhao, H., Ramanathan, S.: Pricing web services for optimizing resource allocation - an implementation scheme. In: *Web 2003*, Seattle (2003)
19. Shmueli, O.: Architectures For Internal Web Services Deployment. In: *Proceedings of the 27th VLDB Conference*, Roma, Italy (2001)
20. Liu, C., Peng, Y., Chen, J.: Web Services Description Ontology-Based Service Discovery Model. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings) (WI 2006)*, pp. 633–636 (2006)
21. Abramowicz, W., Haniewicz, K., Kaczmarek, M., Zyskowski, D.: Architecture for Web services filtering and clustering. In: *The Proceedings of ICIW 2007*, IEEE 2007 (2007)
22. Abramowicz, W., et al.: Application-oriented Web Services Filtering. In: *The Proceedings of International Conference on Next Generation Web Services Practices*, pp. 63–68. IEEE, Los Alamitos (2005)
23. Abramowicz, W., et al.: Architecture for Service Profiling. In: Castellanos, M., Yang, J. (eds.) *Proceedings of 2006 IEEE Services Computing Workshops (SCW 2006)*, pp. 121–127. IEEE Press, Los Alamitos (2006)
24. Maximilien, E.M., Singh, M.P.: Conceptual Model of Web services Reputation. *SIGMOD Record* (2002)
25. Akkiraju, R., et al.: Combining planning with semantic matching to achieve web service composition. In: *4th International Conference on Web Services, ICWS 2006* (2006)
26. Lamparter, S., Luckner, S., Mutschler, S.: Formal Specification of Web Service Contracts for Automated Contracting and Monitoring. In: *The Proceedings of the 40th Hawaii International Conference on System Sciences*. IEEE, Los Alamitos (2007)

27. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic WS-Agreement Partner Selection. In: Proc. of the 15th Int. WWW Conf., Edinburgh, UK (2006)
28. Vivying, S.Y., Cheng, V.S.Y., Hung, P.C.K., Chiu, D.K.W.: Enabling Web Services Policy Negotiation with Privacy preserved using XACML, HICSS. In: 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), p. 33 (2007)
29. Kuroпка, D., Weske, M.: Implementing a semantic service provision platform. In: Concepts and Experiences, *Wirtschaftsinformatik*, vol. 1, pp. 16–24 (2008)
30. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: Proceedings of the IEEE International Conference on Web Services. IEEE Computer Society, Los Alamitos (2005)
31. Canfora, G., Corte, P., De Nigro, A., Desideri, D., Di Penta, M., Esposito, R., Falanga, A., Renna, G., Scognamiglio, R., Torelli, F., Villani, M., Zampognaro, P.: The C-Cube framework: developing autonomic applications through web services. *SIGSOFT Softw. Eng. Notes* 30, 1–6 (2005)
32. Lamparter, S., Schnizler, B.: Trading services in ontology-driven markets. In: Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France. ACM, New York (2005)
33. Lamparter, S.: Policy-based Contracting in Semantic Web Service Markets. Karlsruhe, Univeristy of Karlsruhe (2007)
34. Kuster, W., Koenig-Ries, B., Stern, M., Klein, M.: DIANE: an integrated approach to automated service discovery, matchmaking and composition. In: Proceedings of the 16th international conference on World Wide Web, Banff, Alberta, Canada. ACM Press, New York (2007)
35. Tolksdorf, R., Bizer, C., Heese, R.: A Web Service Market Model based on Dependencies. In: The Twelfth International World Wide Web Conference (WWW 2003) (Posters), Budapest, Hungary (2003)
36. Cheng, S., Chang, C., Zhang, L., Kim, T.-H.: Towards Competitive Web Service Market. In: Proceedings of the 11th IEEE International Workshop on Future Trends of Distributed Computing Systems. IEEE Computer Society, Los Alamitos (2007)
37. Brambilla, M., Ceri, M., Facca, F., Celino, I., Cerizza, D., Valle, E.D.: Model-driven design and development of semantic Web service applications. *ACM Trans. Interet Technol.* 8(3)
38. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Inf. Technol. and Management* 6, 17–39 (2005)
39. Abamowicz, W., et al.: Automatic Web services interactions - requirements, challenges and limits from the F-WebS system perspective. In: The Proceedings of International Conference on Next generation Web Services Practices, IEEE 2006 (2006)
40. Cardoso, J.: The Semantic Web Vision: Where Are We? *IEEE Intelligent Systems* 22, 84–88 (2005)
41. Barros, A., Dumas, M., Bruza, P.: The move to Web services ecosystem. *BPTrends* (2005)

# Business Driven SOA Customization

Pietro Mazzoleni and Biplav Srivastava

IBM T. J. Watson Research Center  
Hawthorne, USA 10532  
{pietro@us,sbiplav@in}.ibm.com

**Abstract.** Service Oriented Architecture, e.g., Web services, as building blocks for IT based on open standards, assist enterprises become more responsive to the changing business environment when they are implemented and used in the context of business processes. In this direction, packaged integration platforms like IBM's Composite Business Services or SAP have pre-configured business processes offered as web services. When the demand for a new capability arises, it can be addressed by building new services or by customizing an existing service. Service providers try to cover as much of the potential customer requirements as possible with provided capabilities but a complete coverage is not possible as individual industries might have unique requirements and customers can integrate services from multiple parties. In this situation, the problem is not whether a particular customization method will work but rather how to determine the overall impact of a new requirement in a complex SOA environment in terms of activities to be done and at what cost.

In this paper, we propose a solution to these problems by introducing the notion of business driven customization of SOA (specifically web services). We introduce a formal model capturing properties and relationships of business objects and business processes, and their implementing services and messages. We also have instance-independent, impact propagation rules to encode the desirable customization behavior of any implementation. Now, we can capture new requirements as change triggers in the model and using the modeled rules, can precisely compute the scope of their overall impact spanning both business and IT domains. Overall, we introduce the customization and impact model, describe its implementation, and illustrate its application in an industry scenario with large number of services with complex characteristics (SAP).

## 1 Introduction

Service Oriented Architecture (SOA) e.g., Web services, is the popular building blocks for open-standards based IT today. Here, service providers publish specification of their IT capabilities wrapped as services onto registries. The services can be discovered by potential clients later and then invoked on the providers, all using standardized interfaces. They can assist businesses become more responsive to the changing business environment when they are implemented and used in the context of business processes.

Packaged integration platforms like IBM's Composite Business Services<sup>[1]</sup> or SAP have pre-configured business processes expose as services. As an example, to simplify the process of adopting SOA, SAP is splitting up its application functionalities

(e.g. ERP, CRM as well as industry specific solutions) into thousands of ready-to-consume services. These services are grouped into sets (or bundles, in SAP terminology) along business scenarios (e.g. complaint management or order to cash) and they have a built-in semantics which partitions data into pre-defined changeable business processes and business objects. Similar semantic is adopted by custom-built services as well as partner-built services which may be created for functionality gaps that are not covered directly by SAP. Other vendors admit industry-standard business processes like RosettaNet, and web services are used to implement these processes. In business, there is an increasing need for service customization as many industries as well as software vendors are moving their architectures towards services.

Given that changes will continuously happen in any business that can affect their business objects and business processes, we are interested in precise methods that can characterize the impact of these changes on their services (specifically, web services) implementation. The business changes rarely impact a single service. Today, the business to IT alignment is implicit and it is impossible to precisely determine the major changes from the minor ones.

Knowing the changes to be made in a complex SOA brings several advantages. On the business side, it provides an understanding on the parts of the businesses which could be affected by the new requirement leading to valuable insights for project plan, cost of implementation, and best practices in building customizable Services. On the technical side, it supports software engineers in identifying which tasks should be implemented in the system. We also aim to provide methods that will help an organization determine guidelines about when to customize an existing service versus create new ones with richer templates so that they have a rich collection of distinguishing services that can be highly reusable. The issues we consider are complementary to how customization may be implemented, and hence, our approach will work with any method for the latter in literature [2,3,4,5]. In [6], the authors propose a method to customize Web Services published by a provider by using WS-Policy based customization points that a consumer can select. The service with the selected customizations is now created and hosted by the provider. The consumer can call the customized service and do its processing. Closely related to the notion of customization are the concepts of *personalization* and *adaptation* of web services. As discussed in detail later in Section 6, personalization deals with how to modify the content output from a web service specific to the user at run time whereas adaptation deals with how to modify the behavior of a deployed web service at runtime based on environmental considerations. A new service is neither created nor deployed for the unique service request. Our contributions are that we:

1. Introduce a business driven model for computing the scope of customization of SOA (specifically web services)
2. Present a prototype implementation, investigate the source of complexity in the model and discuss how it can be used to expose the trade-offs /issues in different types of customization.
3. Show its generality and usefulness by applying the model to a complex service scenario, i.e., SAP.



The paper is organized as follows: we start with preliminaries on service customization, discuss a motivational scenario to bring out the issues in customization and then present our model for customization. Next we discuss its implementation and apply to the complex services scenario of SAP. We round up the paper by discussing the salient features of our work, the limitations and related work.

## 2 Background and Motivating Scenario

### 2.1 Background

When a requirement for a service arises, it can be addressed by building a new service or customizing an existing one. By *customization* of web services, we mean the process by which the behavior of existing web services can be modified to meet the requester's requirements. Specifically, we consider customization as:

- Building parameterized service interfaces so that they can capture a wide range of situations (template);
- Building extensibility mechanisms in the middleware (e.g., proxy) to integrate and extend available services; to meet new requirements unanticipated by the service provider.
- Creating a new customized service instance that is deployed specifically for the unique service requester (and may later cater to others as needed). Customization is expected to be an offline activity unless the middleware supports the new customized service to be deployed on-the-fly.

While any service can be customized in theory to meet any requirement, doing it indiscriminately runs the risk of ending up in the registry with a proliferation of service versions that are not much distinct from each other. In practice, customization should be used to build distinguishing services that provides a robust, reusable service portfolio across changing requirements. Hence, when to build a new service and when to customize from an existing service should be used as a complementary strategy.

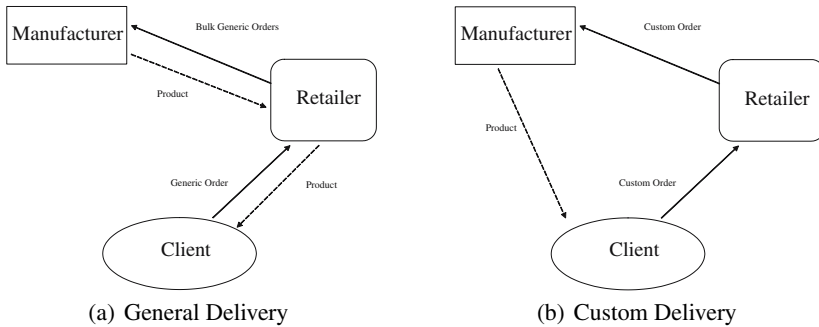
Service providers try to cover as much of the differences as they can anticipate using some of the following techniques:

- *Parametrization*. In here, the actual parameters of the service can be modified based on the data in the arguments [7];
- *Function overloading*. In here, multiple instances of the same operation are defined with different set of arguments [8];
- *Templatization*. In here, a service template is available to capture the typical service of interest. Users are guided to select parameters to instantiate the template and create the service matching their requirements [9].

However, a provider cannot anticipate all types of requirements from potential consumers. The types of differences that can appear between the requirements of a requester and the available services from provider can be along business objects, processes, services and messages.

## 2.2 Motivating Scenario

We present a common scenario to motivate the problem. Consider a simplified business process for product supply-chain, called *General Delivery* where a manufacturer  $M$  sells products to its clients  $C$  via retailers  $R$  (see Figure 1 (left)).  $C$  could place an order for the product with  $R$  and get its delivery.  $R$  on its part can periodically place bulk orders with  $M$  and take delivery to replenish its inventory.



**Fig. 1.** Two differing processes in the example scenario

Suppose that often, some clients want to place custom orders which are different from general products. A typical example is for an apparel manufacturer to allow customers to provide their own logo on the shirts they order. The manufacturer now wants to change the supply-chain slightly to allow these clients to be able to place their custom orders and get the products delivered. Custom orders can still be delivered from  $M$  to  $C$  through  $R$ . However, because custom orders take time to build,  $M$  may want to ship the product directly to  $C$  in new business process called *Custom Delivery*. Such a process for custom orders is shown in Figure 1 (right).

The changes in the scenario come from:

1. Changes in the business objects: Custom orders are now introduced.
2. Changes in the business process: The product now can be shipped to both  $R$  and  $C$  via *General Delivery* and *Custom Delivery* processes. Orders from  $R$  can now be both periodic for general products and unexpected for custom products.
3. The changes in the business object and processes could lead to changes in the interfaces of IT services used to implement them (e.g., *OrderPlacementService*, *ProductTrackingService*) and the messages involved in their communication.

The changes 1 and 2 are business changes necessitated by world events. We want to take them as inputs and automatically determine business and IT changes as shown in 3 for the specific Service-Oriented Architecture (SOA) implementation (at the manufacturer in the example).

### 3 Business-Driven Service Customization Model

In this section, we introduce the model capturing properties and relationships of a business-process driven SOA implementation. The model consists of facts and rules. Facts represent claims about the problem universe and they may be true or false. Rules are used to encode relationships among facts and to infer new facts from the ones in the model. Using the same model, we propose a set of rules to compute the overall impact of a new business requirement spanning both business and IT domains.

We will use logic programming to define the model. Specifically, we used Smodels [10], an implementation of stable model semantics [11] and well-founded semantics [12] for normal logic programs. An answer to a problem is a set of facts, called stable model, which tell which facts are true.

#### 3.1 A Simplified Model for Business-Process Driven SOA Implementation

We envisage that one can define an industry in terms of a collection of scenarios describing what the enterprise does and how. There can be cross-industry scenarios which describe the common activities any enterprise has to perform regardless of its area of business (e.g., *Annual tax filing*). Then there are activities specific to particular industry like *prepare clinical trial* for *Healthcare* or *emission management* for *Mining*.

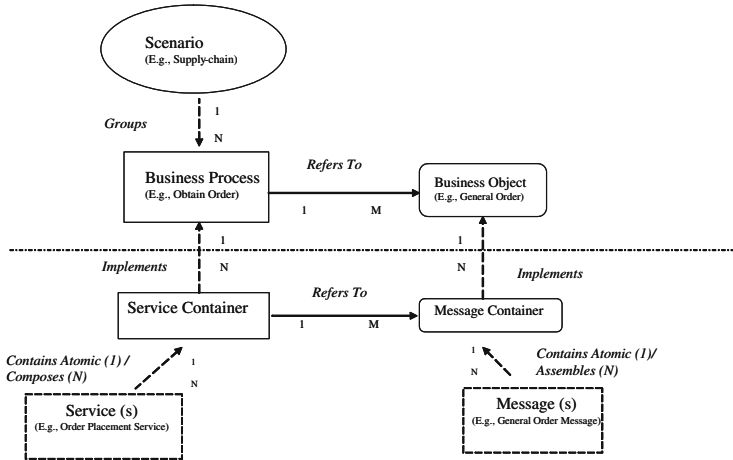


Fig. 2. A simplified model for business-process driven SOA implementation

In Figure 2, we present our simplified model for business-process driven SOA implementation. Note that we explicitly separate business from IT elements and we use directional arrows to indicate dependencies between elements.

At business level, an enterprise domain is decomposed into multiple scenarios. Each scenario is realized by one or multiple business processes (BPs) and by one or multiple business objects (BOs). Each BP is defined with a business logic and references to BOs.

```

% Business Objects Structural Rules
1: busObj (product)
2: busObj (customerAddress)
3: busObj (retailerAddress)
4: busObj (order)
5: boDependsOnBo (order, product)
6: boDependsOnBo (order, clientAddress)
7: boDependsOnBo (order, retailerAddress)
% Business Processes Structural Rules
8: busProc (shipOrder_bp)
9: busProc (receiveOrder_bp)
10: busLogic (shipOrderLogic)
11: busLogic (receiveOrderLogic)
12: bpHasLo (shipOrder_bp, shipOrderLogic)
13: bpHasLo (receiveOrder_bp, receiveOrderLogic)
14: bpRefersToBo (receiveOrder_bp, order)
15: bpRefersToBo (shipOrder_bp, product)
16: bpRefersToBo (shipOrder_bp, retailerAddress)

```

**Fig. 3.** Business level model elements

Each BO can be referred by multiple processes as well as by other BOs. In our model, we denote business scenarios, BPs, business logic, and BOs as *facts* whereas we use *structural rules* to define rules capturing relations between facts.

**Example 1.** Consider the General Delivery scenario presented in Figure 1. For the sake of illustration, we can assume the domain of interest to be represented by four BOs (product, customerAddress, retailerAddress, and order) and two BPs called receiveOrder\_bp and shipOrder\_bp. The former BP collects new orders whereas the latter loads existing orders and prepares new shipments. The business logics of the two BPs are defined in shipOrderLogic and receiveOrderLogic respectively. Figure 3 shows facts and structural rules composing our simple model. As example, structural rule No. 11 tells that BP receiveOrder\_bp refers to (i.e. uses as part of its logic) BO order.

At IT level, each BP is realized by a Service Container (SC) referring to one or multiple (Web) service(s). In case a SC refers to multiple services (due to service composition), it also connects to a logic describing the process flow among those services. In addition, each SC refers to one or multiple Message Containers (MC) describing atomic or aggregate messages that constitute its inputs and outputs. Messages are IT realizations of BOs in specific formats of interest (e.g., order information in XML format). Once again, we use *facts* to generally indicate the base element of the model (i.e. SCs, services, service logics, MCs, and messages) and *structural rules* represent existing relations among facts.

**Example 2.** Figure 4 shows facts and structural rules modeling the IT aspects of our simple SOA example.

```

% Service Container Structural Rules
1: srvContainer(mngReceivedOrder_ws)
2: srvContainer(processOrder_ws)
3: scImplementsBp(mngReceivedOrder_ws, receiveOrder_bp)
4: scImplementsBp(processOrder_ws, shipOrder_bp)
5: scDependsOnSc(processOrder_ws, mngReceivedOrder_ws)
6: scHasLogic(processOrder_ws, processOrder_logic)
% Message Container Structural Rules
7: messCont(storeOrder_ms)
8: messCont(shipOrder_ms)
9: mcImplementBO(storeOrder_ms, order)
10: mcImplementBO(shipOrder_ms, product)
11: mcImplementBO(shipOrder_ms, retailAddress)
12: scRefersToMc(mngReceivedOrder_ws, storeOrder_ms)
13: scRefersToMc(processOrder_ws, shipOrder_ms)

```

**Fig. 4.** IT level model elements

We assume two SCs exist: `mngReceivedOrder_ws` and `processOrder_ws`. The first SC implements `receiveOrder_bp` BP while the latter implements `shipOrder_bp` BP. In the Figure, structural rule No. 5 states that `processOrder_ws` depends on `MngReceivedOrder_ws` which means that the first SC is a composed service using, among others, some of the messages referred by the second SC. The logic of the SC is captured in `processOrder_logic`.

On the message side, two MCs exist, one for each SC. `storeOrder_ms` represents the operation of accepting new orders (defined using BO `order`) whereas `shipOrder_ms` is in charge of creating a new shipment.

Our notion of SC and MS are consistent with web services standards. SC can be represented by a WSDL and stands for an atomic service or a composite service. The latter logic can be described by abstract BPEL. The MC represent messages in WSDL both simple and complex message types.

In our model, we intentionally kept the definitions of services and messages simple. This is because we are interested in the relations between business and IT elements of a SOA and not to extensively represent all details of a SOA implementation. Existing standards, such as OWL-S [13] or broadly used ontologies like SAP Global Data Type (GDT) [8], can be used for more refined models.

### 3.2 Formalizing Business Process-Driven Impact on Services

In this subsection, we extend the model to include *impact rules* to encode the customization behaviors of a SOA implementation. When triggered by a new customization requirement (e.g. create a new BO for custom order), those rules can precisely scope the overall impact of the requirement to the SOA, spanning both business and IT domains.

In our model, we view customization changes as BO and BP driven impacts on implementing services and messages. When a business element of the model changes, we

use impact rules to propagate such change across the model to identify which (business and IT) elements will be affected.

Impact rules are defined using logic programming and reflect the dependencies depicted in Figure 2. An inference engine is used to compute the overall impact of a new business requirement. The inference engine takes three inputs: a) facts and structural rules, defined for the specific SOA implementation, b) impact rules, defined independently from any SOA implementation and c) the new business requirements expressed as additional facts for the model (e.g. `changeBusObj(order)`).

Figure 5 presents the list of impact propagation rules we defined using Smodel. For example, rule No. 2 (`changeBusObj(Y) :- boDependsOnBo(X, Y), changeBsObj(Y)`) would tell that if BO X depends on BO Y and BO Y changes, then BO Y will change. In the example, X and Y are variables whose values is not defined by the user but rather inferred by the engine starting from facts either in the model or inferred from evaluating other rules.

Instead of describing each rule in detail, in what follows we use an example to describe how the rules compute the overall impact of a new customization requirement.

**Example 3.** Consider once again the scenario introduced in Section 2.2. The business requirement can be modeled with two facts: `changeBusObj(order)`, to represent the need to distinguish between custom and general orders, and `changeBusLogic(ShipOrderLogic)` to represent the change in the business logic for handling custom orders. Note there is not a predefined order according to which rules should be evaluated. Instead, the inference engine analyzes all possible combinations of fact searching for stable models. On the business side, `changeBusObj(order)` propagates, through impact-rules No. 2 and No. 3, to BP `busProc(receiveOrder_bp)` and business logic `processOrderLogic`. As result, the engine will introduce two new facts: `changeBusObj(receiveOrder_bp)` and `changeBusLogic(processOrderLogic)`.

On the IT side, rule No.10 propagates the impact of the change to `ProcessOrderLogic` Service logic whereas rule No. 11 extends it to MC `storeOrder_ms`. Finally rules No. 14 takes in input `changeMessCont(storeOrder_ms)` and propagates the impact of the requirement to `mngReceivedOrder_ws`.

As result of the inferencing process, the inference engine highlights two different actions for the two SCs in the model: a change to the service logic of SC `processOrder_ws` and a change for message (`storeOrder_ms`) in case of SC `mngReceivedOrder_ws`. Such advises are specific to the facts and structural rules defined in the previous subsection and would have been different in case of a different SOA implementation.

The impact propagation rules presented in Figure 5 represent the “sufficient but not necessary” set of changes to be considered as result of a new business customization requirement. It is so because all possible changes in the SOA are identified. However, even though the scope of changes known, a system architect might decide not to take action as result of the new business requirement based on assessments that are not captured in the model. As an example, a system architect might decide not to change a

```

% Business Objects Rules
1: % If a BO X changes than all BOs depending from X should change
   changeBusObj (Y) :-boDependsOnBo (Y, X) , changeBusObj (X)
% Business Processes Rules
2: % If a BO X changes, all BPs referred by X might change
   changeBusProcess (Y) :-boRefersToBp (Y, X) , changeBusObj (X)
3: % If a BP X changes, X's business logic should change
   changeBusLogic (Y) :-changeBusProc (X) , bpHasLo (X, Y)
4: % If a Business Logic X changes, the BO using X should change
   changeBusProc (Y) :-changeBusLogic (X) , bpHasLo (Y, X)
5: % If a BP X changes, all BPs depending from X should change
   changeBusProc (Y) :-bpDependsOnBp (Y, X) , changeBusProc (X)
% Service Container Rules
6: % If a Service X changes, all SCs depending from X should change
   changeServCont (Y) :-scDependsOnSe (Y, X) , changeService (X)
7: % If SC X changes, X's service logic should change
   changeServLogic (Y) :-scHasSl (X, Y) , changeServCont (X)
8: % If Service Logic Y changes, X's service logic should change
   changeServCont (Y) :-scHasSl (Y, X) , changeServLogic (X)
9: % If SC X changes, all SCs referred by X should change
   changeServCont (Y) :-scDependsOnSc (Y, X) , changeServCont (X)
10: % If the logic of BP X changes, the Logic of SC implementing X should change
   changeServLogic (K) :-changeBusLogic (Y) ,
% Message Container Rules
11: % If a BO X changes, all Services implementing X should change
   changeMessCont (X) :-changeBusObj (Y) , mcImplementsBo (X, Y)
12: % If a MC X changes, all MCs depending from X should change
   changeMessCont (Y) :-mcDependsOnMc (Y, X) , changeMessCont (X)
13: % If a Message X changes, all MCs depending from X should change
   changeMessCont (X) :-mcDependsOnMe (X, Y) , changeMessage (Y)
14: % If a MC X changes, all SC referred by X should change
   changeServCont (Y) :-mcRefersToSc (Y, X) , changeMessCont (X)

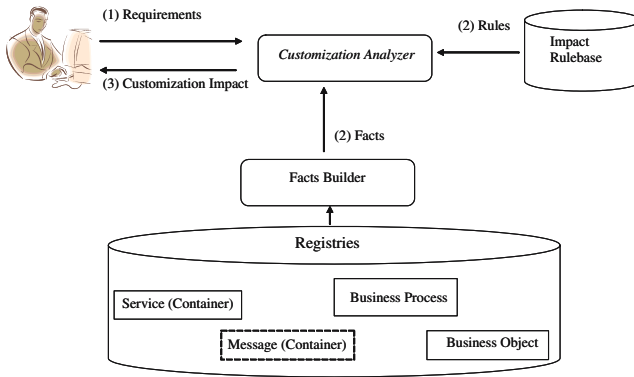
```

**Fig. 5.** Impact Propagation Rules

service message until it is used as part of a business process. Again, even if a BO A becomes irrelevant for the business, a system architect might choose not to change service messages implementing A but simply ignore the content when appears in the services. To better inform the system architect on the changes to be made on the SOA implementation, one can refine the model introducing the concept of “type of change”. In an extended version of our model, business requirements will be defined not only with the affected element (e.g. BO - product) but also with the type of change to implement (e.g. add new BO). Similarly, impact rules will be extended to consider the type of change when propagating a new requirement across the model.

## 4 Implementation Considerations

We now discuss how our approach for business-process driven service customization can be implemented. Recall that the solution consists of three key components - (a) *Facts* - a set of claim about BOs, BPs, and their implementing services and messages of interest, (b) *Structural rules* - rules capturing properties and relationships among facts, and (c) *Impact rules* - universal rules encoding the desirable propagation of customization behavior in the model. A logic checker can work with the facts and the rules to make decisions on what needs to be customized.



**Fig. 6.** A conceptual architecture for realizing *Customization Analyzer*

We envisage this approach to be implemented as a decision support aid as shown in Figure 6. The user wants to enquire about the impact of some business events (requirements). The *Customization Analyzer* translates the requirements to facts. The *Fact Builder* computes facts and structural rules of the SOA instance based on existing registries for known BOs, processes, services and messages<sup>1</sup>. The *Impact Rulebase* contains the impact rules. The *Customization Analyzer* uses the facts and the rules collectively from the requirements, the *Fact Builder* and the *Impact Rulebase* to determine what services could be customized to meet the requirement.

We chose a logic formalism to implement our approach. An alternative would have been to express the relationships using a graph theory formalism like that supported in UML. Unfortunately, such formalisms need extensions to represent constraints and were not our first preference.

**Estimating the size of the model:** With any model-based approach, it is always insightful to know what contributes to the size of the model and use that knowledge for solving problems of interest effectively. Let us consider the size of the model (facts and rules) given the numbers of BOs BPs, and their implementing services and messages as

<sup>1</sup> Current Web Services standards do not require all messages to be explicitly registered but rather that they are addressable and accessible through Uniform Resource Identifiers. Hence, they could be considered optional for the model.



Model Element	# Facts	# Structural Rules	Description
BO	$N_{BO}$	$(N_{BO} * (N_{BO} - 1))/2$	BO can depend on all BOs except itself and those that depend on it.
BP	$2 * N_{BP}$	$((N_{BP} * (N_{BP} - 1))/2) + N_{BP}$	No. of facts includes claims for BP and its business logic. No. of rules analogous to BOs plus the relation between BPs and their logic.
BP-BO	-	$N_{BP} * (N_{BO})^k$	Assume $k$ is an upper limit on the number of BOs referred by a BP.
SC	$(2 * N_{BP})$	$(N_{BP})$	Assume each BP is implemented by at most one SC. No. of facts includes claims for SC and its composition logic.
SC-BP	-	$N_{BP}$	Assume each BP is implemented by at most one SC.
MC	$(2 * N_{BO})$	$(N_{BO})$	Assume each BO is implemented by at most one MC. No. of facts includes claims for MC and its aggregation logic.
MC-BO	-	$N_{BO}$	Assume each BO is implemented by at most one MC.
SC-MC	-	$N_S * (N_M)^k$	Assume $k$ is an upper limit on the number of MCs referred by a SC.
Message	$N_M$	-	Each message is claimed.
Service	$N_S$	-	Each service is claimed.
Service-SC	-	$(N_S)^k$	Assume $k$ is an upper limit on the number of services referred by a SC's logic.
Message-MC	-	$(N_M)^k$	Assume $k$ is an upper limit on the number of messages referred by a MC's logic.

Fig. 7. Estimated maximum ( [ ] ) number of facts and structural rules

$N_{BO}$ ,  $N_{BP}$ ,  $N_S$  and  $N_M$  respectively. Table 7 presents a table estimating the *maximum possible* number of rules in the model with description.

The number of impact rules is a constant,  $I$ , and is independent of the number of entries in the registries. Adding up all the facts and rules up, size of the model is  $O(N_{BP} * (N_{BO})^k + N_S * (N_M)^k + (N_S)^k + (N_M)^k)$ , where  $k$  represents an appropriate constant. This implies that the size is dominated by the number of BOs and how many get referenced by BPs, the number of messages and how many get referenced by services, and the number of services.

In itself, the logic programming system we use, Smodels [10] with stable model semantics, is quite efficient and can handle models with up to a million facts and rules effortlessly [14]. The models in our examples are handled in less than a second. Note that since industry information about BPs and BOs are organized along scenarios, and a user is usually interested in a few scenarios at a time based on their expertise, it should be generally possible to scope the model for most problems of interest.

## 5 Industry Case Study: Customization with SAP Services

In the introduction, we discussed how SAP is moving its architecture towards services. Grouped into sets (called bundles), these services have built-in semantics which partition data into an extensible set of BPs and BOs. In this section, we adopt SAP to

illustrate the application of our approach in an industry scenario with large number of services with complex characteristics.

In our study, we implemented the ordering scenario in Section 2 using the SAP services publicly accessible from SAP Enterprise Workplace<sup>2</sup>. Specifically, we focused on “Sales Order Processing”<sup>3</sup>, the set of capabilities which “allows sales representatives to easily configure, price, and create sales orders for customers”.

To understand the level of complexity in creating an SOA solution using services like the one offered by SAP, consider that SAP implements a large number of services covering different business processes and industry scenarios. Not only, the same process might be offered in different variants (e.g. SAP Workplace lists 11 variants for “Sales Order Processing”). Enterprise services from SAP might be very complex, exposing interfaces (WSDL) with several thousands attributes in input and output. In fact, the approach used by SAP in creating services is to include all possible variations directly as optional elements of the service interface. In our example, if we focus on “the operations that sales employees can use to read or process data about a customer” (manage `customer_in` in the Workplace), we can find 15 different operations (each implemented as service) for the BO called `customer`. `Customer` is a very complex object referring to multiple data elements including, among others, company name and address, communication data (phone, email, etc), contact person, bank details, industry sectors, and marketing attributes. Not all 15 services uses all `customer`’s attributes. In fact, as presented in Figure 8, there can be multiple versions of the same service to perform slightly different functionalities on different subsets of the BO data element.

<i>ServiceName</i>	<i>Input</i>	<i>Output</i>
CustomerBasicDataByID QueryResponse_In	CustomerID	Read Customer Address and CommunicationData
CustomerERPBasicDataByID QueryResponse_In_V1	CustomerID	Read Customer Address and CommunicationData
CustomerERPBasicDataByID QueryResponse_In_V2	Range of CustomerIDs	Customer Address and CommunicationData

**Fig. 8.** SAP offers multiple version of the same service

The reason for having multiple versions of the same service is because customizing SAP-based services to meet specific user requirements is a complex and tedious task. The user has to (a) identify the service(s) to be enhanced, (b) extend the corresponding BO or BP, and (c) choose among different versions of the same service to find the one which fits better with the new requirement and (d) write the corresponding code to model the new behavior [15]. More importantly, if the enhanced BO is used in multiple services, developer has to manually identify and then implement the code for all affected services to ensure consistent behavior [16].

<sup>2</sup> <https://www.sdn.sap.com/irj/sdn/esworkplace>. Last access June 2008

<sup>3</sup> [http://erp.esworkplace.sap.com/socoview\(bD1lbiZjPTgwMCZkPW1pbg==\)/render.asp?id=B8BE8D31D91E4B9EBAAACD83FF85A614&fragID=&packageid=DBBB6D8AA3B382F191E0000F20F64781&iv=](http://erp.esworkplace.sap.com/socoview(bD1lbiZjPTgwMCZkPW1pbg==)/render.asp?id=B8BE8D31D91E4B9EBAAACD83FF85A614&fragID=&packageid=DBBB6D8AA3B382F191E0000F20F64781&iv=)

```

1 : boDependsOnBo(customer,goodsRecipientParty)
2 : boDependsOnBo(customer,billToParty)
3 : boDependsOnBo(customer,salesTerms)
4 : boDependsOnBo(customer,bankAccount)
5 : boDependsOnBo(customer,items)
6 : mcImplementsBo(customerERPBasicDataByIDQuery_sync_V1,customerId)
7 : mcImplementsBo(customerERPBasicDataByIDResponse_sync_V1,customer)
8 : mcIsReferredbySc(customerERPBasicDataByIDQueryResponse_In_V1,
  customerERPBasicDataByIDQuery_sync_V1)
9 : mcIsReferredbySc(customerERPBasicDataByIDQueryResponse_In_V1,
  customerERPBasicDataByIDResponseMessage_sync_V1)
10: mcIsReferredbySc(StandardMessageFault,
  customerERPBasicDataByIDQueryResponse_In_V1)
11: mcImplementsBo(customerERPBasicDataByIDQuery_sync_V2,customerId)
12: mcImplementsBo(customerERPBasicDataByIDResponse_sync_V2,customer)
13: mcIsReferredbySc(customerERPBasicDataByIDQuery_sync_V2,
  customerERPBasicDataByIDQuery_sync_V2)
14: mcIsReferredbySc(customerERPBasicDataByIDResponse_sync_V2,
  customerERPBasicDataByIDResponseMessage_sync_V2)
15: mcIsReferredbySc(standardMessageFault,
  customerERPBasicDataByIDQueryResponse_In_V2)
16: boReferredByBp(customer,manageCustomer)
17: scImplementsBp(customerERPBasicDataByIDQuery_sync_V1,manageCustomer)
18: boReferredByBp(customer,createOrder)
19: boReferredByBp(order,createOrder)
20: scImplementsBp(customerERPBasicDataByIDQuery_sync_V1,manageCustomer)
21: scImplementsBp(createNewOrderService,createOrder)

```

Fig. 9. SAP Scenario - Facts and Structural Rules

Figure 9 shows a very small fragment of structural rules which have been built using our system for the actual SAP services. In the Figure, a BP `createOrder` creates new orders for the customer. `CreateOrder` composes service `customerERPBasicDataByIDQuery_sync_V1`, from SAP, and service `createNewOrderService`, from the specific back-end system used for billing.

In the example, facts and structural rules modeling the IT elements of the SOA solution have been automatically generated by parsing the interfaces of the services available in the service registry (e.g. IBM Web Service Registry and Repository - WSRR). On the other hand, we look at process model documentation (such as the one generated by IBM Websphere Business Modeler-WBM) to create facts and structural rules related to the business.

In what follows, we describe, using an example, how our model can help an organization to identify all services and processes affected by a business change.

**Example 4.** *New Requirement:* Remove bank account from BO Customer as transactions are always paid cash or via credit card.

*Question:* Which are the services to be changed as result of the requirement?

*This question can be easily answer by our model. First, the system architect defines the requirement as new fact for the model. In the example, the fact `changeBusObj (BankAccount)` is defined. Second, the Customization Analyzer computes the stable model given in input facts, structural and impact rules. Impact Rule No. 1 propagates `change (BankAccount)` to `BO Customer` and from there, rule No. 2, propagates it to `BP CreateOrder`. The same process is continued for the IT elements of the model.*

*At anytime, the Customization Analyzer can trace the impact rules from which a given fact is inferred. As example, it can distinguish (a) the SCs which should change because a referred MC changes (impact rule No. 14) from (b) the SCs which should change because the implemented BP changes (impact rule No. 8). Similarly, it can identify which SCs are implementing a BP (structural rule No. 17 in Figure 9). A system architect can now identify which services should be changed even though they don't implement any BP (`customerERPBasicDataByIDQuery_sync_V2` the our scenario) and decide not to change them as result of the new business requirement. Such information can also be used to identify which versions of the same service exist and which one need to be changed to address the new requirement. This allows keeping fewer versions of the same service and to remove them when not longer in the business.*

## 6 Discussion and Related Work

In this section, we look at how our work can be extended and the related work. Until now, we have described how our approach can be used to determine the extent of change in a SOA implementation as necessitated by business requirements. Some representative approaches for customization from different disciplines are: AI [2], semantic web [3], graph theory [4] and Petri Nets [5].

An organization is likely to integrate services from multiple parties, each with a different approach to customization. In this context, it is very difficult to identify which are the actual tasks to be performed on the system and to estimate the overall cost of implementing the business requirement. Our work can be used to answer many other IT and business questions including: (a) if some existing services will be customized, determine what are the choices and associated costs and (b) determine whether new IT services should be created or existing ones be customized. The two questions are closely related as there exists multiple approaches to customize an element of the model and each alternative might be associated with a different cost. As example, the cost of changing the logic of a SC is different depending on if such a logic is defined externally to the composed service using a workflow specification language (such as BPEL) or tightly coupled (hard-coded) on it. Our approach can address this problem as the model can be extended to consider costs and customization requirements as Facts for the IT elements composing specific SOA implementation. The *Customization Analyzer* will now be able to compute the overall cost of the new business requirement by simply considering the cost of all affected elements. This will be a very valuable information for the business in order to estimate the cost of the operation. The system architects can use such information to properly choose the customization approach for the services they integrate in a SOA solution depending on the customization requirement it might

be subjected to during its lifecycle. At the time, we recognize the need to integrate such functionality and we plan to include cost and Customization approaches in the next release. Some work we recently done in this space could be applied here [17].

Closely related to the notion of customization are the concepts of *personalization* and *adaptation* of web services. In [18], the authors present a personalization approach where a general information service can provide different types of information specific to individual users using RSS. The generic schema is published and by selecting specific sections, the system can personalize the content. In this approach, the service is made specific to the user at run time and an instance of it is not created specifically for the user. In adaptation of web services, the behavior of a deployed web service is modified at runtime based on environmental considerations. Adaptation already assumes that the available service met requester's requirement and we want to ensure this continues as the environment changes. Adaptation approaches can come in two main flavors – one where the primary aim is to compose and deploy a correct workflow, and adaptation is viewed as an afterthought [19]; and another where adaptation issues are viewed while composing workflows [20].

## 7 Conclusion

Motivated by the need to determine the overall impact of a new requirement in a complex SOA environment, in this paper, we introduced the notion of business driven customization of SOA. We introduced a formal model capturing properties and relationships of business objects and business processes, and their implementing services and messages. We discussed the implementation of our approach in multiple setting, and illustrate its application in an industry scenario with large number of services with complex characteristics (SAP). Though the approach was illustrated with web services, the approach is relevant for SOA in general. We believe that the work provides a valuable, explicit model of alignment between business processes to service-based IT implementations.

## References

1. IBM-Global-Services: Accelerating business flexibility, while reducing costs, with composite business services (2007), <http://www-935.ibm.com/services/us/index.wss/offering/gbs/a1027243>
2. Richards, D., Sabou, M., van Splunter, S., Brazier: Artificial intelligence: A promised land for web services. In: The Proceedings of The 8th Australian and New Zealand Intelligent Information Systems Conference (ANZIIS 2003), Macquarie University, Sydney, Australia, pp. 205–210 (2003)
3. Fensel, D., Lausen, H., Polleres, A., Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, Heidelberg (2007)
4. Reichert, M., Dadam, P.: Adeptflex-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems* 10(2), 17–93 (1998)

5. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COOCS, pp. 10–21 (1995)
6. Liang, H., Sun, W., Zhang, X., Jiang, Z.: A policy framework for collaborative web service customization. In: Proc. SOSE (2006)
7. Amazon: Amazon web services (Last Accessed June 2008), <http://aws.amazon.com>
8. Campbell, S., Mohun, V.: Mastering Enterprise SOA with SAP NetWeaver and mySAP ERP. John Wiley & Sons, Inc., New York (2006)
9. ten Teije, A., van Harmelen, F., Wielinga, B.: Configuration of web services as parametric design. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS, vol. 3257, pp. 321–336. Springer, Heidelberg (2004)
10. Niemelä, I., Simons, P.: Smodels - an implementation of the stable model and well-founded semantics for normal lp. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 421–430. Springer, Heidelberg (1997)
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K. (eds.) Proceedings of the Fifth International Conference on Logic Programming, pp. 1070–1080. The MIT Press, Cambridge (1988)
12. van Gelder, A., Ross, K., Schlipf, J.S.: The well-founded semantics for general logic programs. *Journal of the ACM* 38(3), 620–650 (1991)
13. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing semantics to web services: The owl-s approach (2004)
14. East, D., Iakhiaev, M., Mikiutiuk, A., Truszczyński, M.: Tools for modeling and solving search problems. *AI Commun.* 19(4), 301–312 (2006)
15. Hirsch, R.: Enterprise soa explorations: Options to deal with enterprise services that don't meet user requirements. Blog Entry at SAP sdn (2008), <http://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8665>
16. SAP: Enterprise services enhancement guide (2007)
17. Chang, Y.C., Mazzoleni, P., Mihaila, G.A., Cohn, D.: Solving the service composition puzzle. In: Proc. SCC (2008)
18. Abiteboul, S., Amann, B., Baumgarten, J., Benjelloun, O., Ngoc, F.D., Milo, T.: Schema-driven customization of web services. In: Proc. VLDB (2003)
19. Au, T.C., Kuter, U., Nau, D.S.: Web service composition with volatile information. In: International Semantic Web Conference, pp. 52–66 (2005)
20. Chafle, G., Doshi, P., Harney, J., Mittal, S., Srivastava, B.: Improved adaptation of web service compositions using value of changed information. In: Proc. ICWS, Salt Lake City, USA (2007)

# Sound Multi-party Business Protocols for Service Networks\*

Michele Mancioppi<sup>1</sup>, Manuel Carro<sup>2</sup>,  
Willem-Jan van den Heuvel<sup>1</sup>, and Mike P. Papazoglou<sup>1</sup>

<sup>1</sup> INFOLAB, Dept. of Information Systems and Management,  
Tilburg University, The Netherlands

{m.mancioppi,wjheuvel,mikep}@uvt.nl

<sup>2</sup> Universidad Politécnica de Madrid

mcarro@fi.upm.es

**Abstract.** Service networks comprise large numbers of long-running, highly dynamic complex end-to-end service interactions reflecting asynchronous message flows that typically transcend several organizations and span several geographical locations. At the communication level, service network business protocols can be flexible ranging from conventional inter-organizational point-to-point service interactions to fully blown dynamic multi-party interactions of global reach within which each participant may contribute its activities and services. In this paper we introduce a formal framework enriched with temporal constraints to describe multi-party business protocols for service networks. We extend this framework with the notion of multi-party business protocol soundness and show how it is possible to execute a multi-party protocol consistently in a completely distributed manner while guaranteeing eventual termination.

## 1 Introduction

Today's application-oriented services cannot scale to meet the number and nature of demands already placed on them, let alone a new generation of more complex applications involving several organizations. Most of today's applications are based on the assumption of the ubiquitous availability of point-to-point integration between any two interacting parties from the perspective of a single organization. One of the main reasons is the use of orchestration languages (e.g., BPEL) to describe how services can interact with each other at the message level from the perspective and under control of a single service. Moreover, the interactions are limited to uni-cast scenarios. This is extremely restrictive for applications characterized by wide-scale and complex dynamic interactions.

---

\* The research leading to these results has received funding from the European Community's Seventh Framework Programme under the Network of Excellence S-Cube - Grant Agreement n° 215483. Manuel Carro was also partially supported by Spanish MEC project TIN2005-09207-C03 *MERIT-COMVERS* and project S-0505/TIC/0407 *PROMESAS*.

## 1.1 Service Networks

The full potential of services technology as a means of developing mission-critical applications used by a wider spectrum of people and organizations will only be realized when business processes (which are services themselves) are able to express business collaborations and transactions that occur between multiple business process endpoints, rather than a specific business process that is executed from the perspective of a single party. Such collaborative, complex service interactions typically require specifying sequences of peer-to-peer message exchanges between a collection of end-to-end services within stateful, long-running interactions involving several parties. This gives rise to the concept of *service networks*.

Service networks comprise large numbers of long-running, highly dynamic complex end-to-end service interactions reflecting asynchronous message flows that typically span several organizations and geographical locations. The term “complex end-to-end service interaction” encompasses a succession of automated business processes, which are involved in joint inter-company business conversations and transactions across a federation of cooperating organizations. This widens considerably the scope of service-based applications by providing the possibility of developing a whole new range of innovative service-based applications.

Service networks properly sequence service activities according to the flow definitions in a process collaboration model into end-to-end service constellations, assign work items to the appropriate human actors or groups, and ensure that both human- and system-based activities are performed within specified time frames. This entails multiple technical requirements, which include binding to heterogeneous systems, synchronous and asynchronous message exchange patterns, data manipulation, flow coordination, exception management, business events, long running business transactions, and so on.

## 1.2 Multi-party Business Conversations in Service Networks

At the communication level, service networks essentially comprise asynchronous message flows between multiple service consumers and providers. Business conversations can be flexible, ranging from conventional inter-organizational point-to-point service interactions (as is the norm with current orchestration technologies) to *fully blown dynamic multi-party interactions* of global reach within which each participant may contribute its activities and services.

At the communication-level, service networks exchange sequences of messages grouped into operations, which have to be structured into complex conversations. The business logic which controls these operations is embedded into the implementation of the interacting parties. This is due to the limitations of standards used in practice, e.g., WSDL 1.1. The new generation of Web services standards that will be based on WSDL 2.0 will focus on custom *Message Exchange Patterns* (MEPs). MEPs are currently elementary building blocks (i.e., one-way messaging, request-reply, solicit-response and notification) from which business protocols can be constructed, describing the multi-party message interactions required by a business process. Ideally, MEPs should also include timing



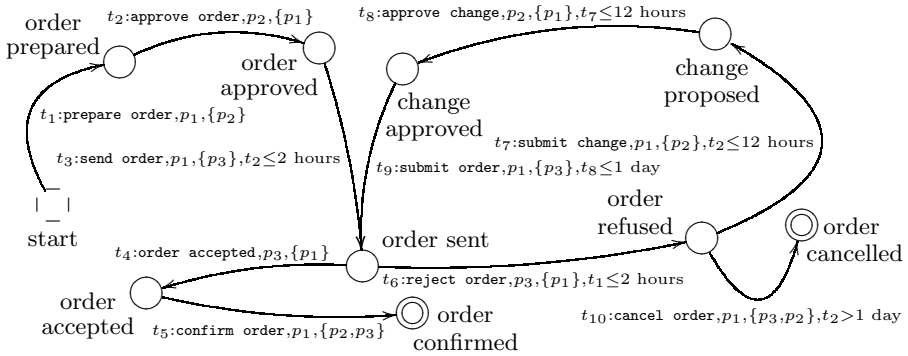


Fig. 1. The *Purchase Order* business protocol

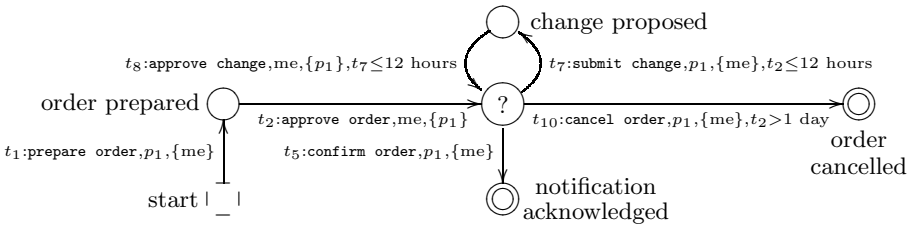


Fig. 2. The *Purchase Order* protocol from the perspective of the sales approver

constraints and have the expressive power necessary to describe complex series of interactions allowing for multiple alternative message exchanges to be performed at any given point in the execution of the MEP.

Figure 1 describes a simplified version of an end-to-end order fulfillment process represented by means of a complex set of interacting Web services. From a service network perspective the *Purchase Order* business protocol essentially corresponds to a *choreography* scenario and involves message exchanges between three parties: a buyer ( $p_1$ ), a sales approver ( $p_2$ ), and a seller ( $p_3$ ), described from a global perspective. A buyer’s order conveys information about the order. The sales approver performs credit check and stock authorization, while the final order fulfillment and billing lies with the seller.

The previous scenario can be contrasted with the one illustrated in Figure 2 which describes the *Purchase Order* protocol from the sales approver’s perspective, which effectively corresponds to a service *orchestration* scenario, where all information not concerning the sales approver, both about states of the protocol and transitions, has been removed. The only message exchanges relevant for the sales approver are the ones in which it appears as either sender or recipient of messages. Note that in the service network business protocol where the sales approver is listed as recipient of a message together with other participants, information about other recipients, like transitions  $t_5$  and  $t_{10}$  in Figure 2, is eliminated as it cannot be observed by the sales approver. Eliminating information regarding transitions that do not involve the sales approver may render

some states of the business protocol irrelevant. Collapsed states are labelled as “?” in Figure 2 and are referred to as *incognito* states. Incognito states are “super-states” in orchestrations which resume states in the original choreography that are not discernible by the participant because of lack of information. The algorithm to extract the point of view of a participant from a choreography (and the creation of the incognito states) [1] is outside the scope of this paper.

Business protocols such as the one depicted in Figure 1 can lead to erroneous results if not managed properly. Therefore, an important consideration is how to factor a multi-party business protocol to achieve end-to-end conversation sequences that are robust and safe. Of particular importance is the use of formal techniques to describe multi-party MEPs for service networks and verify correct execution and temporal properties of these protocols.

In this paper we define a formal model for multi-party business protocols based on *Deterministic Finite Automata* (DFA). Our model has been inspired by [2], which we have extended to describe multi-participant orchestration business protocols and choreographies. We also introduce the notion of multi-party business protocol soundness and show how it is possible to execute a multi-party protocol consistently in a completely distributed and timely manner without relying on any external synchronization mechanisms.

## 2 Formal Exposition of Business Protocols

This section introduces the basics of our formalization of business protocols. We use a graph-based representation which permits us to give an intuitive and simple semantics to the execution of runs, and which makes it easy to perform a mapping to timed automata, which enables model checking-based verification of temporal logic properties. Space constraints force us to be concise; the interested reader can find more details and examples in [1].

### 2.1 Running Example: Purchase Order Business Protocol

Our representation of business protocols is based on DFA, which are considered appropriate for describing message exchanges for e-commerce applications [3], enriched with time conditions on the transitions. The states of the protocol are mapped to states in the DFA, and the protocol evolves by traversing transitions. Transitions are uniquely identified by their *transition identifiers*, and have associated *time conditions* that restrict when they can be traversed. There are two types of transitions: *message-based* and *automatic*. Message-based transitions are associated to a message exchange between a *sender* and a number of *recipients*. Automatic transitions are triggered by their associated time conditions becoming true when time advances.

In Figure 1, the participants communicate with each other by exchanging messages. The buyer *initiates* the conversation (and thus it is the *initiator*) by sending the message **prepare order** to the sales approver (transition  $t_1$ ). The notation

$$t_1 : \text{prepare order}, p_1, \{p_2\}$$

means that the message-based transition  $t_1$  represents the delivery of an instance of the message type `prepare order` by the participant  $p_1$  to the participant  $p_2$ . The sales approver authorizes the order and replies back to the buyer with an `approve order` message (transition  $t_2$ ). Following this, the buyer has to dispatch the order within two hours to the seller (transition  $t_3$ ) with the `send order` message. If the seller accepts the order, it sends to the participants a message `order accepted` (transition  $t_4$ ). The seller can reject an order by sending the message `reject order` (transition  $t_6$ ) within two hours since the reception of the message `send order`. If the order is accepted, the conversation ends by traversing transition  $t_5$ , where the buyer sends the message `confirm order` to both seller and buyer. The buyer can, however, cancel an order that has been rejected within one day ( $t_{10}$ ) or propose changes the order to the sales approver a within 12 hours. In this case, the sales approver must approve the change within 12 more hours ( $t_7$  and  $t_8$ ). The buyer then sends another message to the seller ( $t_9$ ) with the updated order for the seller to either approve or reject it.

Each message-based transition in the business protocol is univocally associated with a message type. Different message types in the business protocol are disjoint: that is, given a message that is an instance of a message type, it is possible to map it back to only that message type.<sup>1</sup> Thus, recipients are able to tell which transition has taken place simply by observing the message they received. Each business protocol has a unique initial state, which has no incoming transitions. All transitions outgoing the initial state are message-based, and their associated time conditions are “*true*”. Multiple final states are allowed, and final states are required to be absorbing (i.e. they have no outgoing transitions).

Time conditions determine when transitions can be traversed. Message-based transitions can be traversed only if their associated time expressions evaluate to “**true**” and the corresponding exchange of messages between the partners actually is actually performed. Automatic transitions are immediately traversed as soon as their associated time expressions evaluate to “**true**”. Time expressions are obtained by composing *atomic predicates*, such as “*true*” or “ $t_3 > 2$  hours” which refer to the time at which a transition happened using directly its transition identifier (e.g.,  $t_3$ ). In a sense this is similar to what happens in timed automata [4] (and we assume similar time expressions are used), where *clocks* store information on elapsed time, and they can be reset by traversing transitions. In our proposal, every transition has one associated timer (named as its transition identifier) which is reset every time the transition is traversed. We will use specific time expressions to denote either absolute points in time (e.g., 12:35AM) or durations (e.g., 2 hours), and we will let the context disambiguate if needed.

The time at which a time expression is evaluated is used as the reference for the evaluation of atomic predicates. Atomic predicates referencing a not-

---

<sup>1</sup> Requiring all message-types to be completely disjoint with each other is not a limitation in the current WS landscape: message instances can be marked with custom SOAP headers (thereby without affecting the actual contents of the message), by embedding message identifiers, or by defining the message types as XSD type or element definitions accompanied by *XPath* expressions acting as assertions.

yet-taken transition evaluate to “**false**”, while “**true**” time expressions can be omitted. Time expressions combining atomic predicates are evaluated with the usual rules for conjunction, disjunction, and negation.

### 2.2 Taxonomy of Business Protocols

In this section we introduce a taxonomy of business protocols according to the following two dimensions:

- *Number of participants* involved in the conversations:
  - two-party**: two participants, or
  - multi-party**: more than two (but finitely more) participants.
- The *perspective* adopted to describe the structure of the conversation:
  - orchestration**: the conversation is described as the point of view of a particular participant (as in Figure 2), or
  - choreography**: the conversation describes all the possible interactions that multiple participants in a service network can have (Figure 1).

The classification is based on the combinations of the two parameters, as presented in Figure 3. We will use this classification later on, as different classes of business protocols have different properties. For instance, soundness in service networks (Section 3) is defined in terms of two-party choreography and multi-party choreography business protocols.

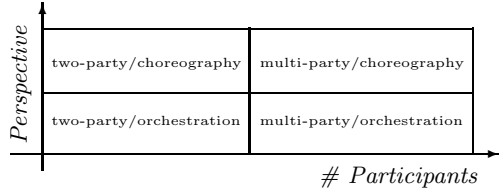


Fig. 3. Taxonomy of Business Protocols

### 2.3 Execution of Business Protocols

The execution of a business protocol essentially implies traversing the states following the usual automata conventions, in addition to the considerations stated in Section 2.1 regarding when a transition can be traversed according to its time condition. A sequence of consecutive transitions that goes from the initial state to a final state is an *execution path*. Examples of execution paths on the business protocols presented in Figure 1 are the following:

$$ex_1 := t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$$

$$ex_2 := t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_6 \rightarrow t_7 \rightarrow t_8 \rightarrow t_9 \rightarrow t_4 \rightarrow t_5$$

There are usually multiple execution paths in the same business protocol (actually, there may be infinite if there are loops).

A particular execution of a business protocol is called a *run*, and it consists of a sequence of *steps*  $(t, \tau)$ , corresponding to the traversal of transition  $t$  at time  $\tau$ . A run  $r_B^n$  of length  $n$  on the business protocol  $B$  is represented as:

$$r_B^n := (t_1, \tau_1) \rightarrow \dots \rightarrow (t_i, \tau_i) \rightarrow \dots \rightarrow (t_n, \tau_n)$$

where  $(t_i, \tau_i) \rightarrow (t_{i+1}, \tau_{i+1})$  represents that  $t_i$  was traversed at time  $\tau_i$ , followed by the step  $(t_{i+1}, \tau_{i+1})$ . In a sense, runs are instances of execution paths. Different executions that follow the same execution paths may give rise to runs that differ on the times associated with the steps. The information available in the run (traversed transitions, their order, and their associated time), is used to evaluate time conditions in the remainder of the execution.

The set of all the runs that can take place on the business protocol  $B$  respecting the time constraints in it is denoted by  $R_B$ .<sup>2</sup> Runs in  $R_B$  are said to be *accepting* on  $B$ . Accepting runs have to follow the usual word accept rules for automata (they start in the initial state, end in a final state, and every transition starts in the state the previous one ended in) and the rules concerning the semantics of time conditions. For example, the following run is not accepting with respect to the business protocol illustrated in Figure 1 because it violates the time constraints associated with the transition  $t_3$ :

$$(t_1, 0h) \rightarrow (t_2, \tau_2) \rightarrow (t_3, 2h:30m) \rightarrow (t_4, \tau_4) \rightarrow (t_5, \tau_5)$$

More precisely, if we denote by  $c_t$  the time conditions associated with transition  $t$  are:

- For any step  $(t, \tau)$ ,  $c_t$  must evaluate to **true** at time  $\tau$ .
- For any two steps  $(t_a, \tau_a) \rightarrow (t_b, \tau_b)$  where  $t_b$  starts in state  $s$ :
  - $\tau_a < \tau_b$  must hold (i.e., time increases monotonically).
  - If  $t_b$  is message-based, no condition of any automatic transition departing from  $s$  may have evaluated to **true** in the time span from  $\tau_a$  to  $\tau_b$ .
  - If  $t_b$  is an automatic transition,  $\tau_b$  is the least time greater than  $\tau_a$  in which  $c_{t_b}$  evaluated to **true**, and no time condition of any transition starting at  $s$  may have evaluated to **true** in the time span from  $\tau_a$  to  $\tau_b$ .

## 2.4 Mapping Business Protocols to Timed Automata

Expressing and checking temporal properties, like “every possible accepting run of the business protocol completes in at most 20 seconds”, is important to, for example, ensuring that time-related QoS properties hold. In order to pave the way towards model-checking of business protocols, we provide a mapping from business protocols to timed automata. Timed automata are labelled transition system which use real-valued variables (*timers*) to model clocks. Timed automata accept *timed words*, where some input symbols can be accepted (i.e., the corresponding transition is taken) only at certain points in time specified by expressions on the timers.

The mapping, adapted from the one in [2] for two-party orchestration business protocols, converts states of the business protocol  $B$  into states of the *Equivalent Timed Automaton* (ETA)  $T_B$  preserving the markings for initial and final states. A timer is created for each transition using the identifier of that transition, which is reset to zero every time the transition is taken. Transitions in  $B$

<sup>2</sup> Borrowing terminology from automata theory,  $R_B$  is the *language* of  $B$ .

are converted into transitions in  $T_B$ , labelled with the original identifier in the business protocol. The time conditions associated with message-based transitions are copied unmodified, but the conditions associated with automatic transitions require some additional tweaking. It is necessary to enforce in the ETA that, in case that several automatic transitions starting in the same state of a business protocol can evaluate to true simultaneously, only one (e.g., the one with the least transition identifier) is traversed.

Therefore, for an automatic transition  $t$  starting at state  $s$  we just need to translate its associated time conditions into  $T_B$  as the (logical) conjunction of the time conditions for  $t$  and the conjunction of the **negation** of the time conditions associated to any other automatic transition originating at  $s$  whose transition identifier is smaller than  $t$ .<sup>3</sup>

The resulting ETA is deterministic. The language of  $T_B$  is, by construction, exactly  $R_B$  (see Section 2.3). Thus, checking whether a run can or can not take place on a certain business protocol boils down to checking if the run belongs to the language of its equivalent timed automaton.

The mapping to timed automata makes it possible to analyze, among other properties, the *inclusion* and *equivalence* of languages of business protocols, i.e., if a given business protocol can execute all or only some of the runs of another protocol. Inclusion and equivalence of languages is the corner-stone for any work on replaceability and compatibility of business protocols. Since the ETAs we generate are deterministic,<sup>4</sup> and they have by construction exactly the same language of the respective business protocols, the analysis of the inclusion of languages [4] on them solves the problem for the business protocols.

### 3 Sound Multi-party Business Protocols

For a given choreography business protocol (either two-party or multi-party) it is important to ensure that can be executed in a completely distributed and timely manner using the message exchanges in the protocol as the only communication means. If participants exchange messages at their own discretion (e.g., outside their time windows), the business protocol may be wrongly executed; they must therefore know when messages can be exchanged. Business protocols in which completely distributed execution is possible are called *participant-sound*.

An additional interesting property of choreography business protocols, called *time-soundness*, is the ability to avoid protocol stalls that could be caused by the *discretionary ability* of participants to decide whether to generate or not messages in a business protocol. Message-based transitions define *time windows* within which senders can generate messages with a *valid timestamp*. Time-soundness guarantees that the protocol eventually concludes disregarding messages which are not generated within the appropriate time frame.

<sup>3</sup> An example of this procedure can be found in [1].

<sup>4</sup> Deciding of language inclusion for non-deterministic timed automata is undecidable. However, the inclusion problem for deterministic timed automata is decidable [4].

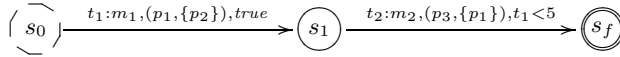


Fig. 4. A non participant-sound multi-party choreography business protocol

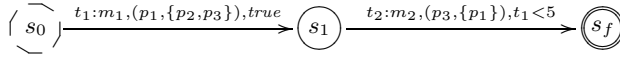


Fig. 5. A participant-sound version of the protocol in Figure 4

### 3.1 Participant-Soundness

A participant-sound business protocol can be executed correctly using as the only communication means among the participants the message exchanges in the protocol. Consider the business protocol presented in Figure 4: participant  $p_3$ , which is not involved in the message exchange upon traversing transition  $t_1$ , does not know that the protocol has entered state  $s_1$  and that it is therefore expected to generate message  $m_2$ . Thus, the protocol is not participant-sound, because if  $p_3$  relies only on the messages exchanged with the other participants, it will not have information enough to take part in the execution without risking to break the protocol by generating a message in the wrong moment. Consider now Figure 5, obtained by adding  $p_3$  as recipient for the message-based transition  $t_1$ . Unlike the protocol in Figure 4, the one in Figure 5 is participant-sound: upon the receipt of message  $m_1$ ,  $p_3$  knows that the protocol is now in the state  $s_1$ , and that it can generate message  $m_2$ .

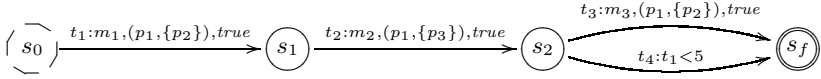
In order for a business protocol to be participant-sound, its participants must be able to evaluate time conditions associated to message-based transitions in which they act as either senders or recipients.

Recipients of message-based transitions can *observe* traversed transitions that affect them by retrieving the types of the messages they receive and using the one-to-one mapping between message types and message-based transitions (Section 2.1). Similarly, a time condition associated with an automatic transition (Section 2.1) defines the time windows in which that transition is immediately traversed if the protocol is currently in its source state. Automatic transitions do not require communication among participants: a participant *infers* that an automatic transition takes place when they can evaluate their time windows and knows that the execution is in the transition’s source state.

If a participant knows when a transition can or cannot be traversed, it is said to be *transition-aware* of that transition:

**Definition 1 (Transition-awareness (Working Definition)).** *A participant  $\mathbf{p}$  is transition-aware of a transition  $\mathbf{t}$  in a business protocol  $\mathbf{B}$  if every time  $\mathbf{t}$  occurs during an execution of  $\mathbf{B}$  one of the following holds:*

- $\mathbf{t}$  is message-based, and  $\mathbf{p}$  is involved in it (that is,  $\mathbf{p}$  is either sender or recipient in  $\mathbf{t}$ ), or
- $\mathbf{t}$  is automatic, and  $\mathbf{p}$  can infer that  $\mathbf{t}$  has occurred.



**Fig. 6.** State-awareness with automatic transitions

Transition-awareness affects the ability of participants to evaluate time conditions. A participant can evaluate a time condition if and only if it is transition-aware of all the transition identifiers appearing in that time condition. Due to transition-awareness, the participant always knows when the transition is traversed, and can keep track of the time of the most recent occurrence to evaluate time conditions defined on the basis of that transition.

Definition 1 is only a “working definition” because, while it delivers the intuition of transition-awareness, it does not explain when participants can infer the execution of automatic transitions. In order to formally explain this, there is some more ground work to do.

Similarly to transition-awareness, a participant is said to be *state-aware* of a state if it is aware of every time that the business protocol enters or leaves that state. This means that the participant is being informed of all transitions incoming and outgoing a specific state. Consider the multi-party choreography business protocol in Figure 6. The participant  $p_1$  is state-aware of  $s_2$  because it is aware of all transitions entering this state, all the message-based transitions leaving it ( $t_3$ ), and it can evaluate when  $t_4$  can be taken because it was also aware of transition  $t_1$ , needed to evaluate the condition  $c_{t_4} \equiv t_1 < 5$ . On the other hand,  $p_3$  is not transition-aware of  $t_1$ . Therefore, it cannot evaluate correctly  $c_{t_4}$ , and thus it is not state-aware of  $s_2$  as it cannot tell when  $s_2$  is left through  $t_4$ .

Transition- and state-awareness, which are mutually dependent, are the keys to participant-soundness: if participants are aware of the message-based transitions in which they are involved, they have enough understanding of the protocol not to break it. The definitions of state- and transition-awareness follow:

**Definition 2 (State-awareness).** *A participant  $\mathbf{p}$  in a business protocol  $\mathbf{B}$  is state-aware of  $\mathbf{s}$  if  $\mathbf{p}$  is transition-aware of all transitions entering  $\mathbf{s}$  and, for every transition  $\mathbf{t}$  exiting  $\mathbf{s}$ :*

- If  $t$  is message-based, then  $\mathbf{p}$  is either the sender or a recipient of  $\mathbf{t}$ , and  $\mathbf{p}$  can evaluate the time condition associated with  $\mathbf{t}$ .
- If  $t$  is automatic, then  $\mathbf{p}$  can evaluate the time condition associated with  $\mathbf{t}$ .

**Definition 3 (Transition-awareness).** *The participant  $\mathbf{p}$  in the business protocol  $\mathbf{B}$  is transition-aware of a transition  $\mathbf{t}$  with source state  $\mathbf{s}$  if and only if one of the following conditions hold:*

- $\mathbf{t}$  is message-based,  $\mathbf{p}$  is sender in it, and  $\mathbf{p}$  is state-aware of  $\mathbf{s}$ , or
- $\mathbf{t}$  is message-based, and  $\mathbf{p}$  is recipient in it, or
- $\mathbf{t}$  is automatic, and  $\mathbf{p}$  is state-aware of  $\mathbf{s}$ .



Definitions 2 and 3 show how awareness *spreads* through the executions of a business protocol: a participant is state-aware of a state if it is transition-aware of the transitions entering that state.<sup>5</sup> A participant has to be state-aware of its source state in order to be transition-aware of automatic and message-based transitions in which it acts as the sender. A participant's awareness of states and transitions follows *paths of awareness*, made up of sequences of transitions and states that it is aware of, throughout the executions of a protocol. The paths always start with a message-based transition in which the participant is a recipient, and always end with message-based transitions entering final states, or states the participant is not aware of. The only exception to this rule are the paths of awareness for the participant that initiates a protocol. These start with a message-based transition originating in the initial state and where the participant is the sender and not a recipient.

Participants need also to know when the protocol has terminated. If a participant is not aware of the completion of a protocol run (i.e., if it does not know that the protocol has entered a final state), it might wait forever for messages that will never arrive. To prevent this from happening, participants are required to be aware of all final states.

It is possible to define participant-soundness on the basis of the definition of state-awareness:

**Definition 4 (Participant-soundness).** *A multi-party choreography business protocol is participant-sound if:*

1. *for every message-based transition  $t$  originating in a state  $s$ , the sender  $p$  of  $t$  is state-aware of  $s$ ;*
2. *all participants are state-aware of the final states that belong to execution paths that contain transitions that the participants are aware of.*

That is, participant-soundness expresses the capability that senders have to generate their messages within the correct time windows and that all have participants to acknowledge the termination of the protocol execution if they were apart of it (note that it may be the case that some participants did not participate in the execution path followed by a certain instance).

### 3.2 Time-Soundness

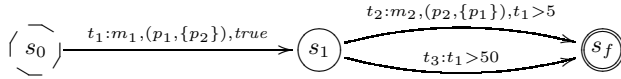
As already explained, senders of message-based transitions can decide not to generate messages (and, thus, not to actually traverse transitions). This may suspend the execution of protocols. Consider the protocol in Figure 7 if  $p_2$  does not wish to generate the message  $m_2$  while the protocol is in the state  $s_1$ , the execution will never reach a final state, and therefore the protocol may *stall*.

Business protocols which never stall because of the participants' discretionary ability to decide whether and when they would participate in message exchanges

<sup>5</sup> For reasons of simplicity, the formulation of Definition 2 and 3 are such that they may be affected by circularity in case the business protocol has loops in it. Definition of transition- and state-awareness not affected by circularity (but for all other practical purposes equivalent to the ones here proposed) are available in [1].



**Fig. 7.** A business protocol which may stall in state  $s_1$  when the participant  $p_2$  decides not to generate the message  $m_2$



**Fig. 8.** A time-sound version of the business protocol presented in Figure 7

are called *time-sound*. The implication is that a business protocol is time-sound if, in every non-initial state of the protocol in which participants apply discretionality, there is always a possibility for the protocol to proceed to a new state. Message-based transitions leaving the initial state (like transition  $t_1$  in Figure 7) are treated as special cases. Since the initial state has no incoming transitions, the traversal of a message-based transition originating in the initial state always represents the beginning of a protocol execution. Figure 8 shows a time-sound protocol, as per Definition 5, below:

**Definition 5 (Time-soundness).** *A business protocol is time-sound if for every non-initial state that has at least one outgoing message-based transition, there is at least one outgoing automatic transition whose associated time condition is satisfied infinitely often*<sup>6</sup>

Because of the discrete time model adopted by business protocols, having infinitely often verifiable time conditions associated with automatic transitions guarantees that, no matter when a state is entered, at some point in the future one automatic transition will be traversable, and therefore the protocol execution will traverse it. While this does not prevent infinite loops from occurring, it is enough to prevent a protocol from stalling in a specific state. Note that these automatic transitions may very well lead the protocol to some *emergency state* to escape from unexpected situations (e.g., deadlines not met).

### 3.3 Full Soundness

Multi-party choreography business protocols that are both participant- and time-sound are *fully sound*. Fully sound business protocols exhibit a *progression property*: their execution is never indefinitely blocked in a non-initial and non-final state. This can alternatively be formulated as follows: finite runs of fully sound business protocols complete (i.e., reach a final state) in finite time. Theorem 10 proves the property of progression.

<sup>6</sup> That is, there exist infinitely many moments in time in which the time condition evaluates to “**true**”.

**Theorem 1 Progression of Fully Sound Business Protocols** *Assuming that participants do not willingly violate time windows for message generation, every accepting run  $r_B^n$  of finite length on a multi-party service network business protocol  $B$  reaches a final state  $s_f$  in finite time. Moreover, at no step in its execution the protocol is broken because of the generation of messages outside their respective time-windows.*

*Proof.* Proven by induction on the construction of an accepting run. There are two basic cases, first and last step of the run, and an inductive one on the  $i$ -th step of the run, with  $i \in (1, n)$ .

- *first step:* since  $r_B^n$  is accepting, the first step traverses a message-based transition originating in the initial state. The time in the execution does not start until the first transition is traversed. Consequently, this case presents no problem. The time condition associated to a message-based transition originating in the initial state must be “true” (Section 2.1), and thus the initiator participant can not possibly violate the time-window for the generation of the first message.
- *last step:* since  $r_B^n$  is accepting, the  $n$ -th (and final) step ends in a final state of  $B$ , and the execution is completed. Since there are no transitions outgoing final states, no messages breaking the protocol can be generated.
- *inductive case:* step  $i - 1$  ( $i - 1 < n$ ) ends in a state  $s$ . Since the run is accepting, the state  $s$  is a neither initial nor final. Since  $s$  is not final, it has at least one outgoing transition. The transition  $t$  to be traversed at the  $i$ -th step can be either automatic or message-based. If  $t$  is automatic, then the time condition associated with  $t$  is infinitely often satisfied due to the time-soundness property of  $B$ . As traversing an automatic transition does not generate any messages, no time-window can be violated. If  $t$  is a message-based transition, the sender  $p$  of  $t$  generates the associated message  $m$  in a finite amount of time, and this happens before the time condition of any automatic transition is satisfied. Otherwise, an automatic transition is traversed instead. Due to the participant-soundness of  $B$ ,  $p$  can evaluate the time-window, and thus it has all the information necessary to generate the message without breaking the protocol.

Theorem 1 proves a fundamental result regarding fully sound business protocols: runs in which participants adhere to the execution rules (i.e., do not generate messages outside the allowed time windows) always complete in finite time. Moreover, because of the participant-soundness of the protocol, participants can execute the protocol in a completely distributed way. This result builds on the following assumptions:

- Reliability of communication channel:** sent messages are always successfully delivered;
- Reliable time measurement:** participants have consistent means of measuring time, i.e., private clocks evolving at the same speed;
- Time efficiency of communication channel:** the messages sent are delivered instantaneously to all recipients;

**Reaction time of participants:** participants take decisions and react without delays, i.e., no noticeable computation is performed in states.

Current enterprise systems can actually offer a run-time environment in which fully sound business protocols can be efficiently executed. Enterprise service buses provide reliable communication channels (e.g., through implementations of the WS-ReliableMessaging specification). Time-efficiency requirements on the communication channel and participants' reaction time can be achieved by employing strategies from communication networks such as *Time Division Multiplexing*, while protocols like the *Internet Network Time Protocol* can be used to ensure that participants have a consistent view of time.

## 4 Related Work

Recently, a simplified version of BPEL 2.0, called BPEL<sup>Light</sup> [5], has been proposed to encode complex, executable MEPs. BPEL<sup>Light</sup> extends BPEL 2.0 with a WSDL-less interaction model that makes it possible to specify processes representing orchestration-based MEPs independently from Web service technologies. While this proposal has the advantage of being directly executable on suitable middleware, there is currently no direct support for model checking and validation.

Other DFA-based formalisms have been proposed to describe asynchronous message exchanges between two parties as orchestrations or choreographies. All these formalisms encode message exchanges as transitions connecting states of the protocol, and can ultimately be mapped to our formalism.

Two-party orchestration business protocols have been studied in [6,7]: for this particular class of business protocols, very relevant problems like compatibility and replaceability [2,8], evolution and migration strategies for running instances [9], have been already addressed.

An approach to matchmaking of two-party choreography business protocols (though not supporting time constraints) is presented in [10] as part of a search engine for ad-hoc business processes, as well as an extraction process for orchestration-based business protocols. These are called *views* and are created from choreographic protocols.

## 5 Conclusions and Future Work

Current orchestration languages (e.g., BPEL) describe how services can interact with each other at the message level from the perspective and under control of a single service. Moreover, the interactions are limited to uni-cast scenarios. This is extremely restrictive for applications characterized by wide-scale and complex dynamic interactions.

In this paper we introduced an intuitive formal model for describing multi-party service network business protocols based on Deterministic Finite Automata. In addition, we defined the notion of full soundness for multi-party

business protocols. Fully sound multi-party choreography business protocols rely solely on message exchanges as the only means of communication and can be executed consistently in a completely distributed manner, while guaranteeing termination. Our framework allows the description of business protocols and verification of their temporal properties using model checking of timed automata. Fully sound multi-party choreography business protocols contribute towards a comprehensive theory of management of business protocols for service networks.

Future work on choreography business protocols will focus on the evolution of business protocols and how it impacts time-related QoS parameters like turn around time, transaction rates, etc. Another area of work concerns protocol replaceability and compatibility analysis and versioning techniques for business protocols.

## References

1. Mancioppi, M.: A Formal Framework for Multi-Party Business Protocols. CentER Discussion Paper 2008-79, Tilburg University (September 2008), <http://center.uvt.nl/pub/dp2008.html>
2. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-Grained Compatibility and Replaceability Analysis of Timed Web Service Protocols. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 599–614. Springer, Heidelberg (2007)
3. Benyoucef, M., Keller, R.K.: An Evaluation of Formalisms for Negotiations in E-commerce. In: Kropf, P.G., Babin, G., Plaice, J., Unger, H. (eds.) DCW 2000. LNCS, vol. 1830, pp. 45–54. Springer, Heidelberg (2000)
4. Alur, R., Dill, D.L.: A Theory of Timed Automata. *Theoretical Computer Science* 126(2), 183–235 (1994)
5. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL<sup>light</sup>. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
6. Yellin, D.M., Strom, R.E.: Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems* 19(2), 292–333 (1997)
7. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. *Data Knowledge Engineering* 58(3), 327–357 (2006)
8. Benatallah, B., Casati, F., Toumani, F.: Analysis and Management of Web Service Protocols. In: Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 524–541. Springer, Heidelberg (2004)
9. Ryu, S.H., Saint-Paul, R., Benatallah, B., Casati, F.: A Framework for Managing the Evolution of Business Protocols in Web Services. In: Roddick, J.F., Hinze, A. (eds.) APCCM. CRPIT, vol. 67, pp. 49–59. Australian Computer Society (2007)
10. Wombacher, A., Mahleko, B.: Finding Trading Partners to Establish Ad-hoc Business Processes. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 339–355. Springer, Heidelberg (2002)

# Automatic Mash Up of Composite Applications

Michael Pierre Carlson<sup>1</sup>, Anne H.H. Ngu<sup>1</sup>, Rodion Podorozhny<sup>1</sup>,  
and Liangzhao Zeng<sup>2</sup>

<sup>1</sup> Computer Science Dept., Texas State University, San Marcos, TX 78666, USA  
{mc1173, rp31, hn12}@txstate.edu

<sup>2</sup> IBM T.J. Watson Research Center, Yorktown Heights, NY 10598  
lzeng@us.ibm.com

**Abstract.** The need for integration of both client and server applications that were not initially designed to interoperate is gaining popularity. One of the reasons for this popularity is the capability to quickly reconfigure a composite application for a task at hand, both by changing the set of components and the way they are interconnected. Service Oriented Architecture (SOA) has become a popular platform in the IT industry for building such composite applications recently with the integrated components being provided as web services. A key limitation of such a web service is that it requires extra programming efforts when integrating non web service components, which is not cost-effective. Moreover, with the emergence of new standards, such as OSGi, the components used in composite applications have grown to include more than just web services. Our work enables progressive composition of non web service based components such as portlets, web applications, native widgets, legacy systems, and Java Beans. Further, we proposed a novel application of semantic annotation together with the standard semantic web matching algorithm for finding sets of functionally equivalent components out of a large set of available non web service based components. Once such a set is identified the user can drag and drop the most suitable component into an Eclipse based composition canvas. After a set of components has been selected in such a way, they can be connected by data-flow arcs, thus forming an integrated, composite application without any low level programming and integration efforts. We implemented and conducted experimental study on the above progressive composition framework on IBM's Lotus Expeditor which is an extension of a Service Oriented Architecture (SOA) platform called the Eclipse Rich Client Platform (RCP) that complies with the OSGi standard.

## 1 Introduction

Composite applications are a line of business applications constructed by connecting, or wiring, disparate software components into combinations that provide a new level of function to the end user without the requirement to write any new code. The components that are used to build a composite application are generally built within a Service Oriented Architecture (SOA). Many of the first SOA platforms exclusively relied on web services (WSDL-based) as components in the composite application. The composition is done generally using process based languages such as BPEL[1] or UML statechart [2]. The web services only integration framework requires extra

programming efforts when integrating with non web service components, which is not cost effective. With the emergence of new standards, such as OSGi [3], the components used in composite applications have grown to include more than just web services. Components can be built from web applications, portlets, native widgets, legacy systems, and Java Beans.

There are challenges to mashing up non-web service components into composite applications, especially when components are developed at different times, by different groups, using different technologies, naming conventions, and structures. Firstly, a given enterprise may have hundreds of similar components available for mash up in a catalog, but manually searching and finding compatible and complementary components could be a tedious and time consuming task. For example, in a portal environment, it is possible to query the system for the available components. However, the list that is returned is typically based on criteria that have no relevance to the application assembler (e.g. alphabetical or last update time). Interfaces like Google Code Search [4] allow the developers to search application code, but it does not allow them to search using the higher level concepts of a component or a model. On the other end of the spectrum, having to manually classify and describe every aspect of components for browsing and searching can be a painstaking task when handling a large number of components.

Secondly, none of the existing OSGi environments provides a way to leverage the semantic search techniques that have been developed to assist the user in locating compatible components for web service based composite applications. Unlike web services, many non-web service components have graphical user interfaces built from technologies such as portlets, Eclipse Views, and native application windows. Moreover, there is currently no standard way of categorizing and cataloging components for use in composite application. Rather, components are discovered by assemblers who must hunt around the web, in documentation, and searching the locally installed system. This does not provide an easy and manageable means of finding and selecting components. Depending on the technology used or the type of user interface being presented, certain components may not be valid for use in a particular composite application. Discerning this could be a difficult process up front, or could result in repeated cycles of trial and error, especially when the target environment supports a variety of technologies.

After suitable components have been discovered, the assembly of composite applications should not require tedious and detailed programming as required of a typical software developer. End users, at least the savvier end users, should be able to compose applications with minimal training. For a call center in an enterprise, this may simply mean being able to assemble a composite application on the fly that takes a caller's information in one application and has the input reflected in other applications that are currently running on his or her desktop for other contextual information. For the savvy end user at home or in a small business, this may mean assembling a GPS routing application together with the list of errands or deliveries for the day and producing a more optimized route.

The main contributions of this paper are as follows. First it is shown that existing techniques, technologies, and algorithms used for finding and matching web service components (WSDL-based) can be reused, with only minor changes, for the purpose of finding compatible and complementary non web service based components for rich

client composite applications. These components may include graphical user interfaces, which are not an artifact in web service components. By building on the techniques initially developed for web services matching, the problems associated with finding useful and valid components for composite applications using high level concepts is possible. This enables the progressive construction of composite applications from a catalog of available components without deep knowledge of the components in the catalog. We demonstrated how the additional characteristics of components, specifically graphical user interface details, can be modeled, described using Semantic Annotation for web services (SAWSDL) [5] and matched in a similar fashion to the programmatic inputs of web service based components. Though similar in some respects to web service based components, our experimental study shows that these additional characteristics of a component allow for further match processing logic to be used to provide much better results for the user when searching for components to integrate into the composite application. Finally, it will be shown using sample applications and scenarios that by taking into account the unique characteristics of a component (i.e. coexistence of user interface components), and new techniques of merging semantic descriptions across multiple components, we can achieve a much more accurate search result for compatible components.

The paper is organized as follows. Section 2 outlines the overall architecture of our progressive composition framework. Section 3 details the concepts of composite application matching, merging multiple components into a single descriptive format for matching, and modeling of component's GUI characteristics. Section 4 provides a set of experiments, results, and analysis of progressive composition framework based on semantic web matching technique with SAWSDL annotations. Section 5 describes the related work and Section 6 provides the conclusion and future work.

## **2 Progressive Composite Application Framework**

### **2.1 Application Components**

The application components referred to in this paper generally contain user interfaces, built from technologies such as JFace, JSPs, HTML, Eclipse Standard Widget Toolkit (SWT), Swing, native windowing systems, etc. Like web services and Enterprise Java Beans, application components can take programmatic inputs and provide programmatic outputs. In application components, programmatic inputs will generally cause changes in the graphical user interface, and user interaction with the graphical user interface will cause the programmatic outputs to be fired. An example of an application component is a Portlet [6].

We adopt the IBM Lotus Expeditor [7] platform to develop application components and composite applications. Expeditor contains a Composite Application Infrastructure (CAI) and an associated Composite Application Editor (CAE). CAI is the runtime environment for the composite applications. It has two main components called Topology Manager and PropertyBroker. The Topology Manager is responsible for reading and interpreting the layout information stored in the composite application. The PropertyBroker is responsible for passing messages between application components of a composite application. The CAE editor is used to assemble, and wire



components into composite applications without the need for the underlying components to be aware of each other at development time. The desired components can simply be dragged and dropped to add them to a composite application. The adding, removing, and wiring can be done in an iterative/progressive fashion to allow the assembler to refine the composite application. This declarative data-flow like wiring of components is one of the main advantages of Lotus Expeditor. The wired components can be saved in an xml file and written to local file system, portal server, or Lotus Domino NSF database.

The programmatic inputs and outputs of an application component in CAI are described using WSDL. Typically the associated WSDL files for CAI components are created when the components are installed or imported into Lotus Expeditor. In the current implementation, the WSDL files for application components in CAI does not include the graphical user interface type (e.g. JSP, SWT, Swing, etc.). The composite application assembler must have previous knowledge of component interfaces they are restricted in and the types of GUI technologies they can use. For example, if the deployment platform does not provide support for Portlet interfaces, then an assembler must know which components in the repository are built from portlets and specifically avoid those when assembling the composite application.

Lotus Expeditor also did not provide a way for finding compatible and complementary components from a catalog of existing components based on components' capabilities. We extended the Expeditor Workbench with a new menu item called Analyze Composite App. which will open a dialog box for user to search for the desired components to use for composition. The following section illustrates in Figure 1 the sequence of screen shots in Lotus Expeditor Client workbench that resulted in a simple HotSpotFinder composite application.

## 2.2 A Scenario of Developing Composite Application

Screen A shows the initial composition workbench. On the right is the panel that shows the list of components (e.g. HotSpotFinder, GoogleMapper, CityStatePicker, OrderTracker) that are available for composition as well as links to available remote components. On the left is the panel that displays all the existing composite applications of a particular user (there is only one composite application available beside the one that is being composed). The middle panel displays the in-progress composite application. When the user clicks on Analyze Composite App. menu, a dialog box in screen B is displayed. After the user entered the desired search criteria at the top of screen and pressed the "Find Matches" button, screen C is displayed. By picking the best matched component (the one with the highest score) from the palette and dropped it on the middle panel, screen D is displayed. The middle panel now has two components (CityStatePicker and HotSpotFinder) which were not aware of each other. At this point, the user can right click on the in-progress composite application which will allow selection of the "wiring" action from the menu. Screen F shows the result of wiring the two selected components on the middle panel. The CityState Picker component (labeled "City View") provides a single output, labeled cityState. The HotSpotFinder component provides a single input named SetLocationCityState. The dotted line indicates that the cityState output has been linked to the SetLocationCityState input. Therefore, when the output cityState is fired, the argument of that output

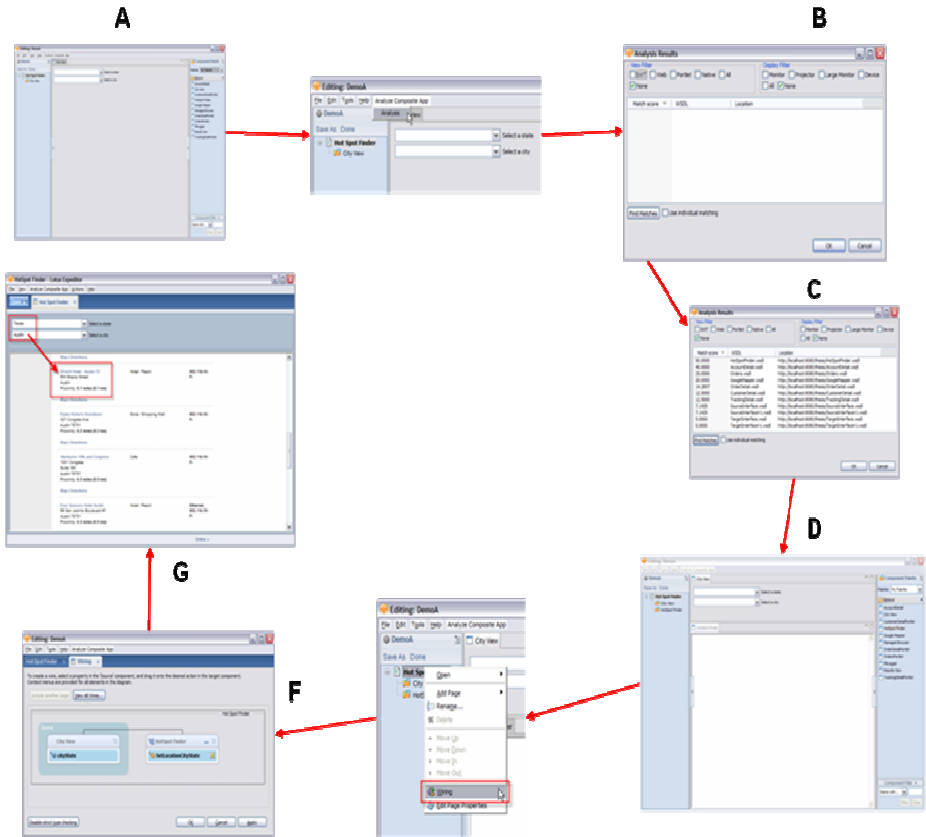


Fig. 1. Sequence of steps in composing an application

will be sent as the argument to the `SetLocationCityState` input. The composition is now completed and screen G displays the result of running the composed application within the Expeditor.

### 3 Composite Application Matching

In this section, we demonstrate techniques, technologies, and algorithms that can be leveraged to provide support for finding compatible components. The WSDL-based programmatic inputs and outputs of a component in CAI can be further modeled using semantic web techniques to enable searching using higher level concepts. Additionally, the implementation technology (e.g. SWT, Swing, etc) used to create the graphical user interface can be modeled using semantic model and added to component's WSDL to further describe the platforms that the component can support.

The additional metadata added did not change the nature of the component's description. Therefore existing semantic web services matching technologies and algorithms can be used directly for matching the application components. One such set of

algorithms is described by T. Syeda-Mahmood [8]. This work describes combining the use of semantic and ontological matching for the purposes of matching web service components. In fact, the matching code that was used to produce the results shown by T. Syeda-Mahmood is the same matching code that is used to conduct the experiments in Section 4. In brief, the matching algorithm works as follows:

1. Using a single input WSDL and a collection of target WSDLs, a score is calculated based on the number of matching terms found between the input WSDL and each target WSDL. A thesaurus, in this case WordNet [9], is used to expand the matching to include synonyms. Thus, this phase is focused on keyword matching.
2. A second search is then invoked using the same input and targets as above. This time the semantic annotations specified in the WSDL files are considered. The semantic models for each component are compared using a custom ontology matching algorithm. This algorithm takes into account the relationships between the elements given, such as inheritance, hasPart, hasProperty, etc. A score for each combination is calculated based on the number of attributes that are matched for each combination.
3. The final score is calculated using a winner-takes-all approach. The maximum of the first score and the second score is reported as the overall matching score for each input and target combination.

Even though we can reuse much of the same logic and algorithms, there are a few fundamental differences when dealing with non-web services based components. In many cases, when searching for a web service, the developer is looking for APIs that can either:

1. Match – Using the output from a single web service and finding a second web service that can take that as input. The developer can continue this process and string together several web services with application specific control-flow pattern in order to complete a business process.
2. Compose – Starting with a known output and a known input, the developer uses search techniques that can allow them to find one or more services that will transform the output of the first web service into something that can be consumed by the final web service.

The difference with respect to our composite applications is that in most cases the goal is not to put together a single business process or tightly link fragments of software processes with specific control flows; rather, the goal is to integrate separately created components together “on the glass” [10] and provide the ability for those applications to communicate or interact without prior knowledge of each other or in any specific order. This means that a composite application may bring together a human resources vacation planning component with a project management component. By linking the two applications together, the project management component could potentially use vacation data in the vacation planning component to adjust project schedules. In no way, however, does the process of scheduling vacation need to be modified in order to use the data. Additionally, the developer of each of these components does not need to have knowledge of the implementation of the other component or how to invoke the programmatic interfaces.

The markup required for composite application matching is similar to the markup required for web services, at least with respect to data inputs and outputs. Figure 2. shows an example of a CityStatePicker component with semantic annotation. A CityStatePicker component allows a user to select a valid city and a state from given lists.

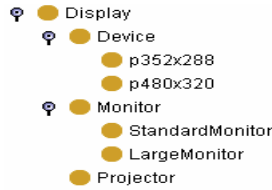
```

1<?xml version="1.0" encoding="UTF-8"?>
2<definitions name="edu.txstate.mpc"
3  targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
4  xmlns="http://schemas.xmlsoap.org/wsdl/"
5  xmlns:TravelOnt="http://localhost:8080/thesis/travel.owl"
6  xmlns:portlet="http://www.ibm.com/wps/c2a"
7  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
8  xmlns:tns="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
9  xmlns:wssem="http://www.w3.org/ns/sawsdl"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12  <types>
13    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"/>
14  </types>
15  <message name="cityState">
16    <part name="cityState" type="xsd:string" wssem:modelReference="TravelOnt#City"/>
17  </message>
18  <portType name="edu.txstate.mpc_Service">
19    <operation name="pubCityState">
20      <output message="tns:cityState"/>
21    </operation>
22  </portType>
23  <binding name="edu.txstate.mpcbinding" type="tns:edu.txstate.mpc_Service">
24    <portlet:binding/>
25    <operation name="pubCityState">
26      <portlet:action activeOnStartup="true" caption="pubCityState"
27        description="Announces the city and state" name="pubCityState"
28        selectOnMultipleMatch="false" type="standard"/>
29    <output>
30      <portlet:param boundTo="request-attribute" caption="cityState"
31        description="Published the city/state selected"
32        name="cityState" partname="cityState"/>
33    </output>
34    </operation>
35  </binding>
36</definitions>

```

**Fig. 2.** WSDL with semantic markup for CityState Picker component

Lines 5 and 9 import the necessary namespaces used to add the semantic annotations. Lines 15 – 17 define a new message named “cityState,” which defines the name of the string that will be output when user interacts with this component. On line 16 you will see that this message has been annotated with a reference to an element in a semantic model. This is shown as `wssem:modelReference="TravelOnt#City"` in the WSDL file. TravelOnt is an OWL-based semantic model that is available on the web. With this annotation, we are describing the message in terms of an OWL class in an OWL model. Continuing along the web services methodology, this message is set as an output of the “pubCityState” operation in a portType (lines 18 – 22) and included in a portlet type binding (lines 23 – 35). With this markup using the standard grammar defined by WSDL and SAWSDL, the component’s output can now be matched against other components’ programmatic inputs using existing web service matching technologies. By including the annotation, the matching engine is able to match based on capabilities of the component as described in the OWL model. For example,



**Fig. 3.** OWL model for display

assume the “cityState” component were to be compared against another component that contained the element “county.” The text-based matching would not count these as a possible match because the two strings are not equal (i.e. “cityState” != “county”). However, if the “county” element had a semantic annotation of “TravelOnt#County” in its modelReference attribute, the matching logic would be able to compare the model types City and County. If the model described a relationship between a City and a County, perhaps using the hasProperty OWL attribute, it could be determined that a city is in a county and both are part of a state. Thus, the match would score higher because the analysis would show that these two elements are very closely related.

Semantic annotations only work if there is a unified ontology model. Therefore, it may be necessary to create a semantic model for the components if none exists. We use Protégé-OWL [11] editor to create OWL-based semantic models for describing application component’s GUI characteristics since none exists. Figure 3 shows the simple GUI semantic model used in our framework. This model describes a single top-level OWL class named Display. There are three subclasses of Display, namely Device, Monitor, and Projector. Further, there are two subclasses of Device, p352x288 and p480x320. The two resolutions for the device class describe certain types of device interfaces. The p352x288 represents many Windows Mobile smart phones and the p480x320 represents Apple’s first generation iPhone. The Monitor class is further subclassed into StandardMonitor and LargeMonitor.

## 4 Experimental Study

To validate the suggested automatic mashup approach, several experiments were performed. These experiments involved using 12 components created with the help of the Java Eclipse IDE toolkit. Once created, they were installed in Lotus Expeditor. The semantic matching algorithm to be validated was implemented as part of the Analyze Composite Application (App.) in Lotus Expeditor workbench. A Lenovo ThinkPad T60p running Microsoft WindowsXP SP2 served as an experimentation platform. In the experimental test bed, the Analyze Composite App first reads the content of the current composite application. The “View Filter” and “Display Filter” sections in Analyze Composite App. enable the user to set the graphical user interface search criteria. The main table displays the score associated with each of the matching target WSDLs (i.e. component descriptions) that were in the repository. The “Find Matches” button starts the search process. The “Use individual matching” checkbox allows the user to specify which type of matching will be used. If checked, each of the

components' WSDLs in the current composite application will be matched individually to the target WSDLs in the repository. If not checked, a merged WSDL across all existing components in the current composite application will be used in the matching process. We validated the capabilities of the approach by running five experiments on two different composite applications. The first composite application that we studied is hotspot tracking and the second application is an order tracking.. Similar results is obtained from the order tracking application and thus we only report three experiments that are related to hotspot tracking application. The first experiment is to validate the fact that the semantic matching algorithm together with SAWSDL annotation indeed provides a better match result than matching without using the semantic annotation. The second experiment is to show increased accuracy in the search process when using a merged WSDL. Finally, the last experiment shows the effect of adding the GUI semantic information for the matching process.

The first experiment involves simple matching using two component WSDLs and the semantic web matching logic described in Section 3. The input for this matching scenario is CityStatePicker component that allows a user to first select a state from a select box and then select a city from a second select box. After the city is selected, the component publishes the selected city and state information. The target component is the HotSpotFinder component. This is an Eclipse SWT Browser which accesses JiWire website for a listing of wireless internet access points in the given city and state. When semantic web matching is applied to CityStatePicker component, a score of 50 for HotSpotFinder is produced. In order to show the effect of the semantic matching only, a modified version of the HotSpotFinder.wsdl is used and the same experiment is run again. In the modified version, the identifying names such as city and address are replaced with random strings. The resultant score which uses solely the ontological match is 37.50 for HotSpotFinder. This lower score is likely due to the fact that only the message elements in the WSDL file have semantic models attached to them. Lastly, we remove all the semantic annotation in the WSDL documents so that only pure keyword matching can be used. In this run, the matching score drops to 25. Additional changes to the WSDL that remove other city and state keywords while preserving its functionality cause the score to drop even more. This shows that the semantic matching algorithm is working as expected in this experiment and that we have a valid environment for conducting other experiments.

The second experiment shows the effect of using the merged WSDL search request to find compatible components for a composite application. In this case, the target WSDLs will be chosen from the complete collection of components available in our framework. In order to start the composition process, we must have a starting component. The CityStatePicker component is used as the first component. This composite application is then matched against the complete catalog of components. The results, as shown in Figure 4, tell us that the HotSpotFinder component has the highest score (50) and should be added to the application.

Once the HotSpotFinder component is added to the application, the analysis is run again using a merged WSDL search request from both CityStatePicker and HotSpotFinder. This time the highest ranking component returned is the GoogleMapper component. This component takes as input an address, and based on this address, the component loads a map for the address using Google Map to provide the actual content. In fact, not only is this component now the highest scoring component, but it has

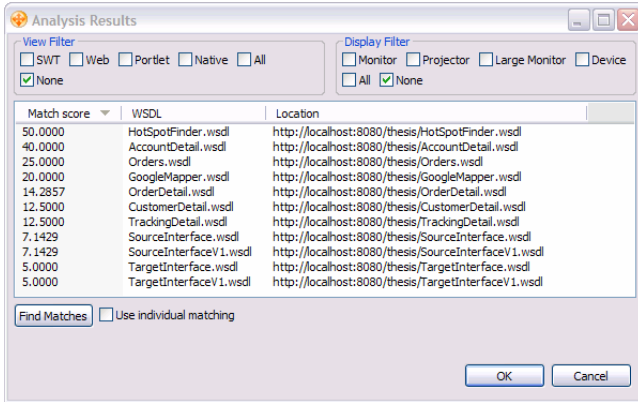


Fig. 4. Matching of CityStatePicker component

also shown a substantial jump from its previous score of 20 to the new score of 50. This result indicates that the GoogleMapper component is a good candidate to add to the application at this time. We further validated the effect of matching with a merged WSDL request in Order Tracking composite application which won't be shown here due to lack of space.

The third experiment showed the effects of using GUI information for conducting the match. We used the OrderTracking composite application to illustrate that. It is done so because this application can be composed from components with greater variety of interface technologies. In order to conduct this experiment, a new message type pertinent to interface technology will be added to the search component WSDL and each of the target components' WSDLs. For this experiment, the Customer Details component is marked as having an SWT GUI, the Order Details component is marked as having a Portlet GUI, and the Account Details is marked as having a Web Browser GUI. The test scenario is the same as in experiment two with the merged WSDL search. Initially, the Orders and Order Tracking components are selected for the composite application and a merged WSDL search is executed. By selecting one or more of the checkboxes in Analyze Composite App. search dialog for SWT, Web, Portlet, Native, or All, the merged search WSDL will be enhanced with this additional criterion. In the first run we will select SWT checkbox. The match score for the Customer Detail component has increased slightly and the scores for the other components have decreased. Customer detail was originally the best match and it remains so with this additional filter. Components that do not have GUI semantic annotation won't even appear as part of the result. This is shown in Figure 5.

Next, let us summarize the results of the experiments described above. Experiment one has shown that the function provided by existing web services matching code can be used in conjunction with composite applications. Because the matching logic uses both text-based matching and semantic matching, the function can be used without adding the semantic markup. However, as we saw in experiment one, the semantic matching always provides better results. For example, when two components are named differently yet provide the same functionality, the semantic matching is able to find the match.

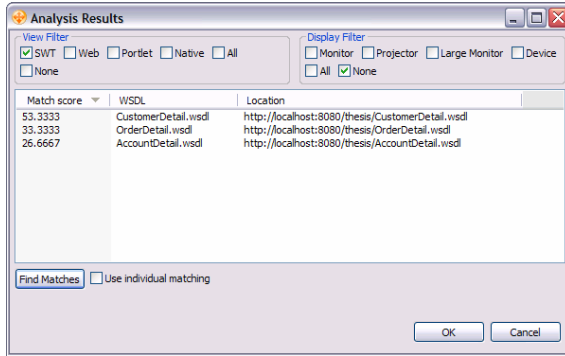


Fig. 5. Matching of components with GUI semantic annotation

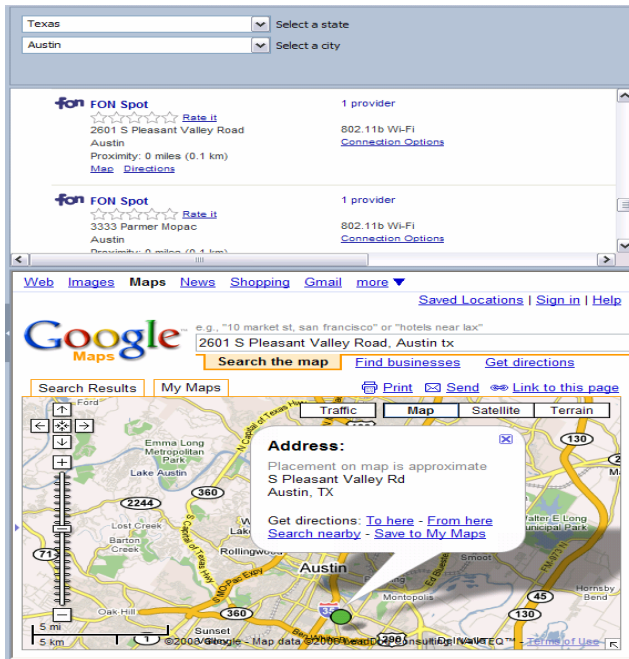


Fig. 6. Completed HotSpotFinder application

Experiment two has shown that it is possible to build a composite application from a collection of different components using semantic annotations and semantic web service matching logic. While there is no automatic way to start the building process, once a starting point is selected, the remaining compatible components begin to stand out in the repository searches. As the HotSpotFinder application was composed, the initial results did not show as much difference in scores as might be expected, but with the addition of the HotSpotFinder component, we can then construct a merged



WSDL search request. This enables the GoogleMapper component to stand out as the next obvious addition.

Experiment three illustrated that the score of a component can be impacted by the addition of the GUI semantics. It can therefore be summarized that any potential meta-data could be added to the WSDL and used in the semantic matching logic later on.

Thus, the experiments enabled us to validate the semantic service matching approach. The tool based on the approach delivered reasonable matches for service specifications thus providing appropriate functional service compositions. The picture of a completed integrated application (HotSpotFinder Application) is shown in Figure 6.

## 5 Related Work

There are several other similar approaches to automatic mashing of application compositions. Two of those include Yahoo! Pipes [12] and the various “Mashup” products, such as Intel Mash Maker [13] and Mash-o-matic [14]. While interesting in their own respects, there are limitations in these technologies that are solved through the use of composite applications running in Lotus Expeditor.

Yahoo! Pipes provides a web-based means of pulling data from various data sources, merging and filtering the content of those sources, transforming the content, and finally outputting the content for users to view or for use as input to other pipes. There are several limitations in Yahoo! Pipes. The first is the limited set of inputs and outputs. There is no way to use arbitrary inputs or outputs when using this application. There are a limited number of input types and output types from which the user can select. A component in a composite application should be able to accept many different types of inputs and provide many different types of outputs. Secondly, the flow of a pipe is static and sequential. While a user can configure many different inputs, all of the connections are executed in a sequential manner until the single output is reached. With composite applications, the different components in the application can communicate with each other in any manner that the assembler chooses. Finally, Yahoo! Pipes is a server-based technology that makes use of only a web user interface. There is no way for a user to construct and execute a pipe without a network connection and execute the pipe using locally stored data. A pipe can be accessed programmatically, like a web service, but in order to execute the pipe the user must be able to connect to the Yahoo! Pipes server. These same limitations exist in other web portal type solutions such as iGoogle and My Yahoo.

Damia [17] extends the type of data sources that can be used for mash up to enterprise types such as Excel, Notes, web services repository and XML rather than just URL based sources as in Yahoo pipes. It has a simple model of treating all data as sequences of XML. There are three kinds of main operators, ingestion, augmentation, and publication. Ingestion bring data sources into the system, Augmentation operator provides extensibility to the system. It allows creation of new mash up operators and is thus more powerful than the fixed Yahoo pipes operators. Finally, there are publication operators which transform the mashup to common output formats such as Atom, RSS or JSON for the consumption of other components. Damia is centered on data rather than component mashup. Another work with a somewhat similar purpose is the COmposer of Integrated Systems (COINS) by Mark Grechanik and Kevin Conroy

[15]. The COINS enables putting together GUI-based applications exploiting an accessibility programmable interface that manipulates the user interface components such as buttons, textfields, menu items and others. The use of the accessibility interface enables the COINS system to integrate applications that comply with the accessibility standard (section 508 of the rehabilitation act). Such a solution has lower performance than using a dedicated API, yet it significantly decreases the time needed to integrate applications (by an average of 3 times). The focus of the COINS system is on integration. They do not deal with semantic annotation of component and finding compatible components.

Kepler[16] is an open source scientific workflow system which allows scientists to compose a composite application (a.k.a workflow) based on available actors. An actor can be built from any kind of applications. However, Kepler is not based on SOA architecture and it requires very skillful low level Java programming to convert applications into actors which can be composed within Kepler framework. Kepler is also not meant to be used in a mobile rich client environment like Lotus Expeditor.

## 6 Conclusion

One of the most difficult problems faced by users in a Rich Client environment is finding compatible and complementary components in a large catalog of components that have been built by different groups, at different times, using different technologies and programming conventions and reuses those components as it is in a different application. In this paper we have demonstrated that this problem can be largely solved by applying technologies related to the semantic web and web services matching and using a progressive composition framework like Lotus Expeditor. The first technology that can be applied is Semantic Annotations for WSDL (SAWSDL), as standardized by the W3C. By adding semantic model references to the message elements of the WSDL, the properties exposed by the component can be better described using modeling languages. Since the modeling attributes can be added to any elements of the WSDL, the definition of the component could be further refined and described using the concepts of SAWSDL.

The second technology group that can be applied is the searching and matching algorithms created for use with web services. These algorithms provide a powerful method for scoring the compatibility of an application component from a large set of possible component choices based on component capabilities. This scoring simplifies the application creation process for the composite application assembler by providing a ranking of potential components. This allows the assembler to focus on the highest ranked components, skipping over the lower ranked components, when considering which items may be compatible in the application being created.

The searching process is further improved based on the fact that a composite application can be viewed and described as a single component when searching against a repository of components. This is done by creating a merged WSDL from each of the component of the composite application. As seen in the experiment results, the use of individual matching may still be valuable, especially when attempting to distinguish between components that score very closely to each other. A potential improvement to the analysis results window would be to display the score for each target

component using both the merged matching and individual matching, when the collection of scores is relatively close.

A larger direction of future work is combining a service mashup approach with a process based integration approach. A semantically rich process language with constructs for conditionals, iterations and methods for insuring reliability of an integrated application can facilitate more complex combination of a larger set of applications. It can also help analysis of an integration specification for general properties such as lack of a deadlock or a cycle and problem domain specific properties such as compliance of an integrated application to a set of business rules.

## References

1. <http://www.ibm.com/developerworks/library/specification/ws-bpel/>
2. <http://www.uml.org>
3. OSGi originally stood for Open Service Gateway Initiative. This term is no longer used and the alliance is now known simply as OSGi., <http://www.osgi.org>
4. <http://www.google.com/codesearch>
5. <http://www.w3.org/TR/sawsdl>
6. Portlets as implemented in the Java programming language are defined by JSR 168, <http://jcp.org/en/jsr/detail?id=168>
7. <http://www.ibm.com/software/lotus/products/expeditor>
8. Syeda-Mahmood, T.S., Akkiraju, R.I.A., Goodwin, R.: Searching Service Repositories by Combining Semantic and Ontological Matching. In: Third International Conference on Web Services, ICWS (2005)
9. Miller, G.: WordNet: A lexical database for the English language. Communications of the ACM (1983)
10. Phifer, G.: Portals Provide a Fast Track to SOA. Business Integration Journal (November/December 2005)
11. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé-OWL Plugin: An Open Development Environment for Semantic Web Applications (2004), <http://Protege.Stanford.edu/plugins/owl/publications/ISWC2004-protege-owl.pdf>
12. <http://pipes.yahoo.com/pipes/>
13. <http://softwarecommunity.intel.com/articles/eng/1461.htm>
14. Murthy, S., Maier, D., Delcambre, L.: Mash-o-matic. In: Proceedings of the 2006 ACM Symposium on Document Engineering, pp. 205–214 (2006)
15. Grechanik, M., Conroy, K.M.: Composing Integrated Systems Using GUI-Based Applications And Web Services. In: IEEE International Conference on Services Computing, SCC 2007 (2007)
16. Kepler Project: <http://Kepler-project.org/>
17. Simmen, E.D., Altinel, M., Padmanabhan, S., Singh, A.: Damia, data mashups for intranet applications. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1171–1182 (2008)

# Non-desynchronizable Service Choreographies

Gero Decker<sup>1</sup>, Alistair Barros<sup>2</sup>, Frank Michael Kraft<sup>3</sup>, and Niels Lohmann<sup>4</sup>

<sup>1</sup> Hasso-Plattner-Institute, University of Potsdam, Germany

`gero.decker@hpi.uni-potsdam.de`

<sup>2</sup> SAP Research Centre, Brisbane, Australia

`alistair.barros@sap.com`

<sup>3</sup> SAP AG, Walldorf, Germany

`frank.michael.kraft@sap.com`

<sup>4</sup> Institut für Informatik, Universität Rostock, 18051 Rostock, Germany

`niels.lohmann@uni-rostock.de`

**Abstract.** A precise definition of interaction behavior between services is a prerequisite for successful business-to-business integration. Service choreographies provide a view on message exchanges and their ordering constraints from a global perspective. Assuming message sending and receiving as one atomic step allows to reduce the modelers' effort. As downside, problematic race conditions resulting in deadlocks might appear when realizing the choreography using services that exchange messages asynchronously. This paper presents typical issues when desynchronizing service choreographies. Solutions from practice are discussed and a formal approach based on Petri nets is introduced for identifying desynchronizable choreographies.

## 1 Introduction

The service oriented architecture (SOA) is an architectural style for building software systems based on services. Services are loosely coupled components described in a uniform way that can be discovered and composed. One realization of a SOA is the web services platform architecture where services are offered as web services [1].

In a first generation of services only pairs of request/response message exchanges were considered. This view is sufficient when considering simple services, for instance, a stock information service, where the current or a past value of a share can be requested. However, more complex interactions must be considered in many real-world business scenarios. For instance, in a typical purchasing scenario, goods can be ordered, orders can be modified or canceled, orders must be acknowledged and delivery can be rerouted, or alternative products or quantities are offered in out-of-stock situations. Also multi-lateral scenarios involving, for instance, external payment, shipment and insurance services need to be considered. These scenarios involve multiple interactions and the complex dependencies between them must be addressed.

Service choreographies are a means to specify the messages exchanged between different services and the dependencies between them. Even multi-lateral scenarios can be captured with typical languages such as the Web Service Choreography Description Language (WS-CDL [2]). Here, interactions between services are the atomic building blocks and ordering constraints are defined between them. That is, sending and receiving of messages are modeled as one step. The ordering constraints then define what other interactions must have occurred before a certain interaction. In the remainder we will refer to this modeling style as *interaction modeling*.

Interaction modeling, as opposed to distinguishing message sending and reception as separate activities in the control flow, results in several advantages for the modeler. (i) Control flow dependencies do not need to be specified per service, but rather as seen from the perspective of an ideal observer. That way redundant control flow relationships are avoided and the chance of modeling deadlock situations is minimized. Furthermore, avoiding redundant structures allows for faster model creation. (ii) Branching structures can be specified globally, that way avoiding modeling errors caused by incompatible branching structures [3].

As a downside of interaction modeling, the intuitive interpretation that all interactions are atomic steps hides certain challenges that arise when taking the choreography to an asynchronous world. Especially in situations where there exists a mutual exclusion between two interactions with different senders, i. e. *mixed choices*, the asynchronous nature of message exchanges might cause severe problems due to race conditions.

This paper addresses the issue of race problems in asynchronous settings. It discusses typical solutions in real-world applications and presents a formal framework for detecting and locating race conditions in service choreographies.

The remainder of this paper is structured as follows. The next section will present a real-world example where mixed choices actually lead to deadlocks. Section 4 lists and discusses typical solutions in practice, before Sect. 3 introduces a formal framework for identifying and locating problematic race conditions. Section 5 reports on related work in this area and Sect. 6 concludes.

## 2 Motivating Example

A purchase order process looks as follows. A buyer submits an order to a seller. The seller returns a confirmation message to the buyer. Finally, delivery and payment are carried out. The buyer and the seller are both realized as services.

Several situations require a deviation from this simple process. While the seller has not sent a delivery notification and the buyer has not initiated payment yet, the buyer has the possibility to issue a change request, e. g. demanding an increased quantity. A change request must be acknowledged by the seller. The buyer might as well cancel the order which in turn needs to be confirmed by the seller. On the other hand, there might be a seller initiated change proposal, e. g. the previous confirmation is revised by proposing a delay of the delivery date.

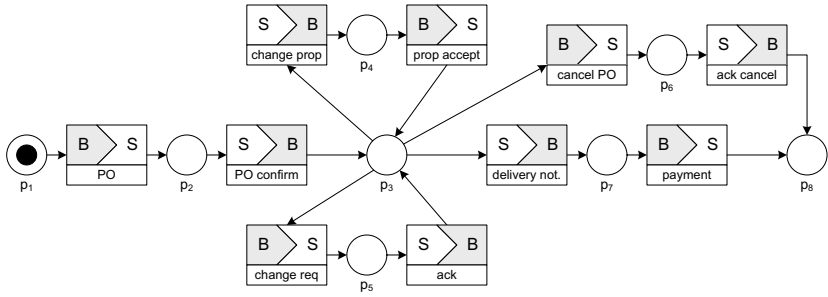


Fig. 1. Purchase order choreography

Figure 1 illustrates these interactions using the interaction Petri net notation from [4]. Each rectangle represents the message exchange between a sender (upper left corner) and a receiver (upper right corner). The label at the bottom of a rectangle indicates the message type. The circles represent places that can contain tokens. The token flow defines the control ordering constraints between message exchanges belonging to the same choreography instance.

The interaction Petri net assumes a world where message send and receive happen in one atomic step. This assumption is not valid in many scenarios. In our example, the different services might send messages concurrently. For example, the buyer’s and seller’s decisions to send change requests or change proposals are decoupled and therefore it is a common scenario that the services send messages concurrently. Therefore, we need to desynchronize the choreography to properly reflect that message sending and receiving are separate steps. Figure 2 shows a corresponding Petri net.

The original purchase order model does not contain any obvious problems. Each choreography instance eventually terminates in a state where there is one token left on place  $p_8$  of the net. The desynchronized model, on the other hand,

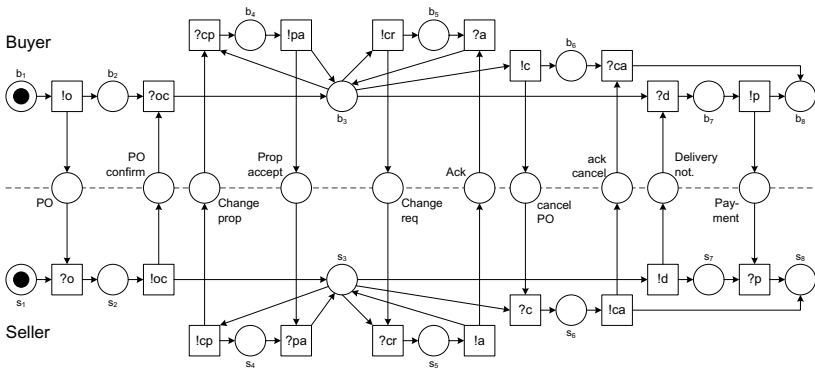


Fig. 2. Desynchronized purchase order choreography

contains several problems. For instance, a deadlock occurs if the buyer decides to cancel the order and the seller proposes a change. Here, the buyer would wait infinitely for a confirmation of the cancellation and the seller waits infinitely for the response to his proposal.

The issues presented in this example are not specific to purchase ordering scenarios. Similar issues can be found in many other areas of enterprise systems. In order to detect and locate such race problems, we introduce a formal approach in the next section.

### 3 Formal Model

This section will introduce a formal framework for detecting and locating problematic race conditions in service choreographies. The framework builds upon Interaction Petri nets (IPNs), a special kind of labeled Petri nets, where a transition models the message exchange between two services. IPNs have been proposed in [4]. While the example presented in Sect. 2 is bilateral, IPNs can also be used to represent multi-lateral choreographies, i. e. more than two roles are involved in the choreography. IPNs have the same token flow semantics as classical Petri nets [5] and concentrate on the control and message flow aspect of choreographies.

**Definition 1 (Petri Net).** *A Petri net is a tuple  $[P, T, F, m_0]$  where  $P$  and  $T$  are two finite disjoint sets of places and transitions, respectively,  $F \subseteq ((P \times T) \cup (T \times P))$  is a flow relation, and  $m_0 : P \rightarrow \mathbb{N}$  is an initial marking.*

We assume the standard firing rule of Petri nets, and write  $m \xrightarrow{t} m'$  if the marking  $m'$  is reachable from marking  $m$  by firing transition  $t$ . Reachability can be canonically extended to transition sequences.

Message types are first-class constructs in IPNs, allowing the distinction between, for example, acceptance and rejection messages. In the following definitions, we denote the set of all roles by  $R$ . A concrete service participating in a choreography instance plays one or several roles, for example, “buyer” or “seller”. The set of message types is denoted by  $MT$ .

**Definition 2 (Interaction Petri Net).** *An Interaction Petri net (IPN) is a tuple  $N = [P, T, F, m_0, final, \lambda]$  where*

- $[P, T, F, m_0]$  is a Petri net,
- *final* is a finite set of valid final markings, and
- $\lambda : T \rightarrow (R \times R \times MT) \cup \{\tau\}$  is a labeling function assigning a sender role, a receiver role and a message type to a transition, or declaring it as silent transition.

The IPN in Fig. 1 has two roles  $R = \{B, S\}$ , ten message types  $MT = \{PO, PO\ confirm, \dots, payment\}$ , and  $[p_8]$  (i. e., one token on place  $p_8$ ) is the only final marking. With the help of final markings, we can differentiate desired final states from unwanted deadlocks. This can be expressed with the concept of *weak termination*.

**Definition 3 (Weak Termination).** *An interaction Petri net weakly terminates iff, from every marking reachable from  $m_0$ , a final marking  $m_f \in \text{final}$  is reachable.*

It can be easily checked that the IPN in Fig. 1 weakly terminates; that is, the marking  $[p_3]$  can be reached from all reachable markings.

The following definition bridges service choreographies and the local views on such a choreography, i. e. the subset of interactions and control flow dependencies that are relevant for a given role.

**Definition 4 (Role Projection).** *Let  $N_1$  and  $N_2$  be interaction Petri nets.  $N_1$  is the projection of  $N_2$  for role  $r$  iff  $\forall t \in T_1 : \lambda(t) = \tau \vee r \in \lambda(t)$ , and there exists a relation  $Q$  between the markings of  $N_1$  and  $N_2$  such that:*

- i.  $m_{01} Q m_{02}$ ,
- ii. if  $m_1 Q m_2$ ,  $m_1 \xrightarrow{t}_{N_1} m'_1$  and  $r \notin \lambda(t)$ , then  $m'_1 Q m_2$ ,
- iii. if  $m_1 Q m_2$ ,  $m_1 \xrightarrow{t}_{N_1} m'_1$  and  $r \in \lambda(t)$ , then  $m_2 \xRightarrow{t}_{N_2} m'_2$  and  $m'_1 Q m'_2$ ,
- iv. if  $m_1 Q m_2$ ,  $m_2 \xRightarrow{t}_{N_2} m'_2$  and  $r \in \lambda(t)$ , then  $m_1 \xRightarrow{t}_{N_1} m'_1$  and  $m'_1 Q m'_2$ ,
- v. if  $m_1 Q m_2$  and  $m_1 \in \text{final}_1$ , then  $m_2 \xRightarrow{t}_{N_2} m'_2$  and  $m'_2 \in \text{final}_2$ ,
- vi. if  $m_1 Q m_2$  and  $m_2 \in \text{final}_2$ , then  $m_1 \xRightarrow{t}_{N_1} m'_1$  and  $m'_1 \in \text{final}_1$ ,

where  $m \xRightarrow{t} m'$  denotes that there exists a (potentially empty) firing sequence of transitions  $t_1, \dots, t_n$  from  $m$  to  $m'$  where for all  $t_i$  holds  $r \notin \lambda(t_i)$  and  $m \xrightarrow{t} m'$  denotes  $m \xRightarrow{t} m'$ . Furthermore,  $r \in \lambda(t)$  denotes that  $r$  is the sending or receiving role of  $t$ .

Role projection ensures that the order of communication actions possible in the local view correspond to the order of interactions as specified in the choreography. By considering the branching structures, role projection goes beyond trace comparison. The relation is similar to branching bisimulation [6], while there are key differences: Whenever branching within a service depends on a choice done by other services, no internal decision must be made. On top of that, role projection relates the sets of final markings.

The projection algorithm in [4] preserves role projection. This algorithm projects a choreography by applying transformation rules on the structure of the interaction Petri net.

We assume that a service choreography is *realizable* [7,4]. This ensures that there is indeed a set of local views that, assuming synchronous communication, collectively show exactly the behavior as specified in the choreography.

An IPN  $N_s$  can be “desynchronized” to a net  $N_a$  using the role projections of  $N_s$ . Thereby, we introduce a place  $p_\alpha$  for each message type  $\alpha$  of the IPN  $N_s$  which models an asynchronous message channel and is connected to the sender and receiver according to the roles. Whereas communication is atomic in  $N_s$ , the sending and receipt of a message is  $x$  explicitly modeled by two transitions  $!x$  and  $?x$  of  $N_a$ .

**Definition 5 (Desynchronized Net).** *Let  $N_s$  be an IPN and  $R = \{r_1, \dots, r_n\}$  the set of all roles involved in  $N_s$ . The IPN  $N_a$  is a desynchronized net for  $N_s$  iff, for all  $i = 1, \dots, n$ ,  $N_i$  are pairwise disjoint role projections of  $N_s$  for roles  $r_i$ , and*



- $P_a = \bigcup_i P_i \cup \{p_\alpha \mid \exists t \in T_s : (\lambda(t) \neq \tau \wedge \alpha = \lambda(t))\}$ ,
- $T_a = \bigcup_i T_i$ ,
- $F_a = \bigcup_i F_i \cup \{(t, p_\alpha) \mid \exists x, mt (\lambda(t) = (r_i, x, mt))\} \cup \{(p_\alpha, t) \mid \exists x, mt (\lambda(t) = (x, r_i, mt))\}$ ,
- $m_{0a} = m_{01} \oplus \dots \oplus m_{0n}$ ,
- $final_a = \{m_{f1} \oplus \dots \oplus m_{fn} \mid m_{fi} \in final_i\}$ , and
- $\lambda_a(t) = \lambda_i(t)$  iff  $t \in T_i$ .

The composition of markings is defined as  $m_1 \oplus \dots \oplus m_n(p) = m_i(p)$  iff  $p \in P_i$ .

The desynchronized net  $N_a$  usually has more behavior than the original IPN  $N_s$ : The atomic message transfer in  $N_s$  can be mimicked by  $N_a$  by firing first the sending and then the receiving transition. Moreover, it might also be possible that  $N_a$  can fire a transition in an intermediate state introduced by the decoupling of sender and receiver. If this additional behavior does not jeopardize weak termination,  $N_s$  is *desynchronizable*.

**Definition 6 (Desynchronizability).** Let  $N_s$  be a weakly terminating IPN.  $N_s$  is desynchronizable iff there exists a desynchronized net  $N_a$  for  $N_s$  that weakly terminates.

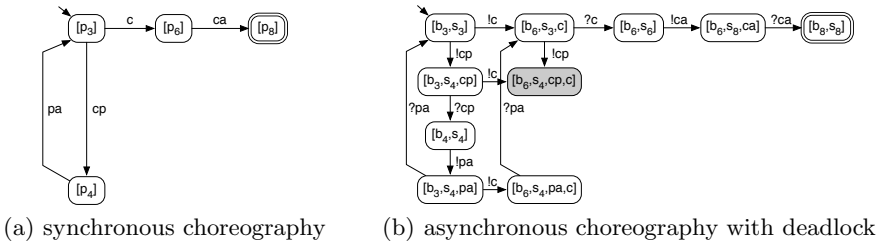


Fig. 3. Reachability graphs showing the example's race problem

The most frequent reason for non-desynchronizable choreographies are mixed choices, where there is a conflict between transitions with different senders. As mentioned earlier, the desynchronized example choreography (see Fig. 2) contains a deadlock. Thus, the original choreography in Fig. 1 is not desynchronizable. This can be detected by analyzing the reachability graphs of the nets. In Fig. 3(a), a part of the reachability graph of the original choreography is depicted. In the marking  $[p_3]$  the transitions  $c$  and  $cp$  are (among others) enabled. The same situation is depicted in Fig. 3(b) for the desynchronized net. Here, the transitions  $!c$  and  $!cp$  are enabled in  $[b_3, s_3]$ , but can occur concurrently: neither transition disables the other, and a deadlocking marking  $[b_6, s_4, cp, c]$  is reachable when firing these transitions in either order.

**Definition 7 (Conflict Transitions).** Let  $N_s$  be a non-desynchronizable interaction Petri net and  $N_a$  a desynchronized net for  $N_s$ . Define the set of conflict transitions  $T_C$  to contain all transitions  $t$  of  $N_a$  such that:

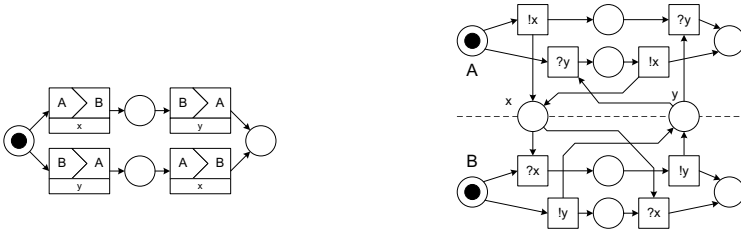


Fig. 4. Structural conflicts do not necessarily lead to deadlocks

- there exists a marking  $m$  with  $m \xrightarrow{*} m_f$  for a marking  $m_f \in final$ , and
- there exists a marking  $m'$  with  $m \xrightarrow{t} m'$  and  $m' \not\xrightarrow{*} m'_f$  for any  $m'_f \in final$ .

The set  $T_C$  contains all transitions whose firings can make a final marking unreachable. From Fig. 3(b) we can conclude that the transitions !c and !cp are conflict transitions for the desynchronized net of Fig. 2. With state-of-the-art Petri net model checkers such as LoLA [8], race problems can be detected efficiently even for larger choreographies.

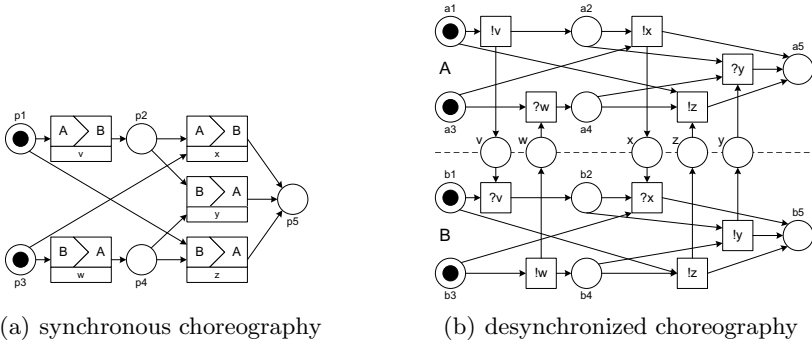
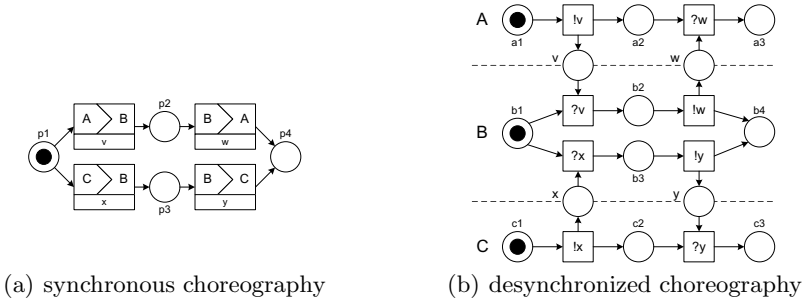


Fig. 5. Conflict transitions without structural conflict

While mixed choices are a typical reason for race problems, not all mixed choices are problematic (see Fig. 4). Here, both services can send a message before receiving one. However, due to the follow-up interactions, the desynchronized choreography weakly terminates.

Finally, conflict transitions do not necessarily need to be in a structural conflict, i.e. sharing common input places. Figure 5(a) shows a synchronous choreography that weakly terminates with final marking [p5]. Here, the choice whether the transition regarding message v or the transition regarding w fires first influences what transitions will be enabled later on. If the transition involving v fires, the transition involving z will not be enabled any longer.

The definition of conflict transitions also captures those scenarios where individual services are able to send messages in a final marking. Figure 6(a) shows



**Fig. 6.** Deadlocks caused by firing transitions in a valid final marking

an example involving three roles and final marking [p4]. Figure 6(b) shows the desynchronized choreography as generated by the algorithm in [4]. The final markings for the individual role projections are [a<sub>1</sub>] and [a<sub>3</sub>] for A, [b<sub>4</sub>] for B and [c<sub>1</sub>] and [c<sub>3</sub>] for C. This results in the valid final markings [a<sub>1</sub>, b<sub>4</sub>, c<sub>3</sub>], [a<sub>3</sub>, b<sub>4</sub>, c<sub>1</sub>], [a<sub>1</sub>, b<sub>4</sub>, c<sub>1</sub>], and [a<sub>3</sub>, b<sub>4</sub>, c<sub>3</sub>] for the desynchronized choreography, while only the first two markings are actually reachable. In marking [a<sub>1</sub>, b<sub>4</sub>, c<sub>3</sub>] role A is ready to fire transition lv. Firing this transition actually leads to a marking from where no valid final marking can be reached any more. Therefore, lv is a conflict transition.

### 4 Typical Resolutions to Race Problems

The definitions from the previous section enable us to locate conflict transitions. As a next step, one or several strategies have to be chosen to remove race problems from a choreography. Instead of formally proposing one particular strategy or defining an algorithm to automatically choose among different possible strategies, we rather sketch several strategies applied in real-world implementations in this section. Each strategy comes with a set of implications on the business level that need to be carefully considered before applying them.

In the remainder of this section we will use the term *conflicting messages* for a pair of messages sent by different partners that correspond to a pair of conflict transitions as defined in the previous section. Both messages must belong to the same choreography instance.

It could be a possible strategy to resolve the choreography in such a way that conflicting messages simply cannot occur any longer. For the example, this would mean that there is no chance of having a delivery notification and a cancel message been sent in the same choreography instance. This could be achieved for instance through total sequentialization of the choreography, where only one partner is allowed to send messages at a time. However, such a strategy is typically not feasible in real-world scenarios. It is often desired that conflicting messages are possible but for the case that this occurs, a predefined resolution must be in place. Therefore, we are going to list different strategies applied in practice that follow this approach.

The following strategies can be categorized into two groups. Either (a) there is a predefined outcome upon conflicting messages, most typically one message is considered and the others are ignored, or there might be different outcomes possible. Here, we can again distinguish three types: (b) one partner could be allowed to determine the outcome and tell the other partners his decision; it could also be defined that (c) each partner decides individually for the outcome, or that (d) there is a negotiation regarding the desired outcome.

#### 4.1 Precedence

The general idea is to define precedence relationships at design-time, prescribing how partners have to behave in the case of conflicting messages. Therefore, precedence mostly falls into category (a). If a partner detects conflict messages, he knows the outcome of this conflict and can immediately continue accordingly. He assumes that the other partners will also detect this conflict sooner or later and also act accordingly.

The definition of precedence relationships must not be seen as pure technicality as it directly has business impact. Therefore, precedence relationships would need to be part of interaction contracts. Regarding the definition of precedence relationships we distinguish three different strategies.

**Singular Interaction Partner Precedence.** This strategy looks at individual interactions, e.g. the cancellation interaction in our example, and defines precedence of one partner over the other. Here, we can distinguish between two settings: (i) the buyer has precedence over the seller or (ii) the seller has precedence over the buyer.

Case (i) means that if the buyer sends the cancellation, the seller has to accept the buyer's cancellation in any case. This means that the buyer can assume that the cancellation message will have the desired effect, once it has been sent. Therefore, the seller does not need to return any confirmation message in this case. This corresponds to category (a).

In case (ii) the seller has a veto right regarding cancellation messages sent by the buyer. The seller can accept this request and return a cancellation confirmation. Only now the buyer can be sure that cancellation was successful. As an alternative, the seller could also send a cancellation rejection. Therefore, the seller can decide on the outcome, implying category (b).

Deciding for each interaction for a partner precedence individually does not solve race problems in the general case. If, for example, we decide that the buyer has precedence regarding buyer initiated cancellation and the seller has precedence regarding seller initiated change proposals, deadlocks are still possible. Now imagine the opposite setting where the seller has precedence regarding buyer initiated cancellation and the buyer has precedence regarding seller initiated change proposals. Here, the partners have veto rights for the corresponding requests. If the buyer sends a cancellation request and the seller sends a change proposal at the same time, the buyer will reject the change proposal as it conflicts with the previously sent cancellation request. The same holds true for the

seller reacting to the cancellation request. After both partners have rejected the respective requests, they can, of course, resend their requests.

**Type-Based Precedence between Multiple Interactions.** While the previous strategy considered interactions individually, this strategy considers precedence regarding combinations of interactions. Here, the message types are considered and always a fixed outcome is defined, therefore category (a). A crucial aspect of this strategy is that no combination of interactions is forgotten.

A precedence rule could be that a delivery notification has precedence over buyer initiated cancellation messages and buyer initiated change requests. On the other hand, a buyer initiated request always has precedence over seller initiated change proposals. Figure 7 illustrates a resolved desynchronized choreography for this precedence rule. This resolved version weakly terminates. All transitions that were added to the original Petri net with striped background.

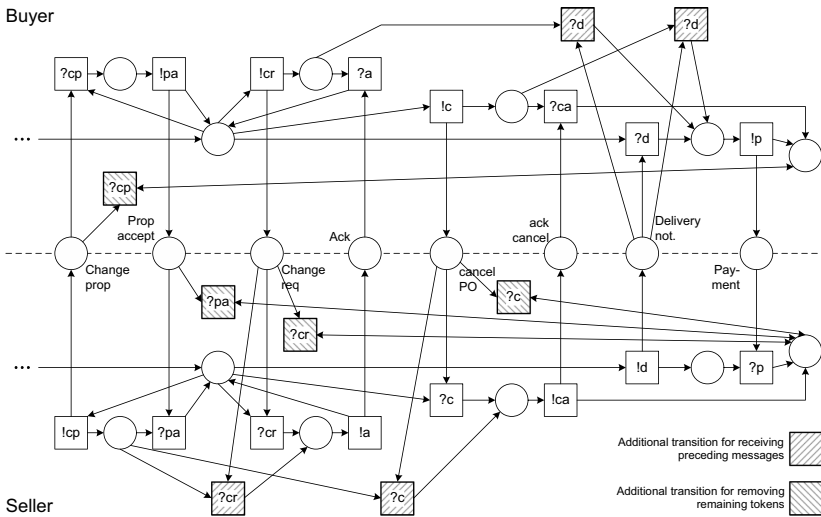


Fig. 7. Resolved purchase order choreography

A conflict between a seller initiated change proposal and a buyer initiated change request is resolved in the following way. In addition to being ready to consume an acceptance message for the change proposal, the seller can also consume a change request or a cancel message instead. This is manifested through the additional transitions ?cr and ?c transitions. The change proposal message must finally be consumed by the buyer without having any effect on the buyer. This happens through the additional ?cp transition.

The proposed solution in Fig. 7 is still not optimal from a business point of view. If the seller sends a change proposal while the buyer sends a change request, the messages conflict and the seller will receive the change request and accept

it. In this situation, the seller assumes that the buyer will ignore the change proposal. However, the buyer could receive the accept message first and receive the change proposal afterwards. Now, the buyer cannot know that this change proposal conflicted with the change request and therefore accepts the proposal. However, the seller is not able to receive this message and will only remove the remaining token at the end of the choreography. From a business point of view this behavior is undesired: the buyer has accepted a change proposal that the seller assumes obsolete.

Another problem might arise when precedences are cyclic: Imagine there are three partners A, B and C. A can send a message to B (interaction  $ab$ ), B to C ( $bc$ ), and C to A ( $ca$ ).  $bc$  has precedence over  $ab$ ,  $ca$  has precedence over  $bc$ , and  $ab$  has precedence over  $ca$ . Now a conflict involving all three interactions occurs. Every partner thinks that his message has precedence over the message received and simply ignores the incoming message. This again could result in a deadlock.

**Situation-Based Precedence between Multiple Interactions.** While precedence rules between different interactions were based on message types, this strategy allows more fine-grained precedence rules and again falls into category (a).

Imagine a logistics scenario where a customer lets a shipper transport his goods. The shipper selects different carriers and creates a shipment that he sends to the customer and which needs to be commented by the customer. At the same time, the customer has the possibility to cancel his order. While cancellation precedes the shipment plan interaction in the default case, this is only true for the first two weeks after the initial order. After these two weeks have passed, the shipment plan interaction precedes the cancellation. This might be due to the fact that cancellation at this point in time would simply involve too much cost. However, while the shipment plan has not been finalized yet, the customer can still cancel the order.

An underlying assumption of this strategy is that both partners come to the same conclusion about precedence. As time is the criterion in this example, both need common understanding about when the two weeks have passed. Therefore, the arrival time of the message cannot be used as criterion, as the corresponding sender might not be able to know when this is.

## 4.2 Allowing Individual Decisions

Allowing individual decisions leaves it open to every partner involved to decide for an outcome individually: category (c). In the case of a buyer initiated cancellation request conflicting with a seller initiated change request, there are two alternatives for each partner:

- The seller either (S1) rejects the cancellation request and assumes that the change request has still relevance or (S2) accepts the cancellation request and assumes that the change request is obsolete.

- The buyer either (B1) rejects the change request and assumes that the cancellation request has still relevance or (B2) accepts the change request and assumes that the cancellation request is obsolete.

Out of these possibilities two are ideal outcomes: The combinations (S1)+(B2) and (S2)+(B1) lead to the acceptance of exactly one request. Even the combination (S1)+(B1) is acceptable, as the choreography instance is exactly in the same state as before the two requests and requests can be issued again. Maybe this time, one of the partner succeeds with his intent.

Only the combination (S2)+(B2) is problematic as both requests were accepted and both partners assume a wrong situation. However, once an accept message finally arrives, the conflict is detected and a resolution can be achieved as described in the other strategies.

Although this strategy does not guarantee a proper resolution in the general case and requires resorting to other resolution strategies, it is still worth considering as most outcomes are acceptable. A major challenge of this approach is that the process instances need to be realigned in case a partner has already continued, assuming his decision led to an acceptable situation. This might involve compensation and becomes especially difficult if communication to other partners is involved.

### 4.3 Negotiation of Outcome

Negotiation is another strategy where the outcome of conflicting messages is not fixed, therefore category (d). Here, the different partners need to reach agreement about the outcome. Such negotiation can either happen through human intervention or automatically. Human intervention could simply involve a phone call or an email exchange. In many cases such human intervention is actually desirable. For instance, the cost of cancellation might increase depending on what actions the partner has already performed. Therefore, it could be negotiated whether cancellation is still desired under the new conditions.

As an alternative, a formal hand-shake to support negotiation could be factored into each partner's process. For this, all partners need to agree on conflicting messages requiring negotiation and implement common exception handling logic. This would involve strictly sequential interactions, as partners arbitrarily reciprocate to resolve the conflict. First, conflicting messages would be detected by a partner and be broadcast to relevant partners. Each partners process would be required to escalate to its common exception handling logic such that all parts of the process impacted by the conflict are suspended. The first part of the exception handling logic would be to determine which partner gets the write token. This remains an open issue although some basic heuristics could be defined, e.g. the first partner detecting the conflicting messages gets the write token. Another serious issue is managing multiple conflicts which can arise concurrently and determining the priority in which they should be handled. These and other issues have been handled in techniques applied for self-stabilizing systems [9].

## 5 Related Work

Different service choreography languages have been proposed that follow the interaction modeling style. The Web Service Choreography Description Language (WS-CDL [2]) is a standard proposed by the World Wide Web Consortium. Alternative proposals from academia are Let's Dance [10] and the Interactive Systems Description Language (ISDL [11]). The issue of race problems when taking choreographies defined in these languages to an asynchronous world has not been tackled so far.

There are different formal models available for describing choreographies. A survey can be found in [12], where a distinction between automata-based, Petri-net-based and process-algebra-based approaches is made. Most approaches include techniques for relating choreographies to models describing the behavior of individual services. For instance, [13] use a bisimulation-like relation to check conformance between a local model and the choreography.

There has been extensive research in the area of compatibility checking, where the absence of deadlocks is of central importance, e.g. [14,15,16,17]. While detecting and locating deadlocks is covered by most approaches, more novel approaches deal with the question of automatically repairing faulty choreographies [18]. While such approaches could indeed be used to repair the choreography presented in Sect. 2, any outcome would include forbidding certain messages. Either the buyer must not send a change request or a cancellation message, or the seller must not send a change proposal or a delivery notification. Such a solution is primarily aimed to explain faulty choreographies by proposing fixed versions, but would, of course, be unacceptable from a business point of view.

The issue of desynchronizability is closely related to the question of synchronizability of asynchronous choreographies [19]. If an asynchronous choreography is synchronizable then the same set of choreography instances are produced under asynchronous and synchronous communication semantics. If applied in a top-down manner, i.e. a synchronous choreography is projected to an asynchronous choreography, synchronizability can be used to detect race problems as presented in this paper. However, our approach goes beyond synchronizability analysis as it allows to locate the reasons of desynchronizability. This is important as there might be different sets of conflict transitions that might be treated in isolation of each other, which in turn might allow for a less-invasive resolution.

The problem of mixed choices between sending and receiving has been studied in the context of distributed message protocols and algorithms [20]. For example, the *crosswalk algorithm* adds round numbers to each sent message which help to identify and solve conflicts. Again, such protocols do not take the content and the original choreography into account and thus are not suitable to solve the problem from a business point of view.



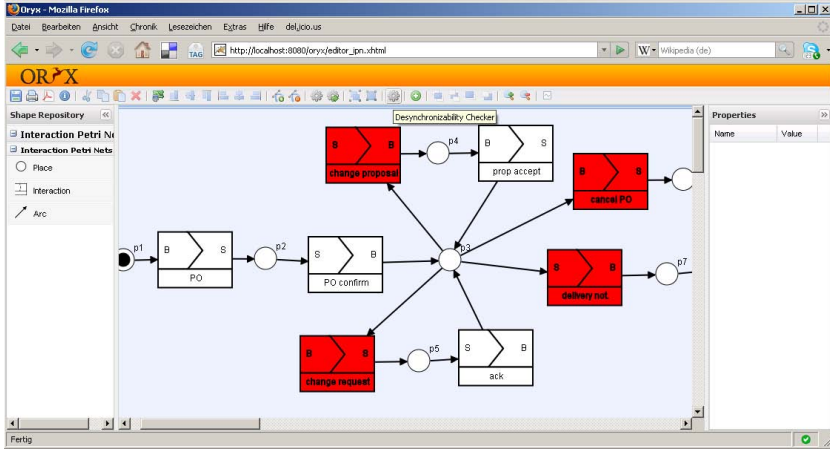


Fig. 8. Screenshot of the interaction Petri net modeler and desynchronizability checker

## 6 Conclusion

This paper has discussed the issue of non-desynchronizability in choreographies where message send and receive activities are considered as atomic steps. We introduced a formal approach for detecting and locating potentially conflicting message exchanges. This approach is based on interaction Petri nets.

We have implemented a tool that realizes this approach. As an extension to the web-based modeling platform Oryx<sup>1</sup>, interaction Petri net stencils and a plugin for desynchronizability checking were developed. Figure 8 shows a screenshot of the tool, where the conflict transitions of the initial example are highlighted in red and bold labels. In future work, we will extend the tool chain to desynchronizability analysis for iBPMN choreographies. iBPMN is an extension for the Business Process Modeling Notation (BPMN [21]), allowing for interaction modeling in a BPMN-like notation [3].

The resolution of race problems is not a pure technical issue that can be carried out late in the actual development of services. The discussion of typical resolution strategies in Sect. 4 has shown business implications when choosing one strategy or another. Therefore, precedence relationships, for instance, would have to be discussed and defined in very early choreography design stages.

The current solution to non-desynchronizable choreographies looks as follows: An interaction model is created, then desynchronizability is verified as presented in Sect. 3. If desynchronizability is detected the desynchronized choreography can directly be used as starting point for implementing services or adapting existing ones. If this is not the case, manual resolution along one of the resolution strategies has to be carried out for the desynchronized choreography first.

<sup>1</sup> See <http://oryx-editor.org>.

One might argue that we do not need interaction models all together as we have to resort to asynchronous models anyway (in the case of non-desynchronizability), and we should rather use asynchronous models from the beginning. However, the advantages gained for the desynchronizable parts of the choreography are already immense. Having the possibility to generate the desynchronized model increases modeling speed.

A classification of different resolution strategies shows the vision for dealing with non-desynchronizability. Ideally, modelers can choose from a predefined set of strategies, being informed about the business implications of a chosen strategy. Such a declarative approach would finally result in generating fully resolved choreographies. With such an approach the modelers would not need to touch the generated model any longer. Therefore, future work will center around formally refining the different strategies and proposing a declarative framework for the resolution of race problems.

## References

1. Curbera, F., Leymann, F., Storey, T., Ferguson, D., Weerawarana, S.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Englewood Cliffs (2005)
2. Kavantzaz, N., Burdett, D., Ritzinger, G., Lafon, Y.: *Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation*. Technical report (2005), <http://www.w3.org/TR/ws-cd1-10>
3. Decker, G., Barros, A.: *Interaction Modeling using BPMN*. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) *BPM Workshops 2007*. LNCS, vol. 4928, pp. 208–219. Springer, Heidelberg (2008)
4. Decker, G., Weske, M.: *Local enforceability in Interaction Petri Nets*. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007)
5. Reisig, W.: *Petri Nets*. Springer, Heidelberg (1985)
6. van Glabbeek, R.J., Weijland, W.P.: *Branching time and abstraction in bisimulation semantics*. *J. ACM* 43(3), 555–600 (1996)
7. Fu, X., Bultan, T., Su, J.: *Conversation protocols: A formalism for specification and analysis of reactive electronic services*. *Theor. Comput. Sci.* 328(1-2), 19–37 (2004)
8. Schmidt, K.: *LoLA: A Low Level Analyser*. In: Nielsen, M., Simpson, D. (eds.) *ICATPN 2000*. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000)
9. Dolev, S.: *Self-stabilization*. MIT Press, Cambridge (2000)
10. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.H.M.: *Let's Dance: A language for service behavior modeling*. In: Meersman, R., Tari, Z. (eds.) *OTM 2006*. LNCS, vol. 4275, pp. 145–162. Springer, Heidelberg (2006)
11. Quartel, D., Dijkman, R., van Sinderen, M.: *Methodological support for service-oriented design with ISDL*. In: *Proc. ICSOC 2004*, pp. 1–10. ACM, New York (2004)
12. Su, J., Bultan, T., Fu, X., Zhao, X.: *Towards a theory of web service choreographies*. In: Dumas, M., Heckel, R. (eds.) *WS-FM 2007*. LNCS, vol. 4937, pp. 1–16. Springer, Heidelberg (2008)

13. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration: A synergic approach for system design. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 228–240. Springer, Heidelberg (2005)
14. Martens, A.: Analyzing web service based business processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
15. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: Proc. WWW 2004, pp. 621–630. ACM, New York (2004)
16. Canal, C., Pimentel, E., Troya, J.M.: Compatibility and inheritance in software architectures. *Sci. Comput. Program.* 41(2), 105–138 (2001)
17. Puhlmann, F., Weske, M.: Interaction soundness for service orchestrations. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)
18. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–247. Springer, Heidelberg (2008)
19. Fu, X., Bultan, T., Su, J.: Synchronizability of conversations among web services. *IEEE Trans. Softw. Eng.* 31(12), 1042–1055 (2005)
20. Reisig, W.: *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer, Heidelberg (1998)
21. OMG: *Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification*. Technical report, Object Management Group, OMG (2006)

# A Framework for Semantic Sensor Network Services

Lily Li and Kerry Taylor

CSIRO ICT Centre  
GPO Box 664, Canberra, ACT 2601, Australia  
{lily.li, kerry.taylor}@csiro.au

**Abstract.** We propose that a semantic service-oriented approach is one of the best techniques to cope with challenges in wireless sensor network (WSN) applications. This paper offers a framework for sensor network services that aims to improve query processing. We expect this framework will address current challenges and issues preventing the wider uptake of WSN technology. More specifically, we propose a semantic service-oriented framework with a focus on query processing to allow distributed end-users to request streams of interest easily and efficiently, based on the principle of pushing the query down to the network nodes as much as possible. As such, the lifetime and utility of the sensor network will be maximised, ultimately leading to the success of WSN deployments. The importance of semantics, which aims to support sensor capability modelling and query writing has been highlighted. On the other hand, query rewriting is emphasised followed by examples to illustrate that query rewriting can significantly contribute to the overall power efficiency of WSNs.

## 1 Introduction

Sensor network applications have been deployed for monitoring space, things and the interactions of things with each other and the encompassing space [1]. Future sensor networks will have sensors with different capabilities wired into a Sensor Web [2] to perform extensive monitoring for timely, comprehensive, continuous and multi-modal observations [3].

Despite existing different domain-specific deployments and the heterogeneity of sensor networks and sensing data, sensor networks share a common feature: they all collect and periodically transmit information from some set of sensors, and they all must carefully manage limited power and radio bandwidth to ensure that essential information is collected and reported in a timely fashion [4]. From a data storage point of view, one may think of a sensor network as a distributed database that is able to conduct query processing. This has led to the design and implementation of query processing techniques, a very popular topic in the literature [4,5,6]. However, query processing in sensor networks works differently to that in a traditional DBMS given that the former is highly resource constrained (e.g. limited energy, memory and computation ability). Furthermore, the volume of the sensing data can be very variable (e.g. depending on sampling rate

variations) within a time period and the access to this data may be delayed arbitrarily due to the motion of sensors. These issues create new challenges for query processing and more needs to be done to realise the potential for sensor network applications. Further, as successful sensor network applications rely heavily on effectively using a large range of heterogeneous data resources including streaming data acquired from sensor networks and historical data stored in databases, the development of a technology framework to enable this is highly demanded.

The primary contribution of this paper is an identification of the services that may be offered to the user community of WSNs through a semantic service-oriented framework. We focus in particular on dealing with query processing elegantly through that framework. Employing both declarative semantics and query rewriting techniques are important features of the framework.

It is natural to choose a declarative language [4,7,8] to describe a query, as declarative queries offer both an easy-to-use interface and energy-efficient execution substrate [4]. More importantly, they open up the possibility for optimisation algorithms to transparently handle efficient access strategies and performance improvements enabled by taking account of current state of the sensor network including concurrently or previously executed queries.

The paper is organised as follows. Section 2 is the brief review of related work. Section 3 outlines the design challenges and issues. Section 4 presents our framework that relies on a service-oriented computing approach enriched with semantic modelling. In Section 5, the main features of the framework, semantic query rewriting will be discussed followed by two examples to illustrate the significant impact of query rewriting on the overall power efficiency of WSNs. Finally, we conclude this paper in Section 6.

In this paper, terms such as sensor networks and wireless sensor networks will be used interchangeably unless otherwise specified.

## 2 Related Work

Sensor networks promise to revolutionise sensing in a wide range of application domains. However, wide acceptance and deployment has not yet been seen mainly because current deployments are closed, single-purpose systems. Tools and standards for easy open access are still unseen. It is well known that programming sensor networks is very hard for end-users who have genuine application requirements for sensor networks. On the other hand, end-users should be shielded from the low-level details and difficulties of sensor networks. While most of the research work in sensor networks to date has been focused on device engineering design and communications and networking questions, interesting work in the near future is most likely to be concentrated on leveraging end-users' workloads in sensor network applications by Web services in a publish-find-bind service-oriented fashion. End-users must be supported for easy access and more control over the sensor network, especially with real time data manipulation.

This section presents recent work that attempts to ease non-trivial programming and reprogramming sensor network tasks.

Different types of queries in sensor networks have been studied in [9,10]. TinyDB [6] is one of the most successful works in sensor network query processing. It offers a SQL-like query interface to raw sensor data. It is advantageous to express queries to a sensor network at a high level logical predicate language as such an abstraction has greatly eased the use. The objective of this paper is to make this abstraction more concise by taking query rewriting techniques into account.

Cougar [8] adds a **query layer** to the protocol stack to accept queries in a declarative language that is then optimised to generate efficient query execution plans with in-network processing. A query plan is constructed by **flow blocks**. The optimiser determines the exact number of flow blocks and interaction between them. However, as pointed out in [8], the creation of the best query plan for an arbitrary query is a hard problem, only little work in query processing has been done in Cougar.

Several other pioneering works have set good examples for the use of sensor services in the presence of Web services and Semantic Web technologies. Existing prototype applications have shown that the heavy burden of programming sensor networks can be alleviated if the underlying infrastructure is flexible enough. Web service technologies are one such enabler that can be used to derive more value from sensor data by making sensor data more accessible to a wider group of people, when combined with services for statistical modelling and machine learning, for example. However, the current state of the art is far from the vision of a highly available, high-performance, easy-to-use sensor web. The following is a brief introduction to these relevant works.

Sensor Web [2], first proposed by NASA in 2001, has received tremendous attention. It is a revolutionary concept toward achieving collaborative, coherent, consistent and consolidated sensor data collection, fusion and distribution. The project GeoSWIFT (<http://sensorweb.geomatics.ucalgary.ca/default.html>) is an exemplar. It offers open geo-spatial sensing services for sensor web developed by sensorWeb@GeoICT [3]. It aims to build a geo-spatial infrastructure to connect distributed sensor networks for the sharing, access, exploitation, and analysis of sensing information. It focuses on a web service approach to answer a request (i.e. HTTP GET request) in an SOA style. However, it does not address efficient query rewriting, and its query language (conjunctive attribute-value style language) relies on a primitive data model that is not amenable to interoperability with other services thus it suffers from common problems of WSNs.

IrisNet [11] (<http://www.intel-iris.net/>) is a general-purpose software infrastructure that supports tasks common to services such as collecting, filtering, and combining sensor feeds, and performing distributed queries (via sensor agent nodes and organising agent nodes). It provides an opportunity to deal with a query over the Internet. It proposes that its sensing service is capable of collecting filtered sensor readings in a database that end-users can query. Although the IrisNet architecture allows filtering code to be uploaded to sensing devices, there

is no discussion about sampling control. In order for a sensor web architecture to be aware of energy consumption, an energy efficient design approach should be emphasised. Furthermore, semantics has not be thoroughly explored in IrisNet.

In contrast, SONGS [12][13] advocates semantics by proposing a semantic-service-oriented sensor information system. The system may take advantage of application domain knowledge to optimise its resource utilisation in collecting, storing, and processing data. As a result, the creation of a semantic information hierarchy and the implementation of the semantic transformations are at the core of their work. However, this work focuses on the use of a larger semantically-rich system of sensors, into which sensor devices are embedded, but does not address the requirements for general purpose sensor network services that support such embedding of WSNs.

Similar efforts from the grid computing research community [14] are worth noting too. The focus of the reported work is on sensor resource management [15] to provide middleware support to the connection and share of heterogeneous sensor resources. The work reported at the website ([http://nicta.com.au/research/projects/nicta\\_open\\_sensorweb\\_architecture](http://nicta.com.au/research/projects/nicta_open_sensorweb_architecture)) is such an example.

From a standardisation perspective, the efforts from OpenGIS Consortium (<http://www.opengeospatial.org/>) is worth mention. A package of standards called Sensor Web Enablement (SWE) has been developed to define service-oriented interfaces to sensor network services, most strongly influenced by requirements arising for earth observation remote sensing services. The SWE specifications support data and service interoperability at a syntactic level: client tools are specifically designed to parse the standardised data model, and interpretation of the content is left entirely to service providers on one hand and end-users on the other. In order to evaluate these specifications, we deployed SWE services over WSN data collected at an experimental site in Queensland and conducted an evaluation in the context of the OGC testbed programme. SWE service implementations from 52North were used (available from <http://52north.org/>). Referring to the most relevant specification, the Sensor Planing Service (SPS) for example, we found that it suffered from limitations such as an inability to model the relationship between input and output parameters. The current SPS permits neither the phrasing nor the answering of query such as “Provide the parameter  $A$  if parameter  $B$  is greater than 5 and parameter  $C$  is prior to 2 hours from now”; the kind of query we take for granted in modern query languages like SQL. More information about this testbed programme can be found at the website (<http://www.opengeospatial.org/>).

To our knowledge, no work has been reported in WSNs that considers the logic structure of the problem, sensor capability and environment setting in query rewriting. This is the first time that query rewriting has been addressed in sensor service design.

We believe that the success of applying semantic web services to WSN applications will significantly contribute to the growing deployment of large-scale sensor networks, and leading to a broad scope in sensor network applications.

### 3 Design Challenges

We discuss issues in the design of semantic sensor network services in this section. As this paper is focused on providing semantic sensor network services to service clients from a broad network community, we avoid discussing typical WSN internal issues such as network connectivity, MAC protocols and routing algorithms, but concentrate on issues related to the service design. In particular, we aim to hide those WSN-specific issues from the broader network users, and propose embedding query rewriting techniques within the sensor network service to achieve this. We do not discuss broader web service issues here (such as security, scalability and quality of service) as we anticipate that the progress of web service research will address these issues in a more general context.

We expect a sensor network service to be the (technical) custodian of a sensor network: to be responsible for local network management, data management, query processing and response, and information security. We expect the service to accept high level requests for data and for the service to be able to map that request to answers that the sensor network can handle, while enabling the service user to be as ignorant as possible about the sensor network capability, technology, topology, control language and current state. This “ignorance” will enable service clients, such as simple web pages, specialist user-oriented interactive GUIs and composition and integration engines (including workflow engines) to interact with a wide range of such services in a common way. Note that for our purposes we are focusing on sensor services, not actuator services, although we include within our scope the tasking of sensors in order to take measurements – for example the movement or rotation of a sensor in order to take a measurement. In the latter case, the need for movement is only implicit within the request for data and is not a separately identified request for actuation.

#### 3.1 Data Persistence

In common with the design principles for TinyDB [6], we believe that the management and archiving of sensor network data is the responsibility of a unitary sensor network service, rather than some external services. This enables the scope of a service to be represented and understood by its user community irrespective of the temporal nature of information. It also means that quality of service guarantees (such as reliability, response time, cost etc.) can be offered according to user priority, financial incentive, or other features in a uniform way.

When a sensor network service accepts a query it will need to recognise whether it is capable of answering some or all of the query, and if so to determine whether the answer could be partly or fully retrieved from persistent data hosted by the service, presumably previously collected from the sensors controlled by the service’s network. Alternatively, the query may be partly or fully answered by the service recognising that its network is already configured to collect the desired data and all it needs is to ensure is that the data is returned to the new requestor (possibly summarised or filtered first). Finally, there may be a



remaining part or all of the request that can only be answered by reconfiguring or reprogramming the sensor network.

We envisage this capability to be offered by a query-rewriting algorithm. By describing persistent data and currently collecting data as “views”, rewriting techniques such as [16] can be used to split an incoming query into the appropriate components to be handled by each aspect of the service.

### 3.2 Sensor Network State

Along with management of data persistence arising from sensor network measurements, we would like to see a sensor network service managing internal sensors and network state while offering a simpler stateless service for its clients. This will make service interaction easier for the clients, and enables optimisation within the service to meet multiple client requests. For example, for mobile sensors we would prefer a sensor request to state the spatial coordinates (and if necessary, temporal coordinate) required for a measurement, and have the sensor network service internally plan the optimal movement of the sensor devices to meet multiple requests, resolving conflicts by a priority-based scheduling method if necessary. Furthermore, we have already proposed that the service recognises requests for reuse of data which the sensor network is previously configured to collect: this capability could also be seen as recognition and management of sensor network state.

### 3.3 Events and Responses

As hinted in the discussion about the need to offer coherent access to persistent data, some requests for data from the service will be answerable with a synchronous response. This might be appropriate for a query for last year’s average daily rainfall, for example. Some requests will naturally behave more as a standing query (e.g. a request for next year’s average daily rainfall) and may be more appropriately dealt with as an asynchronous response. Further, an event-driven response (e.g. daily rainfall for the next three years, a day at a time) will also be needed. More generally, it seems that any query may require a combination of these approaches and an appropriate interaction design will be required.

### 3.4 Programming

Many modern sensor network technologies support over-the-air sensor node programming (e.g. the TinyOs Deluge protocol [17]). A sensor network service should be able to accept tasks from wider networks users for deployment onto the network. However, in line with our desire to hide heterogeneity we suggest a declarative, data-oriented interface to the wider user community is more appropriate than a specialist imperative programming language. We are working to develop a translation process from a high level declarative predicate language

phrased over a “view”, to the executable Snlog language<sup>1</sup> of the DSN architecture [7].

With the declarative language, it is possible to solve the problems yet unsolved by IrisNet. In particular, we can take account of local knowledge about the current state of the sensor network, such as network topology, residual power, alternative routing protocols, local redundancy, and node-specific sensing capability to push queries into network nodes for efficient execution. Results demonstrating this are reported later in this paper.

### 3.5 Capability Modelling

Whilst we wish to enable a semblance of homogeneity of sensor networks to an outside user for ease-of-use, inevitably some fundamental differences in sensor networks will remain and must be made visible to the user community to enable their use. We expect that the capability of a network in terms of the observable phenomena and how they might be measured must be declared by the sensor network to its clients. Although this recognition of need is aligned with the approach adopted through the OGC’s Sensor Web Enablement suite of standards (<http://www.opengeospatial.org/projects/groups/sensorweb>), we propose that machine executable formal ontologies, such as those based on the W3C’s Web Ontology Language (OWL) are the appropriate tools for describing sensor network capability. By employing reasoners<sup>2</sup>, we need to rely less on both specialised web clients (in which the knowledge of the capability is embedded in code) and human readability (in which the knowledge relies entirely on a human interpretation) [18].

### 3.6 Power Management

Power management is a critical issue in a WSN. The typical power management design goal is to meet the required constraints while minimise the energy consumed. Sensor applications have to make trade-offs based on the energy consumption policy (Fig. 11) between the quality of a service (e.g. sensor operations) versus conserving energy, efficiency versus power consumption. It is in the best interest of power management to have each node transmit at the lowest possible power while preserving network connectivity but activating only the necessary number of sensor nodes for a particular task at any particular moment. It is expected that a dedicated power consumption policy should be enforced and met for a specific task.

### 3.7 QoS

The QoS provisioning, usually described very abstractly, is a crucial issue to provide demanded resources effectively and efficiently. Resource reservation and

<sup>1</sup> Snlog is a dialect of Datalog developed for sensor network programming. Details and examples can be found at the website (<http://db.cs.berkeley.edu/dsn/>).

<sup>2</sup> For example, Racer (<http://www.racer-systems.com/products/tools/index.phtml>) and FaCT++ (<http://owl.man.ac.uk/factplusplus/>).

resource allocation should be enforced aligning with either single query or continuous query and the unique characteristics of WSNs. The QoS specification should also be able to offer differentiated QoS and data quality to different users.

### 3.8 Security

We cannot propose a shared sensor network service in the absence of addressing the needs for sensor network security. Assuming the sensor network itself offers some local methods for securing information within its scope (for example, [19]), a sensor network service must be able to protect its privacy and integrity in the wider internet context. For a shareable, reusable sensor network we propose Web Service-style role based access control over XML structures [20] coupled with executable privacy policies [21]. Privacy policies would apply to both persistent and real-time data, and to both data access and to scarce resource access (e.g. permission to program the sensor network to collect new measurements). They could be compiled to a run-time policy-enforcement point within the sensor network service.

## 4 Framework

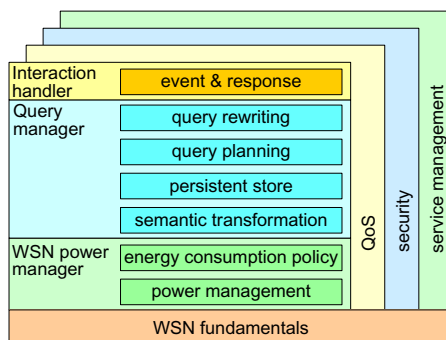
In response to the challenges and partial solutions we have discussed in Section 3, we propose a framework (Fig 1) in this section, relying on a service-oriented computing approach enriched with semantic modelling. We will address the important features in the following.

### 1. Service-oriented computing approach

The design of the proposed framework is based on the principle of service-oriented computing as we expect web service technologies would support high performance, scalability, reliability and availability of sensor services. The core of the sensor service is the service proxy, which is composed mainly of the **interaction handler**, **query manager**, and **WSN power manager**. Major components of each layer are shown in Fig. 1.

The semantic sensor service, which acts as an interface between the WSN and the client, is in charge of the interactions between them (with **interaction handler**). Then it performs query processing (including **query rewriting** and **query planning**, with the support of **persistent store** and **semantic transformation**) and partial results integration - back to the client ( by **query planning**). The success of the above functionalities cannot be achieved without concerning about scarce power resource. **WSN power manager** is designed to cope with energy management enforced by an appropriate **energy consumption policy** and corresponding **power management strategies**. Obviously, the **WSN fundamentals** is the foundation on that our work is built on.

The extended framework also includes overarching concerns such as **QoS**, **security**, and **service management** that apply to all components in the



**Fig. 1.** The Semantic Sensor Network Service Framework

framework, in that **service management** refers to the management of sensor service life-cycle, including service registration, service invocation, and service closure.

## 2. Semantic approach

Sensor Web clients depend on registries and ontology repositories to make use of available sensor services. The repositories provide necessary information about the underlying sensor networks in terms of the phenomena, observation, measurements, sensor capabilities and different models to be used to facilitate the decision making when certain events occur. These repositories can be classified into two categories with the static category providing essential information about sensors (e.g. sensor capabilities) and sensor networks (e.g. sensor platforms), while the dynamic category reflecting the change of the sensor networks, topological change, for example.

Generally, an ontology provides a medium for capturing and reusing the knowledge and experience gained from prior efforts, it thus leads to a greater level of automation at the semantic level. It is believed that ontologies and related semantic techniques and technologies will allow machines to interpret and understand (human-) agreements and formal policies one day in the future. As OWL is a highly sharable and reusable form of knowledge representation language, (sensor) ontologies will be represented in OWL. These ontologies can encode necessary information for query rewriting. A sensor network service should be able to describe what it can do and how to do it (i.e. sensor capability modelling) in terms of a formal OWL ontology. The recognition of need to provide a powerful “virtual device” (depending on available sensing devices) with differing sensor capabilities for different tasks (e.g. modelling or tasking) has attracted our attention. It will be included in the future work.

Different layers in the framework work together to maintain the integrity of sensor data and answer queries in an efficient way. Like other web services, the semantic sensor network service can be made available to be discovered and composed to support a wide variety of WSN deployments.

## 5 Query Rewriting

In this section, we will discuss query processing through the proposed framework, in which employing both declarative semantics and query rewriting techniques are important features.

By applying optimisation techniques during query processing, query rewriting can generate a rewritten query with minimised node transmission and therefore improved energy conservation. Below we first introduce declarative semantics, then unification-based propagation optimisation [22,23]. It will be illustrated by working through examples. A discussion which points out that the optimisation technique is particularly advantageous to some kinds of problems in query processing will follow.

### 5.1 Declarative Semantics and Query Rewriting

It is generally accepted that some sort of knowledge ( the knowledge about sensors and the sensor network) can provide necessary information in query processing. The knowledge consists of the sensor hardware and software characteristics. The following code fragment (represents information about the current configuration of a sensor network) is an example in the WSN programming language, Snlog [7].

```
part 1:
-----
% initial messages of nodes
toTransmit(@1,0).
toTransmit(@2,0).
toTransmit(@3,0).
toTransmit(@8,0).

% the residual powers of node,
residualPower(@1,509). %509units.
residualPower(@2,245).
residualPower(@3,455).
residualPower(@8,505).

% nodes' list
mgsList(@1, [0,2,4]). % {0,2,4} node IDs
mgsList(@2, [12,15]).
mgsList(@3, []).
mgsList(@8, []).

% timer setting of nodes
timerx(@1,2,2048). % 2 minutes
timerx(@2,2,2048).
timerx(@3,2,2048).
timerx(@8,2,2048).
```

Now consider a sensor network program that relies on the configuration to perform a task including message routing details, as follows.

**Example 1:** “Generate a sequential number and transmit it to one of its neighbours (e.g. *@Next*), update its node list (the list of nodes that this particular node has sent messages to) and cost accordingly.”

Suppose we have a top-level clause (i.e. the clause (1)) and the relevant clauses defined as follows. The whole program is composed of parts 1 and 2.

part 2:

```

-----
message(@Next,Src,Dest,X) :- generatedMgs(@Src,X), updateNode(@Src,X,Result),
                             nextHop(@Src,Dest,Next).                               %... (1)
generatedMgs(@Src,Y) :- toTransmit(@Src,X), Y is X+1,residualPower(@Src,Z), Z>500,
                        timer(@Src,2,TimePeriod).                                 %... (2)
timer(@Src,TimerID,TimePeriod) :- timerx(@Src,TimerID,TimePeriod).              %... (3)
updateNode(@Src,X,Result) :- mgsList(@Src,OldList), append(OldList,[X],Result),
                             forward(@Src,Result,Cost).                          %... (4)
forward(@Src,Result,NewCost) :- neighbour(@Src,Neighbours),
                               shortestPath(@Src,Node,Neighbours,Cost),
                               length(Result,Size), timer(@Src,2,TimePeriod),
                               NewCost is (Size*TimePeriod).                       %... (5)
-----

```

- clause (1): the predicate *message/4* is the logic consequence if the current node (i.e. *Src*) with the generated message is *X* and the routing detail (i.e. *nextHop/3*) is determined. That is, the next node (i.e. the *Next*) to send the message to is certain. At the same time, the node list will be updated accordingly.
- clause (2): the predicate *generatedMgs/2* is defined as a increment of the message determined by the predicate *toTransmit/2* if the residual power of the node *@Src* is greater than 500 units within the given *TimePeriod*.
- clause (3): the predicate *timer/3* is defined to convert a *timerx* tuple into a *timer* tuple,
- clause (4): the predicate *updateNode/3* is defined to update the node list. More specifically, the newly generated message will be appended to an existing list (i.e. the *OldList*) and the result (i.e. *Result*) will be sent away to a particular node defined by the predicate *forward/3*.
- clause (5): the predicate *forward/3* is defined to update the cost. The new cost equals to the product of the length of the list (i.e. the size of the *Result*) and the time period (i.e. *TimePeriod*).

For illustration purposes, the predicates *shortestPath/4*, *neighbour/2* and *nextHop/3* are defined as built-in predicates in Snlog.

The observation that the larger a program is, the more energy it may consume drive the selection of optimisation techniques in this situation to reduce redundancy. Two major approaches are available to be chosen from. One is “local computation” in which the code will be injected into the nodes as it was and the node itself is responsible for the computation (i.e. no specialisation). The other is “global computation” in which the code will be specialised before it is sent into a node. Our question is: “ is there any difference between these two approaches when the power consumption of a node is concerned? ”.

Now let us look at the “global computation” scheme. We are interested in using unification-based propagation technique [22,23] for it has potential to make a considerable improvement in terms of the performance because it is capable of reducing most of redundant computation at compile-time so that the computation complexity at nodes can be lessened or lifted greatly. We will not go to the detail in this paper, but highlight the effect of this technique in achieving efficiency instead. A prototype was developed to allow the rewriting to be

achieved in an automatic manner. A systematic way about how this optimisation technique is performed will be discussed in another paper.

With the unification-based propagation technique, the top-level clause will be specialised into the following particular code:

```
the specialised code (Note that the variables have been renamed by the system):
-----
message(@_G1247,1,_G1244,1):- neighbour(@1,_G1253), shortestPath(@1,_G1261,_G1253,_G1263),
                               nextHop(@1,_G1244,_G1247).
message(@_G1210,8,_G1207,1):- neighbour(@8,_G1216), shortestPath(@8,_G1224,_G1216,_G1226),
                               nextHop((@8,_G1207,_G1210).
-----
```

This program entirely replaces both part 1 and part 2 given earlier. It is specialised from part 2 to take account of the configuration in part 1 but is much more compact.

**Example 2:** We now consider a generic program to answer a class of problems. It is: “Find regions with *sensorId* > *\$IntVar* (e.g. *\$IntVar* = 99) and temperature rise of *X%* (e.g. 20%) in the last given time period (e.g.  $\Delta t$ )”.

Assuming a time window is denoted by  $[t - \Delta t, t]$ , and the memory at each node is sufficient to hold data streams during that period. One of critical steps is to avoid sending back unnecessary messages as much as possible and to detect unwanted messages as early as possible because the message size is a very important factor in power consumption. As such, it is clear only the sensors with *sensorId* > *\$IntVar* will be considered further. These sensors will then check the required temperature readings. Again, only the successful node will be asserted to the *location/1*, which is defined to hold the node information. The total cost of answering this particular query is approximated by the sum of costs at each node to check whether it is the required sensor *ID* (*ID* = 0 at the base station).

- The top-level clause is defined as follows:

```
go(@Next,Id) :- zone(@Src,Id,TH:TM:TS,TimePeriod,Lb,Ub,Percent),NextHop(@Src,Dest,Next).
```

- the predicate *zone/7* is denoted by

*zone(HostId,Id,EndTime,TimeDiff,LowerBound,UpperBound,Percent)*. It is a relation which contains a tuple for each node. It is defined as follows:

```
zone(@Src,Id,TH:TM:TS,T,Lb,Ub,Percentage):-
    sensorId(@Src,Id),less(Lb,Ub),
    check_range(Ub,Id,Lb),
    reading(@Src,Id,TH:TM:TS,TempVal_1),
    integer(T),
    T1 is TH*3600+TM*60+TS-T,
    T2H is T1 // 3600,
    T2 is T1 mod 3600,
    T2M is T2 //60,
    T2S is T2 mod 60,
    reading(@Src,Id,T2H:T2M:T2S,TempVal_2),
    TempVal_1 >= f_mult(TempVal_2,Percentage), % a built-in function in Slog
    assert(location(Src)).
```

- the predicate *reading/4* is a relation which contains tuples of sensor readings at each *Src* in the form of *reading(@Src,Id,TimeStamp,TemperatureVal)*. *TimeStamp* is expressed in the form of *H:M:S*.
- the predicate *nextHop(@Src,Dest,Next)* is a built-in.

Given the readings at all nodes (these readings have been omitted here due to space limitation), in this example, the top-level clause is specialised by unification-based program to: `go(@_G1022,12):-nextHop(@12,_G1025,_G1022)`.

This is because among available nodes, only node 12 meets the requirement. Clearly, the specialised top-level clause is very concise (no irrelevant information, only need to deal with the built-in *nextHop/3*). The full advantages of the query rewriting technique will be discussed next.

## 5.2 Discussion

Generally speaking, the specialised top-level clauses have the following advantages over the original ones:

- the length of the specialised code is greatly shorter than that of the original one.

Intuitively, this means fewer rules/facts will be fired for the same problem. The reason behind is that much of the computation has been performed thanks to the optimisation technique. Those predicates whose definitions are available at compile-time, such as the residual power of nodes, should provide sufficient information to instantiate relevant variables and the whole process is propagated throughout the code when the bindings of variables are computed. In the end, some computation and abstraction which can be performed at compile-time, have been completed.

- the specialised code is more efficient than the original one in the following aspects.

- The clause has been simplified.

By removing the superfluous call to `true`, the original top-level clause have been replaced by the new clauses which are very concise. The above specialised codes provide interesting insights into it.

- The message will be sent only to the relevant nodes rather than to all nodes.

Note that in reality, a WSN can have many sensors. As only the specific nodes (e.g. That is, the nodes 1 and 8 in example 1 and the node 12 in example 2) will be informed, there is no need for irrelevant nodes to wake up. More power can be saved because of less computation and transmission. In this way, we also achieved smart distribution of the code throughout the network as a result of specialisation.

- The storage on nodes has been minimised considerably.

It is obviously that only the relevant nodes need to consider the storage issue, not all nodes.

- The execution performance has been improved considerably.

Since the code has been specialised with much redundancy being removed, the node will only consider the run-time.

From the above discussion it is evident that the query rewriting technique has potential to enhance the efficiency of the WSN.



## 6 Conclusion

In this paper, we proposed a service-oriented framework to address some critical design issues and challenges of sensor network services. The sensor network services may allow end-users not only to take advantage of QoS, security and scalability which are promised by Web services, but query processing through the framework as well. We focus in particular on dealing with query processing through that framework.

We highlighted two important features of the framework. As energy efficiency is a main concern of the sensor network deployment, energy consumption at every phrase of the sensor network should be considered carefully. We have taken reducing redundancy as a starting point to demonstrate that applying query rewriting techniques at compile-time can improve the performance significantly for some problems as long as it is desirable to take advantage of the logic structure of the problem, input and static variables known a priori. Early progress from query rewriting has shown an efficient sensor network program can be obtained.

We plan to investigate query processing under the proposed framework in depth. Many issues need to be solved to develop a query optimiser with the presence of unpredictable WSN characteristics. We believe that the optimisation technique discussed in this paper will be constructive for sensor networks to deal with the tight energy and bandwidth limitation. We are also interested in sensor capability modelling.

## Acknowledgment

The authors would like to thank David Ratcliffe for his help on the improvement of the unification-based propagation prototype.

## References

1. Culler, D.E., Estrin, D., Srivastava, M.B.: Guest editors' introduction: Overview of sensor networks. *IEEE Computer* 37, 41–49 (2004)
2. Delin, K., Jackson, S.: The sensor web: a new instrument concept. In: *Proceedings of the SPIE International of Optical Engineering*, vol. 4284, pp. 1–9 (2001)
3. Liang, S.H.L., Croitoru, A., Tao, C.V.: A distributed geospatial infrastructure for sensor web. *Computers & Geosciences* 31, 221–231 (2005)
4. Gehrke, J., Madden, S.: Query processing in sensor networks. *Pervasive Computer* 3, 46–55 (2004)
5. Bonnet, P., Gehrke, J., Seshadri, P.: Towards sensor database systems. In: *Proceedings of the 2nd International Conference on Mobile Data Management* (January 2001)
6. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: An acquisitional query processing system for sensor networks. *Transactions on Database Systems (TODS)* 30, 122–173 (2005)

7. Chu, D.C., Popa, L., Tavakoli, A., Hellerstein, J.M., Levis, P., Shenker, S., Stoica, L.: The design and implementation of a declarative sensor network system. In: The 5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007), Sydney, Australia, November 2007, pp. 175–188 (2007)
8. Yao, Y., Gehrke, J.: The Cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record* 31, 9–18 (2002)
9. Park, K., Elmasri, R.: Query classification and storage evaluation in wireless sensor networks. In: *ICDE Workshops* (2006)
10. Sadagopan, N., Krishnamachari, B., Helmy, A.: Active query forwarding in sensor networks. *Ad-Hoc Networks* 3, 91–113 (2005)
11. Karp, P.B.G.B., Ke, Y., Nath, S., Seshan, S.: Irisnet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing* 2 (2003)
12. Liu, J., Zhao, F.: Towards semantic services for sensor-rich information systems. In: *Proceedings the 2nd IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (Basenets 2005)*, Boston, MA (October 2005)
13. Whitehouse, K., Liu, J., Zhao, F.: Semantic streams: a framework for composable inference over sensor data. In: Römer, K., Karl, H., Mattern, F. (eds.) *EWSN 2006*. LNCS, vol. 3868, pp. 5–20. Springer, Heidelberg (2006)
14. Coulson, G., Kuo, D., Brooke, J.: Sensor networks + grid computing = a new challenge for the grid? *IEEE Distributed Systems Online* 7 (2006)
15. Lim, H.B., Teo, Y.M., Mukherjee, P., Lam, V.T., Wong, W.F., See, S.: Sensor grid: Integration of wireless sensor networks and the grid. In: *LCN*, pp. 91–99 (2005)
16. Compton, M.: A framework for finding equivalent rewritings. Technical report, CSIRO ICT Centre, Australia (2008) (submitted to *ICDE* 2009)
17. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: *SenSys 2004: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 81–94. ACM Press, New York (2004)
18. Taylor, K., Ayyagari, A.: Research topics in semantic sensor networks: Preface to the proceedings of the semantic sensor networks workshop. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) *ISWC 2006*. LNCS, vol. 4273. Springer, Heidelberg (2006)
19. Caniot, G., Lamb, P.: Key establishment in sensor networks. Technical report, CSIRO ICT Centre Conference, Sydney (2007)
20. Lamb, P.: Arc-based XML access control for general DTDs (in preparation, 2008)
21. He, D.D., Compton, M., Taylor, K., Yang, J.: Analysing access control in collaborative environments with description logic. Technical report, CSIRO ICT Centre (2008)
22. Lloyd, J.W.: *Foundations of Logic Programming*, 2nd edn. Springer, Heidelberg (1987)
23. Leuschel, M.: Logic program specialisation. In: *Partial Evaluation*, pp. 155–188 (1998)

# Context-Driven Autonomic Adaptation of SLA

Caroline Herssens<sup>1</sup>, Stéphane Faulkner<sup>2</sup>, and Ivan J. Jureta<sup>2</sup>

<sup>1</sup> PRECISE, LSM, Université catholique de Louvain, Belgium

<sup>2</sup> PRECISE, LSM, University of Namur, Belgium

caroline.herssens@uclouvain.be, stephane.faulkner@fundp.ac.be,

ivan.jureta@fundp.ac.be

**Abstract.** Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. SLAs define the service users' Quality of Service (QoS) requirements that the service provider should satisfy. Requirements defined once may not be satisfiable when the context of the web services changes (e.g., when requirements or resource availability changes). Changes in the context can make SLAs obsolete, making SLA revision necessary. We propose a method to autonomously monitor the services' context, and adapt SLAs to avoid obsolescence thereof.

**Keywords:** SLA, adaptation, service context.

## 1 Introduction

Web services are a response to growing needs of responsive and configurable applications on the Internet. A service is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network [20]. Web services are supported by technologies such as SOAP, UDDI and WSDL [27] and are accessed via a Uniform Resource Locator.

Given the growing number of available web services on the Internet, different service providers may offer services that provide the same functionality to the users. Such competing services can be distinguished by comparison over nonfunctional characteristics, which take the form of Quality of Service (QoS). QoS is a combination of several qualities or properties of a service, e.g., availability, security, response time or throughput [15]. When a user requests a service to perform some given task, a service is selected that fits the user's QoS requirements. The selected service is the one that meets the most adequately user's preferences over quality attributes that go into QoS. Once the service is selected, it is assigned by the definition of a contract that defines a Service Level Agreement (SLA) between the user and the provider [11][17]. SLAs are used to meet user's requirements, manage user's expectations, regulate resources and control costs [22]. In short, SLAs are used to set the QoS level offered by the service provider to the service user; SLAs result from a negotiation initiated between these parties [10].

However, offered and requested QoS may both vary over time. We say in this paper that such variations occur because of changes in the *context* of services.

Given that the term “context” can be widely understood, a definition local to this paper is in order: *context* is any information about the interaction between users and a web service, for which an SLA is specified.

Changes in the context should be reflected in the SLA governing the interaction. Both the offered and the requested QoS levels may vary over the course of the interaction. Moreover, there are dependencies across different context elements that indicate a propagation of variations from one element to multiple context elements. To keep the SLA unchanged in such conditions is to make the SLA obsolescent.

*Contributions.* We propose an approach that enables an autonomic adaptation of SLA to respond to occurring context modifications. We illustrate our approach with a case study based on European Space Agency (ESA) services used to process information provided by the Envisat satellite. We provide conceptual bases necessary for SLA adaptation. We classify context elements into five distinct categories: user, provider, resource, environment and web service. We also introduce dependencies existing between elements of context enabling to propagate context modifications. We then present our SLA adaptation approach. We propose an architecture relying on an SLA manager to drive the autonomic adaptation based on context elements. Our adaptation uses context modifications and dependencies to enable an autonomic adjustment of existing SLAs to ensure the service conformance to user expectations. Adaptation involves the following steps:

1. Context modifications are reported to the SLA manager that identifies changes and starts the adaptation process.
2. Observed context variations are propagated through context dependencies existing over different elements of context by the SLA manager.
3. Once context variations have been propagated to all context categories, the SLA manager checks the compatibility between user expectations and provider capabilities.
4. Upon base of the result of the compatibility checking, the SLA manager keep the existing SLA, set up a new SLA between the user and the same provider or select another service fitting better to user expectations.

*Organization.* Section 2 introduces the ESA case study used throughout the paper. Section 3 presents the conceptual elements used to drive the SLA autonomic adaptation and illustrate these concepts with the case study. Section 4 propose our SLA management architecture and assesses the different steps of our adaptation process. The case study illustrates the adaptation process. Section 5 presents the related work; Section 6 draws conclusions and outlines future work.

## 2 Case Study

The European Space Agency (ESA) program on Earth observation allows researchers to access and use the infrastructure operated and the data collected by

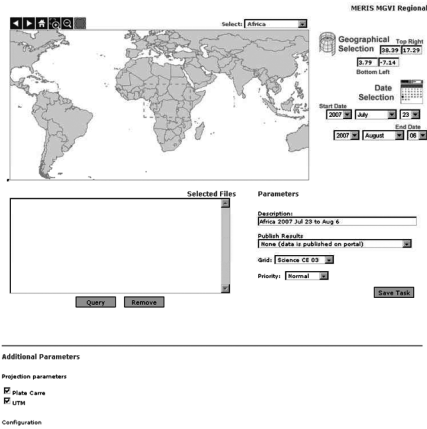


Fig. 1. Graphical interface of the MERIS/MGVI service

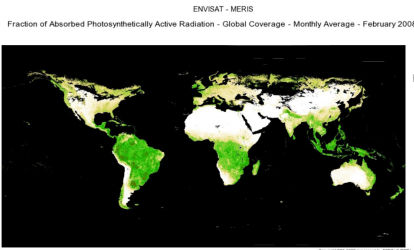


Fig. 2. Output of the MERIS/MGVI service

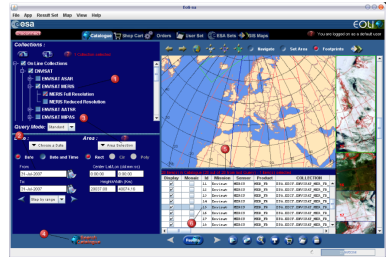


Fig. 3. Graphical interface of the EOLISA service

the agency<sup>1</sup>. Our case study focuses on the information provided by the MERIS instrument of the ENVISAT satellite. MERIS is a programmable, medium-resolution imaging spectrometer operating in the solar reflective range. MERIS is used in observing ocean color and biology, vegetation and atmosphere and in particular clouds and precipitation. In relation to MERIS, a large set of web services is made available by the ESA for access to the data the instrument sends and access to the provided computing resources.

We are interested in the remainder about two specific services. The first provides the vegetation indexes for a given period of time and region of the world. A vegetation index measures the amount of vegetation on the Earth’s surface. The graphical interface used by the requester of the service is shown in Figure 1. An illustration of the output provided for the whole world map is given in Figure 2. The data on the vegetation index can be obtained for any time range and it is possible to delimit the region of the world that is of interest. This service is

<sup>1</sup> <http://gpod.eo.esa.int>

subject to particular nonfunctional properties: the latency is initially situated between 4 and 6 hours by day of the selected period due to the quantity of data to process. Thus, the service user expects to minimize the execution time. For a service facing such significant latency, service reliability is another critical QoS aspect. Indeed, in case of failures, all execution steps must be started over. So, maximizing the reliability reduce risks to have to start over. The second web service used to illustrate our approach is the EOLI-SA service: this service is used to calculate metadata on the products to process. For example: when you submit a zone to process the MGVI with the 'bounding box' argument, these coordinates need to be transformed into the technical data of the satellite at the time of the acquisition of the zone to process (start/stop time, orbit, lat/long, azimuth angle, etc.). The graphical interface of the EOLI-SA service is given in Figure 3. This service presents different nonfunctional characteristics: while it is used by other services as the MERIS MGVI/Regional, the availability of this EOLI-SA must be maximized in order for these services to execute successfully.

### 3 Conceptual Foundations

This section introduces the concepts used to drive our autonomic SLA adaptation. We present our notion of context, and subdivide it into categories in Section 3.1. Section 3.2 introduces the concept of dependencies over context elements. All the concepts are illustrated through services introduced in Section 2.

#### 3.1 Context Categories

Unexpected events can modify the current execution context and have an impact on the performance of the services. These modifications can breach the SLA. To adapt existing SLAs to context modifications, context elements need to be accurately defined by services providers and users. We classify context elements into several categories, shown schematically along with potential between-category interactions in Figure 4.

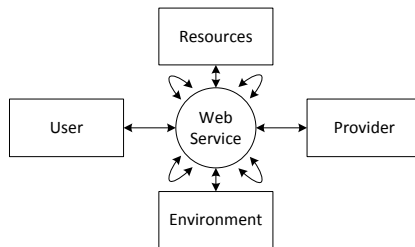


Fig. 4. Context categories and between-category interactions

**User context.** The *user context* covers the user's QoS requirements. These requirements are expressed with help of preferences over QoS values that the

service must achieve but also with QoS priorities specifying which QoS properties will be maximized over others [21]. The user context also carries information on past executions of services, along with advertised and observed QoS values during these executions [16]. Changes in user context may eventually induce the definition of new SLAs between the user and the provider. The user specifies and updates the user context.

**Resources context.** Web services executions are influenced by the availability of the resources that concern the network connection between the provider and the user but also the hardware used in executing the service and/or retrieving its results [17]. It is clear that resource availability has a direct impact on delivered QoS, thereby affecting the satisfaction of SLA. Both the service user and the service provider specify the resource context by providing their respective resource-related information. They also update this information when changes occur.

**Environment context.** The environment context contains information about where the user is located [5] and about its surrounding environment like the current weather or date [21]. This information also includes about the network, which is not within direct control of service user or provider [9,17]. The network that is not under the user's or provider's responsibility can have an immediate impact on the service performance. The environment context depends on the user of the service and is specified by the SLA manager.

**Provider context.** Provider context covers, among others, information about the provider's current execution load, the duration of its current opened sessions and announced intended length of usage by the application requesting access [9]. All activities performed by the web service and its execution charge have a direct impact on the service's QoS. Increasing or decreasing the computation charge may require changing the SLA. Provider context is specified and updated by the service provider.

**Web service context.** The Web Service context refers to nonfunctional characteristics of the service. It provides information about possible ranges of execution time, levels of security, expected best reliability, and so on [13,21]. Latency or security are determined by the service's implementation, while metrics like mean availability or reliability are obtained from its past executions. Any changes in the web service context will affect QoS levels, leading to SLA adaptations. The web service context is specified by the service provider.

Provider, web service and some part of the resource categories are related to elements of the provider side and define the level of service that can be offered. User, environment and the other part of the resource categories concern items of the user side and determine the expected level of service. The modifications of all elements of context categories are performed either by the service provider, or by the service user, except for the environment context that can be affected by external events.

**Context Illustration.** We illustrate in Table 1 the context elements for services from the case study. Both services are offered by the same provider and are

**Table 1.** Context particularities of MERIS/MGVI and EOLI-SA services

Category	MERIS/MGVI Regional	EOLI-SA
user context	maximize reliability and minimize execution time the execution time must be inferior to 7 hours by day of the selected period	maximize availability
resources context	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN
environment context	service user is an human	service user is another web service
provider context	current execution charge of the computing cluster	current execution charge of the computing cluster
web service context	execution time: 4 and 6 hours by day of the selected period reliability: upper than 95% availability: upper than 92%	execution time: inferior to 1 min  reliability: upper than 98% availability: upper than 98%

executed on the same computing cluster. Their provider and resources elements are consequently similar.

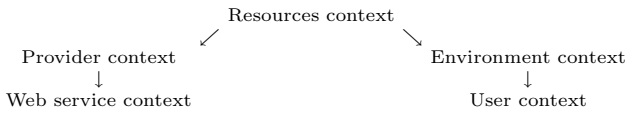
The MERIS/MGVI service has an important execution time. To prevent failures and potential restart of the execution, the user wishes both to minimize the execution time and maximize the reliability of the web service. Moreover, the user adds a hard constraint on the execution time, stating that it must be inferior to 7 hours by day of the selected period. This constraint prevents an accumulation of unfulfilled requests by the MERIS/MGVI service. The EOLI-SA service has a faster execution, its reliability is not so critical. As it is used by other services to compute data, its availability must be maximized to increase reliability of these other services.

### 3.2 Context Dependencies

Context dependencies refer to relationships that exist between distinct context elements. For example, QoS properties supported by a service provider can be interrelated [28] e.g.; change in the execution time can affect reliability. These relationships can also occur over elements in different context categories – e.g., the execution charge of computing resources (provider context) affects various quality characteristics of the service (web service context). To highlight such dependencies between elements allows us to propagate failures and performance modifications to all context categories related to the initial variation. Dependencies are defined over context elements with an associated *coefficient*, *direction* and *strength*. The *coefficient* attribute specifies that context elements involved in the dependency are parallel or opposite, meaning that their coefficient is positively or negatively correlated. The *direction* determines, which of the two considered context elements induces the value of the other; a dependency can be directed both ways, meaning that both context elements impact each other. The *strength*, represented by a value between 1 and 10, corresponds to the importance of the influence.



While all context dependencies occurring on the same context category are allowed, between-category dependencies are subject to some restrictions. We specify in Section 3.1 that part of the resources context, the provider context and the web service context are defined by the service provider. The other part of the resources context, the environment context and the user context are delimited by the service user. Dependencies can not involve influence of provider context categories to user context categories and vice versa. Dependencies are also constrained by the attribute direction over different categories. An improvement of the computing resources (resources category) can induce the service quality performance (web service context). Nevertheless, the service quality performance (web service context) has no influence on the computing resources (resources category). The impact direction of dependencies in context categories is given below:



**Examples of context dependencies.** Context dependencies are specified by the SLA manager to indicate existing interactions between context elements. Table 2 gives examples of context dependencies for MERIS/MGVI and EOLISA services.

**Table 2.** Examples of Context Dependencies

Common dependencies of MERIS/MGVI and EOLISA services		
Dep 1	Coefficient: parallel Direction: → Strength: 10	Resources context - Web Service context
Dependencies of the MERIS/MGVI service		
Dep 2	Coefficient: parallel Direction: → Strength: 6	Execution charge of the computing cluster (provider context) - Execution time (web service context)
Dep 3	Coefficient: opposite Direction: → Strength: 8	Execution charge of the computing cluster (provider context) - Availability (web service context)
Dep 4	Coefficient: parallel Direction: → Strength: 5	User bandwidth (environment context) - Transfer Time (user context)
Dependencies of the EOLISA service		
Dep 5	Coefficient: parallel Direction: ↔ Strength: 9	Availability (web service context) - Reliability (web service context)
Dep 6	Coefficient: opposite Direction: → Strength: 2	Execution charge of the computing cluster (provider context) - Availability (web service context)

*Dep 1* states the relationship between the resources used by the service provider and web service performance. Indeed, the capability of the web service is immediately linked to the resource used to compute the web service. If the server used to compute MERIS/MGVI is down, all its performance indicators will be affected. The EOLI-SA service is also subject to that dependency. *Dep 2* underlines the impact of the execution charge of the cluster on the execution time needed to execute the service. Increasing the execution charge of the provider decreases resources allocated to the execution of the service and increases its execution time. *Dep 3* states that the increasing of the execution charge of the cluster will decrease the availability of the service: if the cluster charge is full, the MERIS/MGVI service that requires an important resources utilization will not be given a high priority. Consequently, its availability will be reduced. *Dep 4* is about the influence of the user's bandwidth on the user's network capacities. If the bandwidth provided by its Internet Service Provider decreases, the service user will amend its expected total transfer time. The EOLI-SA service is less subject to context variations because it does not consume that much resources. It is subject to the dependency linking its availability to its reliability: *Dep 5*. This dependency is directed both ways meaning that the increasing/decreasing of one of the quality property affects the other. It is also subject to the *Dep 6* stating that the increasing of the execution charge of the cluster causes a diminution of the availability. This dependency is the same that the one observed for the MERIS/MGVI service but its strength is not so prominent because EOLI-SA does not require a long execution time and is less subject to the provider's utilization.

## 4 Dynamic SLA Adaptation

We outline here our adaptation process that fits SLA established between service user and provider to context elements variations. Section 4.1 gives a SLA description and presents our SLA management architecture. Section 4.2 details the different steps of our adaptation process.

### 4.1 Managing Service Level Agreements

Contracts between a service provider and a service user are given by SLAs [11]. An SLA covers the functional side of the service (the provided service corresponds to the requested service in terms of input, output, pre- and postconditions) and concerns also the nonfunctional properties of the service. When users can choose among a set of functionally equivalent web services, QoS considerations become the key criteria for service selection. As a consequence, SLAs must be defined and managed between service users and providers [3]. The contract about non-functional properties is defined for each QoS property through a Service Level Objective (SLO) [22]. SLOs are defined over QoS values and appropriate metrics. Definitions of metrics include the description of their calculation mode and are provided by the party in charge of measurement and aggregation,

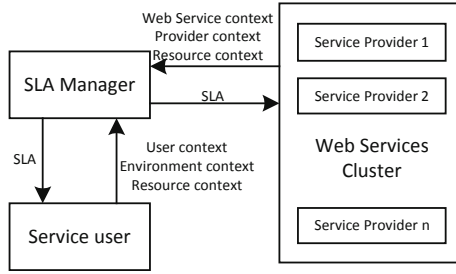
i.e.; either the service provider, the service user or a third tier manager [3,29]. An SLA is then a contract between the service user and service provider about a set of SLOs. These SLOs refer to web service and user context elements. Web service context elements are QoS capabilities of the provider while user context elements are quality requirements of the service user. A complete definition of a SLA and its component is available in [23]. We work on the assumption that an initial SLA has already been negotiated between the service user and the service provider with a negotiation process such as the one specified in [29]. We propose in Table 3 examples of SLAs established between providers and users for both services presented in the case study section.

**Table 3.** Examples of SLAs

SLA established for the MERIS/MGVI service	
SLO 1:	the execution time must be between 5h and 6h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
SLA established for the EOLI-SA service	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 92%

To manage SLAs and their adaptation, we introduce a third-part service: the *SLA manager*, in charge of the mediation between the service user and the service provider. The adaptation process refers to automatic monitoring, enforcement and optimization of SLAs between the services's user(s) and provider. The SLA manager is also responsible of the assignation of services to users. Our management architecture is illustrated in Figure 5. We dedicate one SLA manager for each existing cluster of web services (i.e., services that offer the same functionality). Gathering of functionally equivalent web services is ensured by means of clusters of web services, that provide several web services inside a unique wrapper, used by the clients as a standard web service [6]. We suppose than an initial SLA has already been negotiated between the service user and the service provider chosen with an adequate selection method [14].

The role of the SLA manager is to continuously check the conformance of the web service to the SLA established between the user and the provider. This monitoring requires a constant verification of SLOs compliance between the service user and the service provider. To achieve this verification, context information about the web service, the provider and the part of the resources information are given by the provider while information related to the user context, its environment and its resources are communicated by the service user. The SLA manager records information about all context elements and builds execution statistics about mean observed latency, reliability or availability. The SLA manager also monitors context dependencies with help of information provided by the user and the provider. With such statistics and information about the execution context and dependencies, the SLA manager is able to check the conformity of SLOs



**Fig. 5.** SLA Management Architecture

established between the service user and the service provider. If some SLOs are breached, the SLA manager processes adaptation mechanisms to adjust the SLA, as explained in Section 4.2.

## 4.2 Adapting Service Level Agreements

The SLA manager is designed to respond to eventual SLA breaches or QoS variations through different mechanisms of adaptation. *Adaptation* usually refers to the alteration of an application's behavior or interface in response to arbitrary context changes [2]. For web services, the adaptation must consider all context particularities introduced in Section 3.1 as well as existing dependencies over context elements presented in Section 3.2. The aim of such adaptation mechanisms, referred as *SLA adaptation*, is to adjust the initial SLA to context variations reported to the SLA manager. If the initial web service provider is no longer able to perform its task to the quality level requested by the user, the SLA manager proceeds to *select a new provider*. It establishes a new contract between the service user and a service provider selected in the cluster of available services.

The SLA manager process this adaptation through four steps:

**1. Modification notification.** The SLA adaptation process is driven by the observation of a modification in at least one context category. Such changes are highlighted by information provided by users and providers and statistics made by the SLA manager. The adaptation is initiated differently following the category of the context variation. Provider, web service and some part of the resources context come from the service provider and their changes will modify the service offered, while the user, the environment and the other part of the resources context are defined by the service user and will affect the service level expected.

**2. Modification spreading.** The second step of the SLA adaptation is the propagation of observed context variation to elements of the same category and to other relevant context categories. Spreading the modification is subject to rules presented in Section 3.2, which define the direction of the allowed dependencies. The impact of the context variations is governed by the coefficient,

direction and strength attributes and reflects changes to all elements of the concerned context categories. Context dependencies allow the SLA manager to propagate the impact of context categories until their influence to related QoS: all context variations are converted to elements used in the SLA contract (i.e., user and web service context).

**3. Compatibility checking.** Once all dependencies have been propagated, the SLA manager is able to determine the final quality expectations of the user and the web service QoS offered by the service provider. To ensure their compatibility, the SLA manager checks context elements accounted for in the SLA – i.e., the user and web service context. If the web service context presents abilities that meet the expectations of the user context, these are compatible.

**4. Adaptation.** The last step of the process is the adaptation resulting from compatibility checking. Three different scenarios are possible. (1) The compatibility is present between web service and user context and the initial SLA is still applicable. In this instance, the SLA is preserved between stakeholders. (2) The compatibility is verified between web service and user context but the initial SLA no longer applies. The SLA initiates the set up of a new SLA between the current provider and the service user. To achieve the negotiation between the user and the provider, the manager uses a negotiation process such as the one proposed in [29]. (3) The last possibility occurs while the compatibility between the user and the provider is not verified. The SLA manager then select another service able to meet the quality requirements of the user context in the web services cluster. We do not review here details of selection mechanisms but various existing approaches [7,14,31] can be applied by the SLA manager. The SLA is negotiated between the new provider and the service user by the SLA manager.

**Adaptation Illustration.** We illustrate here adaptation steps through a particular situation involving services introduced in the case study.

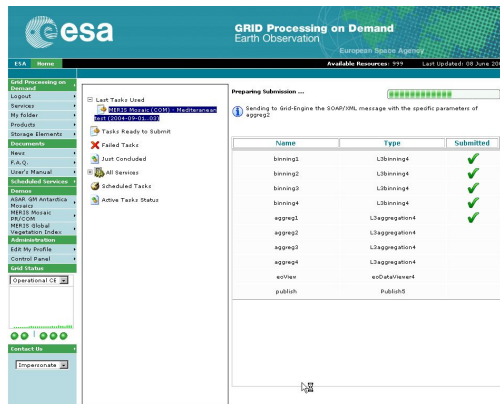


Fig. 6. Current tasks of the provider

The adaptation process described here occurred with an increase of the execution charge in the computing cluster. The execution charge of the computing cluster belongs to the provider context category. The services accessing the computing cluster offered by the ESA are monitored and managed through a particular access interface illustrated in Figure 6. The execution charge can significantly increase with entrance of new requests in the computing cluster. The adaptation mechanisms initiated in response to these new requests will differ with the extent of the increasing. The adaptation process initiated by this increasing is described through its four steps here.

The *first step* is the modification notification. The provider charge is monitored through the application illustrated in Figure 6. With this application, the provider is able to notice the growth of the cluster utilization. The cluster is allowed to work without delays within the execution duration advertised at a fixed level of charge. When the charge moves beyond this level, the provider notifies the SLA manager. We observe the effect produced by two different increases: the first case is an increase of the charge of the computing cluster for 20%; the second involves an increase of 50%.

The *second step* of the adaptation process amounts to spread context modifications. The dependencies are directly related to an increase of the execution charge, i.e., *Dep 2*, *Dep 3*, *Dep 6*. *Dep 2* induces an increase of the execution time and *Dep 3* leads to a decrease of the availability of the MERIS/MGVI service. The *Dep 6* leads to a decrease of the EOLI-SA availability. This decrease enables *Dep 5*, which refers to a decrease of reliability. The effects of an increase of the execution charge on service context of both services are: an increase of the execution time and a decrease of the availability for the MERIS/MGVI service; and a decrease of the availability and the reliability for the EOLI-SA service. The web service contexts of both services resulting from increases of 20% and 50% are illustrated in Table 4.

**Table 4.** Web Service context of MERIS/MGVI and EOLI-SA services after an increasing of the execution charge

Increasing of the execution charge of 20%	
MERIS/MGVI Regional	EOLI-SA
execution time: 5h and 7h hours by day of the selected period	execution time: inferior to 1 min 10 sec
reliability: upper than 95%	reliability: upper than 97%
availability: upper than 85%	availability: upper than 97%
Increasing of the execution charge of 50%	
MERIS/MGVI Regional	EOLI-SA
execution time: 8 and 10 hours by day of the selected period	execution time: inferior to 1 min 30 sec
reliability: upper than 95%	reliability: upper than 80%
availability: upper than 78%	availability: upper than 80%

The *third step* of the SLA manager's process is the compatibility checking between user requirements and the provider's capabilities. With an increase of 20%, the MERIS/MGVI capabilities still meet user requirements. The EOLI-SA

**Table 5.** SLAs resulting from the increasing of the execution charge

New SLA established for the MERIS/MGVI service with an increasing of the execution charge of 20%	
SLO 1:	the execution time must be between 6h and 7h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
New SLA established for the EOLI-SA service with an increasing of the execution charge of 50%	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 80%

is also facing the user expectations with this increase of the execution charge. With an increase of 50%, MERIS/MGVI does not meet the constraint on the maximum allowed execution time, so that the compatibility does not verify. In contrast, the EOLI-SA service is still facing the user requirements and does not break any constraint of the user.

The *fourth step* of the SLA manager is the adaptation. With an increase of 20%, the MERIS/MGVI service is in scenario 2; it is compatible with user requirements but breaches the initial SLA: its execution time is above 6 hours by day of the selected period. The SLA between the user and the provider must be renegotiated. This new SLA is illustrated in Table 5. The EOLI-SA service respects the scenario 1; it is still compatible with user requirements and the initial SLA is still applicable. The initial SLA is preserved between the user and the provider. With an increasing of the execution charge of 50%, the MERIS/MGVI service follows the scenario 3. The service fails to meet the constraint stating that the execution time must be inferior to 7 hours by day of the selected period. The SLA manager selects another service in the services cluster that is able to meet user requirements. The EOLI-SA is in the scenario 2; it is compatible with user expectations but the *SLO 3* of its SLA is breached, the availability is inferior to 92%. A new SLA is negotiated between the service user and the provider, it is illustrated in Table 5.

## 5 Related Work

Adaptation to failures and SLA violations has received attention [18,19,24]. However, the influence of context on SLA adaptation has not been studied in depth. Analyzing the impact of context variations on software behavior is a problem outlined in various other areas such as computer human interaction [5], pervasive computing [18,30] and autonomic systems [2,24]. Context-sensitivity is usually defined as an application software system's ability to sense and analyze context from various sources. It lets application software take different actions adaptively in different contexts [18]. In response to these changes, several adaptation strategies exist [4,12]. Among them, In et al. [8] outline the problem caused by QoS of situation-aware applications. The relationships

between changes of situations and resources required to support the desired level of QoS is not clear. They solve this problem with a situation-aware middleware able to predict all QoS requirements of the applications and to analyze tradeoff relationships among the different QoS requirements. The resource availability may be changed according to dynamically varying situations. Such changes in QoS requirements and QoS constraint violations are identified by their middleware that resolves conflicts by rescheduling resources for supporting high priority missions. In contrast to this model, our proposal relies on an existing definition of dependencies between context elements. Moreover, in the web service area, all resources cannot be modified or rescheduled or are even out of the scope of the service provider or the service user. Our model adapts SLA to context changes and does not intervene on the context elements to comply to QoS requirements. Tosic [26] proposes an alternative to custom-made SLA, the utilization of Web Service Offerings which is supported by an infrastructure (WSOI) and a specific language (WSOL) [25]. Each service is proposed with some classes of service that differ in usage privileges, service priorities, response time guaranteed and verbosity of response information. Their approach cuts off the negotiation problem between the service provider and the service user. However, such predefined classes of service only allow a discrete variation of QoS offered to the service user. Classes of services are predefined, limiting their number and therefore the adaptation possibilities.

## 6 Conclusions and Future Work

The management of SLAs between an user and a provider in the context of web services is essential to enable autonomy of web service executions. It allows an automatic resolution of conflicts occurring after web service failures or updated expectations of the user. The first advantage of our method is the reliance on the identification of context elements and existing dependencies between these context elements. The context and dependencies allow the SLA manager to anticipate problems. The modifications of context elements are reported to the SLA manager by the provider and the user before the service is executed by the service user. Thus, the SLA manager is able to anticipate and adapt consequently the existing SLA or establish a contract with a new provider. The second advantage of our method is that the SLA manager tries to preserve the existing contract between the service provider and the service user. Long term collaborations between stakeholders are protected from the continuous switching over existing services and new services are selected only when the current provider is not able to meet the user expectations.

Future work consists of the definition of an appropriate language that enables us to integrate context and dependencies elements in existing web services architectures and technologies.



## References

1. Bianculli, D., Jurca, R., Binder, W., Ghezzi, C., Faltings, B.: Automated Dynamic Maintenance of Composite Services Based on Service Reputation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 449–455. Springer, Heidelberg (2007)
2. Bradbury, J.S., Cordy, J.R., Dingel, J., Wermelinger, M.: A Survey of Self Management in Dynamic Software Architecture Specification. In: Proc. ACM SIGSOFT Worksh. Self-healing systems, pp. 28–33 (2004)
3. Cappiello, C., Comuzzi, M., Plebani, P.: On Automated Generation of Web Service Level Agreements. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 264–278. Springer, Heidelberg (2007)
4. Cibrán, M.A., Verheecke, B., Vanderperren, W., Suvé, D., Jonckers, V.: Aspect-oriented Programming for Dynamic Web Service Selection, Integration and Management. *World Wide Web Journal* 10(3), 211–242 (2007)
5. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* 16(2-4), 97–166 (2001)
6. Fernandez Vilas, J., Pazos Arias, J., Fernandez Vilas, A.: High Availability with Clusters of Web Services. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 644–653. Springer, Heidelberg (2004)
7. Herssens, C., Jureta, I.J., Faulkner, S.: Dealing with Quality Tradeoffs during Service Selection. In: ICAC 2008: IEEE Int. Conf. Autonomic Comput. (2008)
8. In, H.P., Kim, C., Yau, S.S.: Q-MAR: An Adaptive QoS Management Model for Situation-Aware Middleware. In: Yang, L.T., Guo, M., Gao, G.R., Jha, N.K. (eds.) EUC 2004. LNCS, vol. 3207, pp. 972–981. Springer, Heidelberg (2004)
9. Julien, C.: Adaptive Preference Specifications for Application Sessions. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 78–89. Springer, Heidelberg (2006)
10. Kaminski, H., Perry, M.: SLA Automated Negotiation Manager for Computing Services. In: CEC/EEE 2006: IEEE Int. Conf. E-Commerce Tech. (2006)
11. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. IBM Corporation (2003)
12. Lundesgaard, S.A., Lund, K., Eliassen, F.: Utilising Alternative Application Configurations in Context- and QoS- Aware Mobile Middleware. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025, pp. 228–241. Springer, Heidelberg (2006)
13. Maamar, Z., Mostefaoui, S.K., Yahyaoui, H.: Toward an agent-based and context-oriented approach for Web services composition. *IEEE Trans. Knowl. and Data Eng.* 17(5), 686–697 (2005)
14. Maximilien, M.E., Singh, M.P.: Toward Autonomic Web Services Trust and Selection. In: ICSOC 2004: Int. Conf. Service Oriented Comput. (2004)
15. Menascé, D.A.: QoS Issues in Web Services. *IEEE Internet Computing* 6(6), 72–75 (2002)
16. Muldoon, C., OHare, G., Phelan, D., Strahan, R., Collier, R.: Access: An agent architecture for ubiquitous service delivery. In: Klusch, M., Omicini, A., Ossowski, S., Laamanen, H. (eds.) CIA 2003. LNCS, vol. 2782, pp. 1–15. Springer, Heidelberg (2003)
17. Myerson, J.: Use SLAs in a Web Services Context, Part 1: Guarantee your Web Service with a SLA. IBM Research Report (2004), <http://www.ibm.com/developerworks/library/ws-sla/>

18. Nahrstedt, K., Dongyan, X., Wichadakul, D., Baochun, L.: QoS-aware middleware for ubiquitous and heterogeneous environments. *Comm. Mag., IEEE* 19(11), 140–148 (2001)
19. Netto, M., Bubendorfer, K., Buyya, R.: SLA-Based Advance Reservations with Flexible and Adaptive Time QoS Parameters. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007*. LNCS, vol. 4749, pp. 119–131. Springer, Heidelberg (2007)
20. Papazoglou, M.P., Georgakopoulos, D.: Introduction. *Comm. ACM* 46(10), 24–28 (2003)
21. Qiu, L., Chang, L., Lin, F., Shi, Z.: Context optimization of AI planning for semantic Web services composition. *Service Oriented Comput. and Applications* 1(2), 117–128 (2007)
22. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A.P.A., Casati, F.: Automated SLA Monitoring for Web Services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) *DSOM 2002*. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
23. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Research Report HPL-2001-310 (R.1), Hewlett-Packard Laboratories Palo Alto, July 2002 (2002),  
<http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf>
24. Skorin-Kapov, L., Matijasevic, M.: Dynamic QoS Negotiation and Adaptation for Networked Virtual Reality Services. In: *WOWMOM 2005: IEEE Int. Symp. World of Wireless Mobile and Multimedia Networks*, pp. 344–351 (2005)
25. Tomic, V., Pagurek, B., Patel, K.: WSOL - A Language for the Formal Specification of Classes of Service for Web Services. In: *ICWS 2003, IEEE Int. Conf. Web Serv.* (2003)
26. Tomic, V.: Service offerings for xml web services and their management applications. PhD thesis (2004)
27. Walsh, A.E. (ed.): *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference, Englewood Cliffs (2002)
28. Wang, C., Wang, G., Wang, H., Santiago, R.: Quality of Service (QoS) Contract Specification, Establishment, and Monitoring for Service Level Management. *J. Object Tech.* (2007)
29. Yan, J., Kowalczyk, R., Lin, J., Chhetri, M.B., Goh, S.K., Zhang, J.: Autonomous Service Level Agreement Negotiation for Service Composition Provision. *Future Generation Computer Systems* 23, 748–759 (2007)
30. Yau, S.S., Karim, F., Wang, Y., Wang, B., Gupta, S.K.S.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *Pervasive Comput.* 1(3), 23–30 (2002)
31. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.* 30(5), 311–327 (2004)

# Determining QoS of WS-BPEL Compositions

Debdoot Mukherjee<sup>1</sup>, Pankaj Jalote<sup>2</sup>, and Mangala Gowri Nanda<sup>1</sup>

<sup>1</sup> IBM India Research Lab, New Delhi  
{debdmukh,mgowri}@in.ibm.com

<sup>2</sup> Indian Institute of Technology, Delhi  
jalote@cse.iitd.ac.in

**Abstract.** With a large number of web services offering the same functionality, the Quality of Service (QoS) rendered by a web service becomes a key differentiator. WS-BPEL has emerged as the de facto industry standard for composing web services. Thus, determining the QoS of a composite web service expressed in BPEL can be extremely beneficial. While there has been much work on QoS computation of structured workflows, there exists no tool to ascertain QoS for BPEL processes, which are semantically richer than conventional workflows. We propose a model for estimating three key QoS parameters - Response Time, Cost and Reliability - of an executable BPEL process from the QoS information of its partner services and certain control flow parameters. We have built a tool to compute QoS of a WS-BPEL process that accounts for most workflow patterns that may be expressed by standard WS-BPEL. Another feature of our QoS approach and the tool is that it allows a designer to explore the impact on QoS of using different software fault tolerance techniques like Recovery blocks, N-version programming etc., thereby provisioning QoS computation of mission critical applications that may employ these techniques to achieve high reliability and/or performance.

**Keywords:** Quality of Service, composite web services, workflows, BPEL.

## 1 Introduction

Services Oriented Architecture aims to provide infrastructure for a marketplace wherein more services will be produced by composing various web services rather than coding programs from scratch. As in any competitive market, where a number of offerings are available for the same functionality, Quality of Service is slated to be the key differentiator. Knowing the QoS of the web service being composed is extremely crucial during the process of service orchestration (binding concrete web services to tasks in the workflow). The integrator of the WS-composition has to judiciously choose every web service that he binds to the composition in order to attain a high level of QoS and meet his Service Level Agreement(SLA) requirements. A tool that can estimate the QoS of the resultant WS-composition, given the values of QoS parameters for constituent web services will come in most handy for the integrator. This paper aims to provide such a framework for QoS determination in WS-BPEL [1] processes.

Business Process Execution Language (BPEL) has emerged as the de-facto standard for representation of industrial workflows. BPEL has been proven to be more expressive than traditional workflow modeling languages [2] - most of which support only block structured flow constructs. Some of the key features of BPEL that distinguish it from other workflow languages are: (i) *Synchronization links* and the *transition* and *join conditions* that one can impose on these links, (ii) *Fault handlers* and *compensation handlers* to support fault handling and backward recovery of long running transactions respectively, and (iii) Event driven programming constructs like *receive*, *pick* and *event handlers*.

Despite BPEL emerging as the standard for web service composition, most of the work on QoS of composite web services has so far focused on composition through structured workflows [3,4]. As mentioned, BPEL is semantically more powerful than workflow languages, hence QoS computation using conventional workflows can handle only a small subset of BPEL programs. In this paper, we determine *Reliability*, *Response Time* and *Cost* of any composition expressed in standard WS-BPEL. The model also allows adding fault tolerant constructs like *N-version programming* [5], *Recovery block* [6], *Return the Fastest Response* and *Deadline Mechanism* to enhance the reliability and performance of the tasks in the BPEL composition and then determining their impact on QoS. Although the use of fault tolerant constructs in web services have been studied in literature [7,8,9], there has been no research that quantifies the QoS improvement that may be brought about by these constructs.

**Outline of our Approach.** For determining QoS, the BPEL file is parsed to build an activity graph that consists of activities as nodes and captures the dependencies between activities that arise from the control flow structure of the BPEL process. All QoS computations happen at the level of a node in this graph. The dependencies of an activity are tracked in order to determine the probability with which the activity may start execution and also the time instant in a run of the BPEL process when it is fired. These estimates about an activity alongwith QoS estimates of its children help us to model its QoS. A recursive algorithm runs through the activity graph computing QoS parameters at each node and recombining these values to arrive at reliability, response time and cost for the overall WS-BPEL process. The entire approach has been implemented in a tool that takes as input a BPEL process and QoS of constituent web services and computes QoS for the composition. In this paper, we illustrate the approach by applying it to an E-Governance application that models the passport office workflow. Our tool also offers the flexibility to replace an invocation of a web service by a more robust invocation of a set of web services tied via fault tolerant constructs. QoS improvement achieved through such a design can then be precisely measured through rules defined specific to fault tolerant constructs that aggregate QoS of constituent web services. The major contributions of the paper include providing a QoS determination framework for WS-BPEL processes and provisioning an environment where integrators of composite web services can try out fault tolerant designs and gauge QoS of such processes at design time.

## 2 WS-BPEL (Business Process Execution Language)

WS-BPEL lays down a grammar for capturing the behavior of a business process based on interactions between the process and its partner processes. The root element of any WS-BPEL file is  $\langle process \rangle$  - an element that outlines the scope of the business process and encloses declarations for all *partner links*, *variables*, *handlers* and an *activity* (this activity may in turn contain other activities). An activity can be of two types: *basic* or *structured*.

Basic activities either describe interactions with other partners or model primitive steps in the process. Basic activities include -  $\langle invoke \rangle$  to call operations;  $\langle receive \rangle$  and  $\langle reply \rangle$  to accept and respond to inbound messages respectively;  $\langle assign \rangle$  used to carry out updates on variables;  $\langle wait \rangle$  to introduce delays;  $\langle exit \rangle$  to immediately end the business process;  $\langle throw \rangle$  and  $\langle rethrow \rangle$  to signal internal faults; and  $\langle empty \rangle$  to do nothing.

*Structured activities* encode control-flow logic and can have other activities nested in them. WS-BPEL enumerates seven different structured activities. A  $\langle sequence \rangle$  contains one or more activities that are performed in the order in which they appear within the  $\langle sequence \rangle$  element. A  $\langle flow \rangle$  provisions execution of activities concurrently and also allows for synchronization between the activities contained in it through the notion of *links*. An  $\langle if \rangle$  consists of one or more conditional branches defined by the  $\langle if \rangle$  and optional  $\langle elseif \rangle$  elements, followed by an optional  $\langle else \rangle$  element. A  $\langle pick \rangle$  waits for the occurrence of exactly one event from a set of events and executes the activity contained within that event. The events can either be receipt of inbound messages ( $\langle onMessage \rangle$ ) or triggering of timer based alarms ( $\langle onAlarm \rangle$ ). WS-BPEL 2.0 supports three forms of loop constructs -  $\langle while \rangle$ ,  $\langle repeatUntil \rangle$  (both checked by truth value of the  $\langle condition \rangle$  set in them) and  $\langle forEach \rangle$  (controlled by an implicit index variable that is initialized to  $\langle startCounterValue \rangle$  and ends in  $\langle finalCounterValue \rangle$ ).

WS-BPEL's notion of a  $\langle scope \rangle$  offers the ability to specify a behavioral context within which an activity may execute. A scope allows definition of variables, partner links, message exchanges and correlation sets that are visible only within the scope. Event handlers, fault handlers, a compensation handler, and a termination handler may also be attached to a scope.

## 3 A Running Example - Passport Application Service

We present an example of a passport office workflow implemented with the help of WS-BPEL to elucidate the key concepts throughout this paper. The composite web service for the passport workflow calls upon operations in external web services to verify parameters such as age and date of birth whose validation is required for issuing a passport. Age may be verified by either an education board or a municipal office. After verification of age and address, one may proceed to issuing a passport only if the bank payment has been made by the applicant.

A part of the BPEL process is shown graphically in Figure [1](#). The synchronization links named X, Y and Z and the  $\langle transitionCondition \rangle$ s specified on

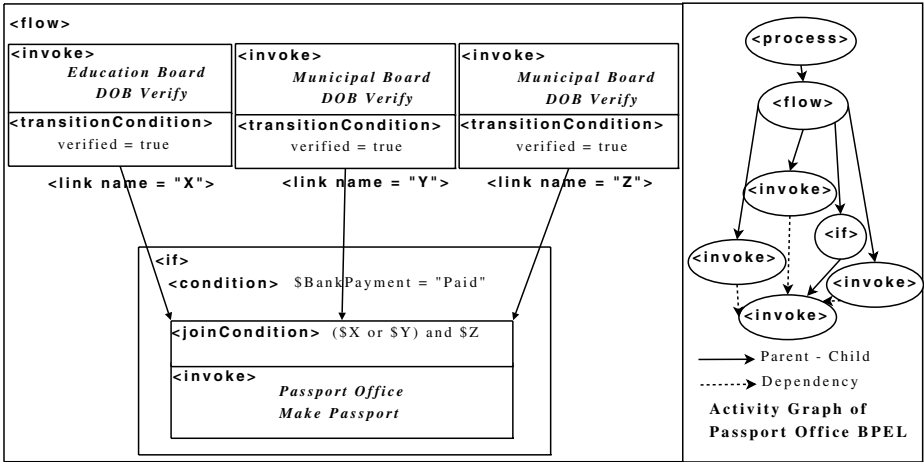


Fig. 1. Passport Office BPEL Process

them are used to ensure that the web service to issue passport may be started only if the furnished documents have been successfully validated. The boolean expression on the links in the *joinCondition* at the *invoke* for passport issue web service examines whether at least one age-proof is valid and the given address proof has been verified. Our QoS model will require as input reliability, time and cost for each of the four constituent web services and the probabilities of success of each of the transition conditions and the *if* condition.

### 4 Determining QoS

Our QoS model estimates reliability, response time and cost for a WS-BPEL process and is capable of dealing with the complex graph-like control flow structures, event driven programming constructs and fault handling mechanisms that may be present in it. Since activities in WS-BPEL may be heavily intertwined with synchronization links, one cannot perform reductions analogous to those proposed in [3] and derive QoS of a block by simply aggregating the QoS of its constituents. In order to infer QoS of an activity we track all the dependencies that the activity may have and obtain their QoS. An activity *A* is said to be *control dependent* on another activity *B*, if *A* may only start execution after the completion of *B*. In WS-BPEL, a control dependency of an activity *A* may be another activity *B* if *B* is either the source of an incoming link to *A* or *B* precedes *A* in a *sequence*.

The model captures the three key QoS parameters addressed by it in the following way:

- *Reliability* of a BPEL process is calculated as the probability that the process scope will successfully complete execution.

- *Response Time* is a random variable characterized by values for mean and standard deviation representing the expected time taken for completion of the process.
- *Cost* of a BPEL composition is calculated as the aggregate of expected costs of all activities contained in it, assuming that each web service invocation incurs some fixed cost.

**Inputs to the Model.** Apart from the values of reliability, response time and cost for all web service invocations of the BPEL process, the model also assumes certain parameters that characterize the control flow of the business process to be available as inputs. These include the probability of selecting branches or events in *if* and *pick* activities respectively, the average number of iterations in loops and for each *catch* or *catchAll* block the fraction of failures of its associated scope that it successfully intercepts. Average waiting times for all *<receive>* activities (that are not start activities) and all *<onMessage>* events that are used to intercept inbound messages to the business process are required by the model. All of these attributes can be determined from the execution log of the business process.

#### 4.1 Overall Approach

All computations in our model occur at the level of an *activity* or a *scope* or a *handler*, which are various units of encapsulation of process logic in WS-BPEL. The BPEL workflow is represented as an *activity graph* where the activities/scopes/handlers are represented by nodes. A node,  $X$ , in the BPEL process is annotated by: (a) its child nodes, i.e., activities that are directly contained by  $X$  (b) its control dependencies. Additionally, nodes for *invoke* activities and *scopes* are annotated with *catch* blocks, *fault handlers*, *event handlers* and *compensation handlers* that they may be associated with. The node corresponding to the *<process>* scope forms the root node of this activity graph. Figure 1 shows the activity graph obtained out of the passport office BPEL process.

**Basic Elements of the Model.** QoS computation of a BPEL process requires that we compute for each node  $X$  in this graph - (a)  $P(\mathcal{S}_X)$ : Probability that  $X$  *successfully* completes execution in a single run of the process, (b)  $ET_X$ : Expected end time or the time of completion of  $X$  measured relative to the start of the process, (c)  $Cost_X$ : Sum of the expected costs of all its child nodes. These three parameters for the root node of the activity graph give reliability, response time and cost respectively for the WS-BPEL composition.

*Successful completion* of a node encompasses the corresponding activity / scope / handler delivering its desired functionality to the effect that it measures upto the expectations of all other nodes that might be dependent on it. In determining  $P(\mathcal{S}_X)$  of a node  $X$ , we make use of  $P(start_X)$  which is the probability that the activity may start execution in a state that is semantically in accordance with one which is expected at that point.  $P(start_X)$  abstracts all effects that the dependencies of a node  $X$  may have on  $P(\mathcal{S}_X)$ . The computation of  $P(\mathcal{S}_X)$  may involve use of  $P(start_X)$  and a suitable aggregation (separately defined for

**Table 1.** Algorithm QoS Computation

---

**Algorithm** *setQoS(X)* : Set  $P(\mathcal{S})$ ,  $ET$  and  $Cost$  for a node  $X$

---

**for all**  $Z$  such that  $Z$  is a dependency of  $X$  **do**  
   *setQoS(Z)*  
**end for**  
Compute  $P(start_X)$ ,  $ST_X$  and  $PC_X$   
**for all**  $Z$  such that  $Z$  is a child of  $X$  **do**  
   *setQoS(Z)*  
**end for**  
Compute  $P(\mathcal{S}_X)$ ,  $ET_X$  and  $Cost_X$  according to rules specific to Type of  $X$ .

---

each activity type) of  $P(\mathcal{S})$  of all activities / scopes / handlers contained in  $X$ . Response time computation involves obtaining for each activity their start and end times of execution in some run of the business process.  $ST_X$  denotes the time instant when all dependencies of  $X$  are complete and  $X$  is ready to start. The end times of all nodes nested within  $X$  are estimated and suitably aggregated with  $ST_X$  to obtain  $ET_X$ .  $Cost$  is computed more on the lines of the reduction based approach.  $PC_X$  is the conditional probability that  $X$  will start execution in an instance of the BPEL process given that the parent of  $X$  starts execution in the same instance.  $Cost_X$  for a node  $X$  is given by the sum of costs of all its child nodes relaxed by their  $PC$ s.

**Algorithm Description.** After the BPEL XML document has been parsed to prepare the *activity graph*, we can proceed with QoS determination. Algorithm 1 presents the recursive structure of QoS computation followed for any node in our activity graph. If we invoke the function *setQoS()* on the root node, all nodes of the activity graph are traversed.  $P(\mathcal{S})$ ,  $ET$  and  $Cost$  are set in each of them before we obtain these parameters for the root node and hence QoS of the BPEL process.  $P(\mathcal{S})$ ,  $ET$  of all dependencies of  $X$  are required to compute parameters  $P(start_X)$ ,  $ST_X$  that help to abstract the effects of the dependencies of  $X$  in the QoS computation of the children of  $X$ . The termination of Algorithm 1 is guaranteed because well-formed WS-BPEL processes cannot have control cycles formed with the help of links (See SA00072 in [1]).

## 4.2 Determining $P(start)$ , Start Time ( $ST$ ) and $PC$

The methodologies to calculate  $P(start_X)$ ,  $ST_X$  and  $PC_X$  for any node  $X$  remain the same irrespective of the type of  $X$  whereas determination of  $P(\mathcal{S}_X)$ ,  $ET_X$  and  $Cost_X$  happen according to rules defined specific to each activity type (See Section 4.3).

The computation of  $P(start_X)$  requires the probability,  $P(joinCondition_X = true)$ , that the join condition (explicit or implicit) attached to the node  $X$  evaluates to true. A *join condition* is a boolean expression on the incoming links of a WS-BPEL activity and its status decides whether the activity can start execution. A *transition condition* on a link is defined at its source and refers



to the condition that must hold good for the link to attain a true value. The probability that a link  $A_i$  assumes a true value is dependent on the successful completion of its source activity and its transition condition being evaluated to true. In our model,  $P(\text{transitionCondition}_{A_i} = \text{true})$  is obtained as an input.

$$P(A_i = \text{true}) = P(\mathcal{S}_{\text{source}(A_i)}) \cdot P(\text{transitionCondition}_{A_i} = \text{true}) \quad (1)$$

**Example:** In our passport application example, the link  $X$  connects the *invoke* activity for *DOB Verify* to that for *Make Passport*. Thus,  $P(X = \text{true})$  or simply  $P(X)$  may be computed as a product of  $P(S_{\text{invoke}_{\text{DOB Verify}}})$  and the input probability for success of the transition condition.

If we have an *AND* in the boolean expression of the join condition, we consider the intersection of events that the constituent links are true and in case of *OR* we determine the union of those events. To compute the probability that the join condition for an activity evaluates to true,  $P(\text{joinCondition}_X = \text{true})$ , we convert the boolean expression into a canonical Sum of Products (SOP) form which is evaluated with the help of the standard law of probability for union of events, assuming that the events  $P(A_i = \text{true})$  for all links  $A_i$  are independent. It may be noted that in this work we only carry out a control flow analysis on WS-BPEL processes, if we track data flow too we can do away with the assumption of links being independent and achieve more accurate QoS estimates.

**Example:** We determine the probability  $P((X \cup Y) \cap Z)$  for the join condition at the *Make Passport invoke* after we compute values of  $P(X)$ ,  $P(Y)$  and  $P(Z)$ .

A WS-BPEL activity,  $X$ , may start execution if the following conditions are met.

- The parent activity of  $X$  has started execution which guarantees that the control dependencies of the parent have completed execution.
- If  $X$  is a child of a  $\langle \text{sequence} \rangle$ , then the prior activity (if any) in the  $\langle \text{sequence} \rangle$  has successfully completed execution.
- If  $X$  contains incoming synchronization links then its join condition (implicit/explicit) has been evaluated to true.

Thus, in order to compute  $P(\text{start}_X)$ , we take a product of the following probabilities, if they are applicable: (a)  $P(\text{start}_{\text{parent}_X})$ , (b)  $P(\mathcal{S})$  of predecessor in *sequence*, (c)  $P(\text{joinCondition}_X = \text{true})$ . Note, that we consider  $P(\text{start}_{\text{parent}_X})$  only if  $X$  is one of the following - (i) first activity inside a *sequence*, (ii) activities inside a *flow* that do not have incoming links (iii) activities in all branches of *if* or *pick* (iv) activity inside a scope. The other children of a sequence / flow are directly or indirectly dependent on these first activities, so their dependency on their parent gets captured in our model. Loops are handled differently (See Section 4.3), and thus the calculation of  $P(\text{start})$  of a child activity of a loop does not consider  $P(\text{start}_{\text{loop}})$ .

**Example:** In case of the *Make Passport WS invocation*, we note that only (a) and (c) are applicable and  $P(\text{start}_{\text{invoke}_{\text{Make Passport}}})$  is given by  $P(\text{start}_{\text{if}}) \cdot P((X \cup Y) \cap Z)$ .

The start time (ST) for an activity is taken to be the maximum of the end time of its predecessor in *sequence*, the expected end times of all the source activities of its incoming links and the start time of its parent. For the activities inside message based events in *pick* and the receive activities (that are not start activities, i.e., *createInstance* = “no”), we also add their average waiting times to their start times.

The cost model computes for each activity the probability that its starts given its parent has already started. This probability, referred to as PC, is used extensively in our model for aggregation of the costs of child activities in order to estimate the expected cost of an activity.

$$\begin{aligned}
 PC_X &= P(start_X | start_{parent(X)}) \\
 &= \frac{P(start_X)}{P(start_{parent(X)})} \text{ Since, } P(start_{parent(X)} | start_X) = 1 \quad (2)
 \end{aligned}$$

### 4.3 Activity-Wise Rules for Determining $P(S)$ , $ET$ and $Cost$

In this section, we detail our formulations to estimate  $P(S)$ ,  $ET$  and  $Cost$  for every WS-BPEL activity. It may be noted that for any activity reliability modeling must be performed before determination of time and cost may take place.

**Basic Activities.** All *basic activities* except *invoke* have zero costs associated with them and are assumed to complete successfully and instantaneously when they start. Hence, the probability of successful completion,  $P(S)$  of a basic activity is equal to the probability that it gets to execute,  $P(start)$  and its end time( $ET$ ) is equal to its start time( $ST$ ). Only,  $ET$  of a *wait* activity is computed differently after adding the delay specified to its  $ST$ .

**Invoke.** *Invoke* activities denote the point of calling external web services. These may be prone to failures and have an associated latency and cost. For each *invoke* activity, the model expects as input the reliability ( $R_{ws}$ ), response time ( $T_{ws}$ ) and cost ( $C_{ws}$ ) of the web service bound to it.  $R_{ws}$  represents the conditional probability  $R_{ws}$  that an invocation to an external web service fails despite the call being made with proper arguments.  $R_{ws}$  and  $T_{ws}$  incorporate failures and latencies respectively that arise both at the service site or from the network. Now,  $P(start_{invoke})$  gives the probability that the *invoke* activity begins in a consistent state with proper arguments available. Thus, we may write:

$$P(S'_{invoke}) = R_{ws} \times P(start_{invoke}) \quad (3)$$

$$ET'_{invoke} = ST_{invoke} + T_{ws} \quad (4)$$

$$Cost_{invoke} = C_{ws} \quad (5)$$

However, an *invoke* activity may have *catch* blocks and *compensation handlers* attached to it and completion of an *invoke* activity would encompass their completion too. Thus, we may write the expression for  $P(S)$  and  $ET$  of an *invoke* activity after accounting for the attached catch blocks and compensation handler if there are any. QoS determination for handlers is discussed later in this section.

**Structured Activities.** QoS computation of structured activities require determination of QoS parameters of their children. We briefly describe here the rules for estimating  $P(S)$ ,  $ET$  and  $Cost$  for all structured activities. Table 2 lists the equations for QoS determination in structured activities.

**Table 2.** QoS of Structured Activities

Activity	$P(S)$	$ET$	$Cost$
Sequence	$P(S_{lastChild})$	$ET_{lastChild}$	$\sum_i PC_{child_i} \times Cost_{child_i}$
Flow	$\prod_{\forall_i} P(S_{sink_i})$	$Max_{\forall_i}(ET_{sink_i})$	$\sum_i PC_{child_i} \times Cost_{child_i}$
If	$\sum_i P(sel_i) \times P(S_{br_i})$	$\sum_i P(sel_i) \times ET_{br_i}$	$\sum_i P(sel_i) \times PC_{br_i} \times Cost_{br_i}$
Pick	$\sum_i P(sel_i) \times P(S_{evt_i})$	$\sum_i P(sel_i) \times ET_{evt_i}$	$\sum_i P(sel_i) \times PC_{evt_i} \times Cost_{evt_i}$
Loop	$P(start_i) \times P(S_{child})^n$	$ST_i + n \times ET_{child}$	$n \times Cost_{child}$

**Sequence.** An activity nested inside a *sequence* can only start if the previous activity in the sequence has been successful. Thus, if the  $i^{th}$  child of sequence is executing, then all child activities from the first to the  $(i - 1)^{th}$  can be taken to be complete. Therefore, we can model  $P(S)$  and end time of a *sequence* by that of its last child. The expected cost of a *sequence* is simply a weighted sum of the expected costs of all its child activities with their PCs being the weights.

**Flow.** A *flow* activity is deemed to complete only if all activities enclosed by it are complete. However, since the synchronization links in effect model the control flow of execution, it may be contended that the completion of all child activities without any outgoing links would mark the completion of the *flow*. Therefore,  $P(S)$  and end time for a *flow* activity may be modeled as an aggregation of that of the *sinks* which are children with no outgoing links. Cost of a *flow* is modeled exactly the same way as that of a *sequence*.

**Example:** The *flow* in the passport office workflow has only one sink namely the *if* activity. Thus, both  $P(S_{flow})$  and  $ET_{flow}$  will be given by those for the *if* activity.

**If.** An *if* activity is complete when the activity nested in the taken branch completes. It completes immediately if no *condition* evaluates to true and no *else* branch is specified. All QoS parameters -  $P(S)$ ,  $ET$  and  $Cost$ , of an *if* activity are calculated as weighted sums of the values of the same for the activities contained in all its branches, the weights being the probability,  $P(sel_i)$ , with which a branch gets selected for execution. Note that in cost modeling, the costs of the branch activities are relaxed by their PCs while aggregation.

**Example:** In the *If* activity in our example,  $P(S_{if})$  is computed by taking a weighted sum of that of the *invoke* for *Make Passport* and an *empty* activity ( $P(S_{empty})$  is always 1) assumed to be present in the non-existent else branch.

**Pick.** *Pick* activities are treated in a similar way as *if* activities; with probabilities of selection of each event being taken as input.

**Loops.** QoS modeling for the activity inside a loop may be performed independently without requiring QoS information of its parent activity or activities

outside the loop. This is facilitated by the WS-BPEL stipulation (See SA00070 in [1]) that synchronization links cannot enter into repeatable constructs by crossing their boundaries. The various iterations of the loop are assumed to be independent and in effect the loop construct is treated as a number of copies of the contained child activity running in sequence. In case of *while* and *repeatUntil*, the number of iterations,  $n$ , is obtained as an input to the model. In a *forEach* activity, the number of iterations,  $n$ , is obtained by parsing the values of  $\langle finalCounterValue \rangle$ ,  $\langle startCounterValue \rangle$  and the *completion condition* if specified. The special case where the loop is rolled in parallel (the *parallel* attribute being set to true) is handled by taking time taken by one iteration only whilst calculating response time. Again, PC of the child activity of a loop will always evaluate to 1 because it cannot have any dependencies, so there is no scope of relaxation of the cost of the child of a loop.

**Handlers.** The handlers in WS-BPEL impact QoS of *scopes* and *invoke* activities to which they may be attached. For lack of space, we present here only the QoS modeling for fault handlers. (Refer to [10] for a discussion on Compensation Handlers and Event Handlers)

The probability that fault handlers start execution is dependent upon the rate of faults thrown up by the web service invocation or scopes.

$$FaultRate_X = \begin{cases} 1 - R_{ws} & \text{if X is an invoke,} \\ 1 - P(\mathcal{S}_{scopeChild} | start_{scope}) & \text{if X is a scope} \end{cases} \quad (6)$$

The model assumes as input, the fraction of faults caught by each catch block *FractionCapture*, out of the total number of faults produced by its scope. Thus, the probability that a catch block starts may be given by:

$$P(start_{catch_i}) = FaultRate_X \times FractionCapture \times P(start_X) \quad (7)$$

where X = scope/invoke

$P(\mathcal{S})$ ,  $ET$  and  $Cost$  of a catch block are taken to be the same as that of the activity nested in it. The fraction of faults removed  $FFR$  by all catch blocks in a fault handler may be computed as:

$$FractionFaultRemoval(FFR) = \sum_{\forall i} (FractionCapture_i \times P(\mathcal{S}_{catch_i})) \quad (8)$$

After taking into account the fraction of faults removed, the improved probability of the *invoke* / *scope* will stand as below:

$$P(\mathcal{S}''_X) = P(\mathcal{S}'_X) + FaultRate_X \times FFR \quad (9)$$

where, X may be *invoke*/*scope*.

Time and Cost of a fault handler (FH) may be given as:

$$T_{FH} = Max_{\forall i} (FractionCapture_i \times FaultRate \times (ET_{catch_i} - ST_{catch_i})) \quad (10)$$

$$Cost_{FH} = FaultRate \times \sum_{\forall i} FractionCapture_i \times Cost_{catch.Activity_i} \quad (11)$$

**Scope.** The following equations model QoS of a scope in presence of fault handlers(FH), compensation handlers (CH) and event handlers (EH). Similar equations may be written for *invoke* to incorporate the effects of attached handlers.

$$P(\mathcal{S}_{scope}) = P(\mathcal{S}'_{scope} \times P(\mathcal{S}_{CH}) \times P(\mathcal{S}_{EH})) \quad (12)$$

$$ET_{scope} = \text{Max}(ET'_{child} + T_{FH}, ET_{CH}, ET_{EH}) \quad (13)$$

$$\text{Cost}_{scope} = \text{Cost}_{child} + \text{Cost}_{FH} + \text{Cost}_{CH} + \text{Cost}_{EH} \quad (14)$$

The above equations when applied to the *(process)* (that is nothing but a special form of scope) will give the QoS of the composite web service written in BPEL.

**Suppression of Join Failures.** The entire exposition above assumes that a *bpel : joinFailure* is thrown whenever a join condition is not satisfied. However, if *suppressJoinFailure* attribute is set to *yes*, failure of a join condition results in the activity being skipped and a false status being propagated on all its outgoing links with no fault generation. Since a skipped activity is also deemed to be complete, the expression for the probability of successful completion is improved by the probability of the activity being skipped.

$$P(\mathcal{S}') = P(\text{joinCondition} = \text{true}) \times P(\mathcal{S}) + (1 - P(\text{joinCondition} = \text{true})) \quad (15)$$

However, in the evaluation of  $P(\text{link} = \text{true})$  (See Equation [11](#)), the probability of successful completion of the source will substituted by old  $P(\mathcal{S})$  and not  $P(\mathcal{S}')$  because a failed status is propagated on each of the outgoing links of a skipped activity.

## 5 Impact of Fault Tolerance on QoS Computation

During orchestration of a web service composition, the designer may find that all web services available to perform some task do not meet reliability requirements or show huge variations in their response times. In such cases, fault tolerant constructs may be used to create dependable web services out of undependable ones and attain the desired reliability and performance levels. Our QoS model and the tool supports four conventional fault tolerance (FT) techniques - *N-version programming*, *Recovery Blocks*, *Return Fastest Response* and *Deadline Mechanisms* - and helps in QoS determination after application of these FT-constructs. The first two constructs focus on reliability improvement and the latter two seek to enhance performance. Again for lack of space, we only derive expressions for QoS computation of the *Deadline Mechanism* construct in this paper (Details of the other 3 constructs can be found in [10](#)). The following discussion assumes for every task in the workflow we have a set of web services  $(A_1, A_2, \dots, A_n)$ , having diverse designs, for which QoS parameters - reliability ( $R$ ), response time ( $T$ ) and cost ( $C$ ) are known.

*Deadline mechanism* supports setting deadlines for completion of tasks and provision forking off redundant services if a primary service does not complete

within some specified length of time. In our model, the user sets a hard deadline,  $HD$ , for completion of a task. Moreover, the designer specifies for each alternate web service  $A_i$ , the time instant  $TF_{A_i}$  when it may be fired if no response is received from services that have been running. In any invocation of a *Deadline Mechanism (DM)* block, the service that returns first gets counted. For each service in a DM block, we find the probability that it returns the first response and use these probabilities to compute the reliability and time of a DM block.

$$R_{DM} = \sum_{i=0}^n P(T'_{A_i} = \text{Min}\{T'_{A_1}, T'_{A_2}, \dots, T'_{A_n}\} \text{ and } T'_{A_i} < HD) \times R_{A_i} \quad (16)$$

where,  $T'_{A_i} = T_{A_i} + TF_{A_i}$

$$T_{DM} = \text{Min}_{\forall i} \{T'_{A_i}\} \quad (17)$$

Again, a service inside a DM block may start only if a successful response has not been generated by other services till the point of its firing. Thus,

$$P(\text{Start}_{A_i}) = P(TF_{A_i} \geq \text{Min}\{T'_{A_1}, T'_{A_2}, \dots, T'_{A_n}, HD\}) \quad (18)$$

Cost of a DM block is a weighted sum of the costs of all services in it, the weights being the  $P(\text{Start})$  of the services, i.e.,  $\sum_{i=0}^n P(\text{Start}_{A_i}) \times C_{A_i}$ .

## 6 Implementation

We have implemented our QoS model for WS-BPEL 2.0 processes in a stand-alone software using Java 1.5. The tool accepts as input a BPEL source and parses it to ask the user for the various control flow parameters (See Inputs to the Model in Section 4) and the values for reliability, time and cost for each web service invocation present in it. The tool outputs the three QoS parameters for the overall process and also shows  $P(S)$ ,  $ET$  and  $Cost$  for every activity. For improving QoS through fault tolerant constructs the user has to provide WSDLs as well as QoS values of the redundant web services and the web services to be used for voting and assertion checking. Figure 2 shows the block diagram of our implementation and Figure 3 shows some of the user interfaces that are a part of the tool.

## 7 Related Work

Cardoso’s thesis [3] is the seminal work in literature to propose a framework for estimation of QoS in web service processes. He proposes Stochastic Workflow Reduction (SWR) to arrive at QoS estimates for the overall workflow, provided the QoS values for all tasks in the workflow are known. The SWR algorithm repeatedly applies a reduction on various structured constructs until only one atomic task remains. He introduces reduction rules for sequential, parallel, conditional, loop and fault tolerant systems. However, there is a restrictive rider

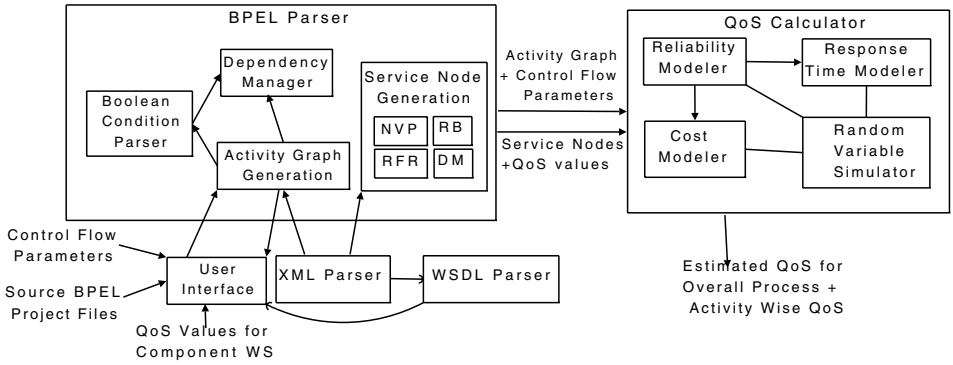


Fig. 2. Block Diagram of Implementation

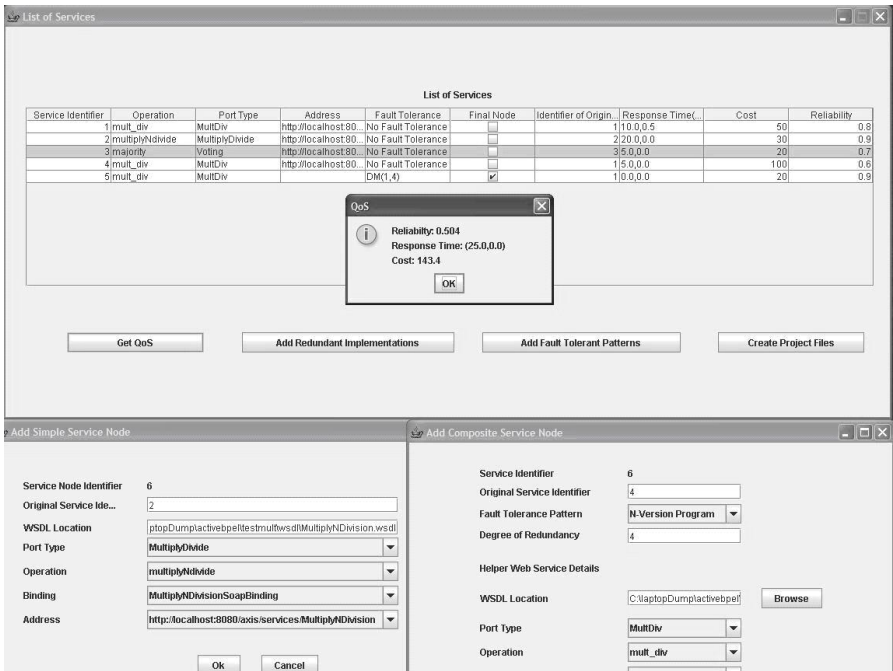
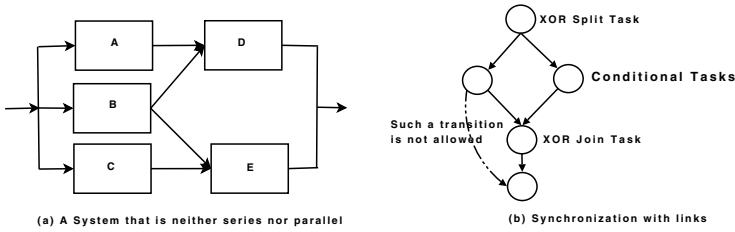


Fig. 3. User Interfaces in QoS tool

added with most of the reduction rule-sets given above. For example, in the sequential system the start task cannot be a split and the end task cannot be a join. Cardoso’s model adapts from the reductions used in standard reliability theory for computing reliability of series-parallel systems [11], [12]. But the model is not capable of handling complex systems such as the one shown in Figure 4(a) that can neither be reduced to a series nor decomposed as a parallel system.



**Fig. 4.** Limitations of Structured Workflows

Again, presence of goto-like transitions that extend from one structured construct to another as shown in Figure 4(b) prevent application of the proposed reductions.

Hwang et al. [13] propose a probabilistic framework for QoS computation. They extend Cardoso's model to have each QoS parameter for a web service represented by a discrete random variable having a certain probability mass function (PMF). Canfora et al. [14] apply Cardoso's QoS model with minor modifications in their middleware that uses genetic algorithms for QoS aware composition and replanning. Zeng et al. [4] model composite web services as state charts and put forward aggregation functions to ascertain QoS of *execution plans*. Jaeger et al. [15] propose aggregation of QoS dimensions on the workflow patterns listed in Van der Aalst's seminal work [16]. The approach is an elegant one but the authors do not explain how to dig out such workflow patterns from a process an to carry out an implementation of the same. D'Ambrogio and Bocciarelli [17] propose a model driven approach wherein a BPEL process is described by an UML model which is then annotated with performance data to obtain a LQN (Layered Queueing Network) model that may be solved to predict performance. Although the process of conversion of models built according to the UML Profile into BPEL has been thoroughly described in [18], the complex control flow offered by BPEL have not been exhaustively mapped back onto the UML profile. However, BPEL to UML transformation is an active research topic [19].

## 8 Conclusions

We have presented a comprehensive QoS determination model for WS-BPEL processes. Our QoS calculator provides values for reliability, response time and cost for each activity / scope / handler. WS-BPEL is capable of expressing arbitrarily complex structures with the help of synchronization links between activities and facilitates fault handling, event driven programming and backward recovery through compensation handlers. Although QoS research in workflows has gained importance, no QoS estimation technique exists in literature to the best of our knowledge, that captures graph-like control flow structures or supports the kind of behavior that is rendered through WS-BPEL handlers. Our QoS tool enables the designer to modify parts of the BPEL process workflow



(differently organize the control flow or add/remove fault handlers and compensation handlers etc.), add fault tolerance constructs to it and check for QoS improvement.

The limitations of the model are: (i) It does not support detection of mutual exclusiveness of links at a join making estimation of  $P(start)$  less accurate in face of mutually exclusive paths. (ii) It does not handle *isolated scopes* that provide control over concurrent access of shared resources. (iii) Forced termination behavior is not captured by the model and it does not deal with termination handlers. All of these issues lay scope for future work in the area.

## References

1. OASIS WS-BPEL Technical Committee, Web Services Business Process Execution Language Version 2.0 (2007)
2. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of Web Services Composition Languages: The Case of BPEL4WS. In: Proceedings of the 22nd International Conference on Conceptual Modeling, ER (2003)
3. Jorge, A., Cardoso, S.: Quality of Service and Semantic Composition of Workflows, Ph.D. Thesis, University of Georgia, Athens, Georgia (2002)
4. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions in Software Engineering (2004)
5. Avizienis, A., Chen, L.: On the implementation of N-version programming for software fault tolerance during execution. In: Proceedings of IEEE COMPSAC (1977)
6. Randell, B.: System structure for software fault tolerance. In: Proceedings of the international conference on Reliable software (1975)
7. Dobson, G.: Using WS-BPEL to Implement Software Fault Tolerance for Web Services. In: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 126–133 (2006)
8. Gorbenko, A., Kharchenko, V., Popov, P., Romanovsky, A., Boyarchuk, A.: Development of Dependable Web Services out of Undependable Web Components, School of Computing Science, University of Newcastle (2004)
9. Looker, N.M., Xu, M.J.: Increasing Web Service Dependability Through Consensus Voting. In: Computer Software and Applications Conference (2005)
10. Mukherjee, D.: QoS in WS-BPEL Processes, M.Tech. Thesis, Department of Computer Science & Engineering, Indian Institute of Technology, Delhi (2008)
11. Grant Ireson, W., Coombs Jr., C.F., Moss, R.Y.: Handbook of Reliability Engineering and Management. McGraw Hill, New York (1996)
12. Hoyland, A., Rausand, M.: System Reliability Theory: Models and Statistical Methods. John Wiley and Sons, Chichester (1994)
13. Hwang, S.-Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. Journal of Information Sciences (2007)
14. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation (2005)

15. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: Proceedings of the Enterprise Distributed Object Computing Conference (2004)
16. Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. *Journal of Distributed Parallel Databases* (2003)
17. D'Ambrogio, A., Bocciarelli, P.: A model-driven approach to describe and predict the performance of composite services. In: Proceedings of the 6th international workshop on Software and performance (2007)
18. Amsden, J., Gardner, T., Griffin, C., Iyengar, S.: Draft UML 1.4 profile for automated business processes with a mapping to BPEL 1.0 (2004)
19. Reiter, T.: Transformation of Web Service Specification Languages into UML Activity Diagrams, Diploma thesis, University of South Australia (2005)

# An Initial Approach to Explaining SLA Inconsistencies\*

Carlos Müller, Antonio Ruiz-Cortés, and Manuel Resinas

Dpto. Lenguajes y Sistemas Informáticos  
ETS. Ingeniería Informática - Universidad de Sevilla (Spain - España)  
41012 Sevilla (Spain - España)  
{cmuller, aruiz, resinas}@us.es

**Abstract.** An SLA signed by all interested parties must be created carefully, avoiding contradictions between terms, because their terms could carry penalties in case of failure. However, this consistency checking may become a challenging task depending on the complexity of the agreement. As a consequence, an automated way of checking the consistency of an SLA document and returning the set of inconsistent terms of the agreement would be very appealing from a practical point of view. For instance, it enables the development of software tools that make the creation of correct SLAs and the consistency checking of imported SLAs easier for users. In this paper, we present the problem of explaining WS-Agreement inconsistencies as a constraint satisfaction problem (CSP), and then we use a CSP solver together with an explanation engine to check the consistency and return the inconsistent terms. Furthermore, a proof-of-concept using Choco solver in conjunction with the Palm explanation engine has been developed.

**Keywords:** Service Level Agreement, WS-Agreement, Consistency Checking, Debugging, Quality of Service.

## 1 Introduction

SLAs consist of a set of terms that include information about functional features, non-functional guarantees, compensation, termination terms and any other terms with relevant information to the agreement. An agreement signed by all interested parties should be redacted carefully because a failure to specify their terms could carry penalties to the initiating or responding party. Therefore, agreement terms should be specified in a consistent way, avoiding contradictions between them. However, depending on the complexity of the agreement, this may become a challenging task. For instance, in a scenario in which a provider offers computing services to other organizations, an SLA could be agreed that includes non-functional attributes such as: the *availability* -in percentage-, the *mean time*

---

\* This work has been partially supported by the European Commission (FEDER), Spanish Government under CICYT project Web-Factories (TIN2006-00472), and project P07-TIC-2533 funded by the Andalusian local Government.

between two consecutive requests of the service (MTBR) -in seconds-, and the efficiency. Assuming that: Availability ranges between [90..100], MTBR ranges between  $\in [5..60]$  and Efficiency = Availability/MTBR. If the SLA includes a term obligating to guarantee Efficiency > 20, at first sight the SLA is consistent. However the highest valid value for efficiency is  $100/5=20$ , so this term cannot be satisfied. Therefore, a consistency checker that automatically checks the SLA for inconsistencies between its terms would be very appealing from a practical point of view.

Furthermore, it is of interest not only to obtain an automated way of checking the consistency of an SLA document, but also to return an explanation if the document is inconsistent. This explanation is the set of inconsistent terms of the agreement. So, in the previous scenario we would obtain as debugging information that the inconsistent terms are: [(Efficiency > 20), (Availability  $\in [90..100]$ ), (MTBR  $\in [5..60]$ ), (Efficiency = Availability/MTBR)], because they are inconsistent terms. This automated consistency checking enables the implementation of a software tool which makes the creation of correct SLAs and the consistency checking of imported SLAs easier for users.

Nevertheless, as far as we know, the consistency of SLAs has been taken for granted by most authors. In this paper, we describe a mechanism to check and explain SLAs specified with WS-Agreement [2], which is a proposed recommendation of the Open Grid Forum (OGF) which provides a schema for defining SLAs and a protocol for creating them. To this end, we map the terms of a subset of the WS-Agreement document into a constraint satisfaction problem (CSP). Then we use the CSP as an input for a CSP solver with an explanation engine, which will return the set of inconsistent constraints. Finally, we trace back the constraints to the original SLA terms in order to give useful debugging information to users.

As a proof-of-concept, we have developed a prototype of our consistency analyser using Choco solver [1] in conjunction with the Palm explanation engine [3]. This prototype is available for testing at <http://www.isa.us.es> in the tools section.

This paper is structured as follows. Section 2 presents some background on constraint satisfaction problems and WS-Agreement. Section 3 details the subset of WS-Agreement which is used to explain the SLA inconsistencies in 3.1 and its mapping to CSP in 3.2. Section 4 describes our process to explain the WS-Agreement\* inconsistencies. Section 5 shows our proof-of-concept. Section 6 reports on the related proposals. Finally, Section 7 details our conclusions and future work.

## 2 Preliminaries

### 2.1 Constraint Satisfaction Problems

Constraint Satisfaction Problems [8] have been the object of research in Artificial Intelligence over the last few decades. A Constraint Satisfaction Problem (CSP)

is defined as a set of variables, each ranging on a finite domain, and a set of constraints restricting all the values that these variables can take simultaneously. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously. These are some basic definitions of what a CSP is.

**Definition 1 (CSP).** *A CSP is a three-tuple of the form  $(V, D, C)$  where  $V \neq \emptyset$  is a finite set of variables,  $D \neq \emptyset$  is a finite set of domains (one for each variable) and  $C$  is a constraint defined on  $V$ .*

Consider, for instance, the CSP:  $(\{a, b\}, \{[0..2], [0..2]\}, \{a + b < 4\})$

**Definition 2 (Solution).** *Let  $\psi$  be a CSP, a solution of  $\psi$  is whatever valid assignment of all elements in  $V$  that satisfies  $C$ .*

In the previous example, a possible solution is  $(2, 0)$  since it verifies that  $2+0 < 4$ .

**Definition 3 (Solution space).** *Let  $\psi$  be a CSP of the form  $(V, D, C)$ , its solution space denoted as  $sol(\psi)$  is made up of all its possible solutions. A CSP is satisfiable if its solution space is not empty.*

$$sol(\psi) = \{S \mid \forall s_i \cdot s_i \in S \Rightarrow C(s_i) = true\}$$

In the previous example there are eight solutions. The only assignment that does not satisfy  $a + b < 4$  is  $(2, 2)$ . Nevertheless, if we replace the constraint with  $a + b < -1$ , then the CSP is not satisfiable.

In many real-life applications, if a CSP has no solution, we would like to know which set of constraints are responsible for this situation. This can be done by interpreting the CSP as an explanation problem.

**Definition 4 (Explanation problem).** *Let  $\epsilon$  be a CSP of the form  $(V, D, C)$  with an empty solution space:  $sol(\epsilon) = \emptyset$ . It is considered to be an explanation problem if its objective is to find a set of constraints  $C' \subset C$  that cannot be satisfied.*

**Definition 5 (Explanations).** *Let  $\epsilon$  be an explanation problem, the resulting set of inconsistent constraints  $C'$  are known as the explanations for the problem. They are divided into two parts: a subset of the original set of constraints  $C' \subset C$  and a subset of decision constraints introduced so far in the search of solutions  $(dc_1, \dots, dc_k)$ .*

As defined in [3], a contradiction explanation, also known as “nogood” [7], is a subset of the constraints of the problem that, left alone, leads to a contradiction (no feasible solution contains a nogood).

The previous CSP example, with the constraint replaced with  $a + b < -1$ , is not satisfiable, and by interpreting it as an explanation problem the explanation

engine should obtain as the set of inconsistent constraints:  $[(a + b < -1), (a >= 0), (b >= 0)]$ .

## 2.2 WS-Agreement in a Nutshell

WS-Agreement specifies an XML-based language and a protocol for advertising the capabilities and preferences of service providers, and creating agreements based on agreement offers. The structure of an agreement in WS-Agreement comprises:

- **Name:** it identifies the agreement and can be used for reference.
- **Context:** it includes information such as the name of the parties and their roles as initiator or responder in the agreement. Additionally, it can include other important information for the agreement.
- **Terms:** they are grouped by the following term compositors: **ExactlyOne**, **OneOrMore**, or **All**. The two main types of terms are:
  - *Service terms:* they provide service information by means of:
    - \* Service description terms and service references, which includes information to instantiate or identify the services and operations involved in the agreement.
    - \* Service properties, which includes the measurable properties that are used in expressing guarantee terms. They consist of a set of variables whose values can be established inside the service description term, and whose domain can be established by the `metric` attribute pointing to an external XML document.
  - *Guarantee terms:* they describe the service level objectives (SLO) agreed by a specific obligated party, either using a free-form element or using a key performance indicator. It also includes the scope of the term (e.g. if it applies to a certain operation of a service or the whole service itself), and a qualifying condition that specifies the validity condition under which the term is applied.

Figure 1 depicts an example of a WS-Agreement between a computing services provider and a consumer. It defines several service properties whose domain is specified in an external XML document (depicted in Figure 2). Note that other XML documents, such as an XML Schema definition, could have been used instead. The service properties defined in the WS-Agreement document of Figure 1 are the following ones:

- the *availability* -integer from 1 to 100-
- the *mean time between two consecutive requests of the service* (MTBR) -integer greater than 1-
- the *mean time to response* (MTTR) -integer greater than 1-
- the *initial cost for the service* (InitCost) -integer greater than 1-
- the *final cost for the service* (Cost) -integer greater than 1-

- the *increase of the cost if the MTBR  $\geq 10$*  (ExtraMTBRCost) -integer greater than 1-
- the *increase of the cost if the MTTR  $\geq 05$*  (ExtraMTTRCost) -integer greater than 1-

We have extracted the following information from the SLA:

- $MTBR \in [5..60]$ .
- $MTTR \in [1..10]$ .
- If  $MTBR \geq 10$  Then Availability  $\in [90..100]$ .
- If  $MTBR \geq 10$  Then Availability  $\in [95..100]$ .
- Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost.
- If  $MTBR \geq 10$  Then ExtraMTBRCost = 15.
- If  $MTTR \geq 05$  Then ExtraMTTRCost = 15.
- If  $MTBR \geq 10$  and  $MTTR \geq 05$  Then ExtraMTBRCost = 0 and ExtraMTTRCost = 0.

### 3 Mapping WS-Agreement onto CSP

#### 3.1 WS-Agreement\* as a Subset of WS-Agreement

Due to the flexibility and extensibility of WS-Agreement, we focus in this paper on a subset of WS-Agreement that is a bit less expressive than WS-Agreement, but still useful for our purpose. The subset of WS-Agreement is called WS-Agreement\*. A WS-Agreement\* document ( $\alpha$ ) is composed of the following elements:

- Service properties must define all variables that are used in the guarantee terms. In addition, attribute `metric` of the variables is mandatory. This attribute must point to another XML document or schema that provides the data type and a general range of values ( $\delta$ ) for each service property. Figure 2 shows an example of an ad-hoc XML document that includes the mentioned information for service properties of the example of Figure 1, although other XML documents could be used. XML node `Location` ( $\lambda_v$ ) of each variable establishes the specific XML node inside the service description term where it is defined the value for such variable ( $\text{value}(\lambda_v)$ ).
- Terms ( $T$ ) can be composed using the three term compositors defined in WS-Agreement: `All` ( $\odot$ ), `ExactlyOne` ( $\otimes$ ), and `OneOrMore` ( $\oplus$ ).
- Service description terms can be included but only to impose specific value definitions for each variable ( $v$ ) of service properties. Other service description terms could be added but they are ignored when checking the SLA consistency.
- Guarantee terms ( $\gamma$ ) can be included. Both qualifying condition ( $\kappa$ ) and the SLOs ( $\sigma$ ) must be defined as constraints on the variables defined in the service properties, and only to those variables (i.e.  $\kappa \in C$  and  $\sigma \in C$ ). The scope of a guarantee term cannot be established, but all guarantee terms apply to the whole service.

```

<Agreement ''id=exampleScenario''>
  <Context> ... </...>
  <All>
    <ServiceDescriptionTerm Name='ComputingService''>
      <...>
        ... </...>
        <InitCost> 20 </...>
        ... </...>
      </...>
    </ServiceDescriptionTerm>
    <ServiceProperties>
      ...<Variable name='Availability'' metric='metricXML:Availability''>
        <Location> \\Availability </Location>
      </Variable>
      <Variable name='MTBR'' metric='metricXML:MTBR''>
        <Location> \\MTBR </Location>
      </Variable>
      <Variable name='MTR'' metric='metricXML:MTR''>
        <Location> \\MTR </Location>
      </Variable>
      <Variable name='InitCost'' metric='metricXML:InitCost''>
        <Location> \\InitCost </Location>
      </Variable>
      <Variable name='Cost'' metric='metricXML:Cost''>
        <Location> \\Cost </Location>
      </Variable>

      <Variable name='ExtraMTBRCost'' metric='metricXML:ExtraMTBRCost''>
        <Location> \\ExtraMTBRCost </Location>
      </Variable>

      <Variable name='ExtraMTTRCost'' metric='metricXML:ExtraMTTRCost''>
        <Location> \\ExtraMTTRCost </Location>
      </Variable>...
    </ServiceProperties>
    <GuaranteeTerm Name='MTBRDomain''>
      <SLO> MTBR >= 5 and MTBR <= 60 </...>
    </GuaranteeTerm>
    <GuaranteeTerm Name='MTRDomain''>
      <SLO> MTR >= 1 and MTR <= 10 </...>
    </GuaranteeTerm>
    <GuaranteeTerm Name='CostDefinition''>
      <SLO> Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost </...>
    </GuaranteeTerm>
    <ExactlyOne>
      <GuaranteeTerm Name='LowerAvailabilityDomain''>
        <QualifyingCondition> MTBR >= 10 </...>
        <SLO> Availability >= 90 and Availability <= 100 </...>
      </GuaranteeTerm>
      <GuaranteeTerm Name='HigherAvailabilityDomain''>
        <QualifyingCondition> MTBR < 10 </...>
        <SLO> Availability >= 95 and Availability <= 100 </...>
      </GuaranteeTerm>
    </ExactlyOne>
    <OneOrMore>
      <GuaranteeTerm Name='MTBRIncrement''>
        <QualifyingCondition> MTBR < 10 </...>
        <SLO> ExtraMTBRCost = 15 </...>
      </GuaranteeTerm>
      <GuaranteeTerm Name='MTRIncrement''>
        <QualifyingCondition> MTR < 05 </...>
        <SLO> ExtraMTTRCost = 15 </...>
      </GuaranteeTerm>
      <GuaranteeTerm Name='CheaperCost''>
        <QualifyingCondition> MTBR >= 10 and MTR >= 05 </...>
        <SLO> ExtraMTBRCost = 0 and ExtraMTTRCost = 0 </...>
      </GuaranteeTerm>
    </OneOrMore>
  </All>
</Agreement>

```

Fig. 1. Example of a WS-Agreement document with all kinds of compositors



```

<metricXML>
  <Availability type='integer' min='1' max='100' />
  <MTBR type='integer' min='1' max='unbounded' />
  <MTTR type='integer' min='1' max='unbounded' />
  <InitCost type='integer' min='1' max='unbounded' />
  <Cost type='integer' min='1' max='unbounded' />
  <ExtraMTBRCost type='integer' min='1' max='unbounded' />
  <ExtraMTTRCost type='integer' min='1' max='unbounded' />
</metricXML>

```

Fig. 2. Ad-hoc XML document for the variable domains of Figure 1

Note that, although WS-Agreement\* is not as expressive as WS-Agreement, it does allow for the expression of complex agreements. For instance, the agreement depicted in Figure 1 is compatible with WS-Agreement\*.

Thus, we can formally define an agreement specified with WS-Agreement\* as follows:

**Definition 6 (A WS-Agreement\* document).** A WS-Agreement\* document  $\alpha$  is a set of variables  $v_i$ , variable domains  $\delta_i$ , and a set of terms  $T$ , including service description terms, guarantee terms and terms compositors as follows:

$$\alpha = (v_1, \dots, v_n, \delta_1, \dots, \delta_n, T_1, \dots, T_m), \text{ where } T_i = \left\{ \begin{array}{l} \lambda_v \\ \gamma \\ T_{i_1} \odot \dots \odot T_{i_k} \\ T_{i_1} \otimes \dots \otimes T_{i_k} \\ T_{i_1} \oplus \dots \oplus T_{i_k} \end{array} \right\}$$

### 3.2 Mapping WS-Agreement\* onto CSP

Figure 3 depicts the mapping ( $\mu$ ) of a WS-Agreement\* document ( $\alpha$ ) onto an equivalent CSP, ( $\psi_\alpha$ ). The variables ( $v$ ) defined inside the service properties are the CSP variables; the variable domains ( $\delta$ ) included in the document specified by the metric attribute are the CSP variable domains; and the constraints from the service description terms ( $\lambda_v$ ), guarantee terms ( $\gamma$ ) and term compositors ( $\odot$  as a logic “AND”,  $\otimes$  as logic “XOR”, and  $\oplus$  as logic “OR”) are the CSP constraints.

Thus, in general, our WS-Agreement\* to CSP mapping can be defined as follows:

**Definition 7 (Mapping an WS-Agreement\* to CSP).** The mapping ( $\mu : \alpha \rightarrow \psi$ ) of a WS-Agreement\* document ( $\alpha$ ) to a CSP ( $\psi$ ) can be defined as follows:

$$\begin{aligned} \mu(\alpha) &= \mu(v_1, \dots, v_n, \delta_1, \dots, \delta_n, T_1, \dots, T_m) = \\ &= (\{v_1, \dots, v_n\}, \{\delta_1, \dots, \delta_n\}, \{\mu_T(T_1, \dots, T_m)\}) = \psi_\alpha \end{aligned}$$

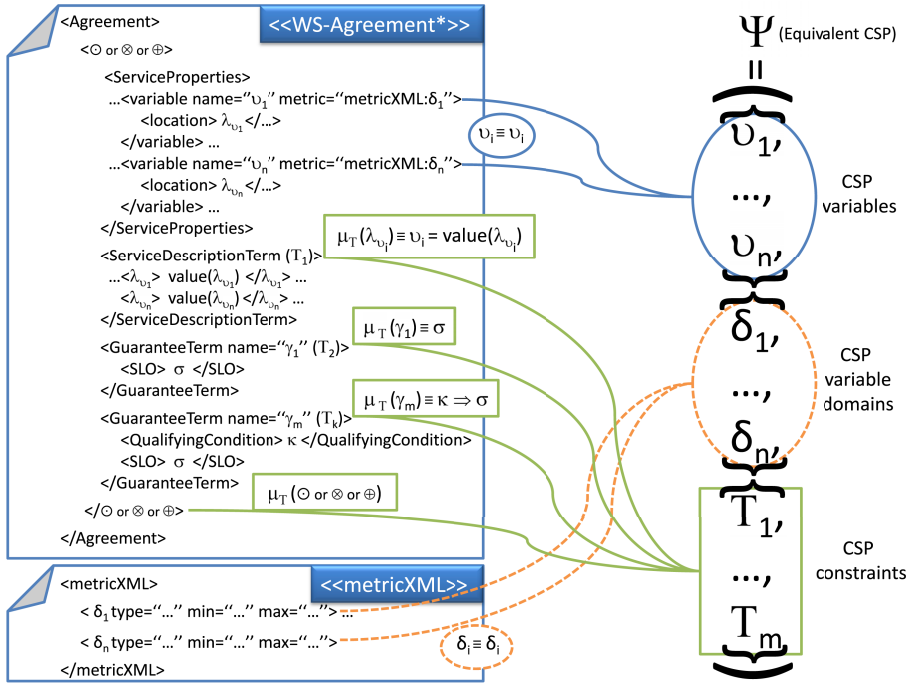


Fig. 3. Summary of WS-Agreement\* to CSP mapping

where  $\mu_T : T \rightarrow C$  is a mapping function of terms into constraints defined as follows:

$$\mu_T(T) \equiv \begin{cases} v = \lambda_v & \text{if } T \text{ is a service description term } (\lambda_v) \\ \sigma & \text{if } T \text{ is a guarantee term without qualifying condition } (\gamma_\sigma) \\ \kappa \Rightarrow \sigma & \text{if } T \text{ is a guarantee term with a qualifying condition } (\gamma_{\sigma,\kappa}) \\ \bigwedge_{i=1}^k \mu_T(T_i) & \text{if } T \text{ is a composite term } (T_1 \odot \dots \odot T_k) \\ \bigwedge_{i=1}^k \mu_T(T_i) \Leftrightarrow (\bigwedge_{j=1, j \neq i}^k \neg \mu_T(T_j)) & \text{if } T \text{ is a composite term } (T_1 \otimes \dots \otimes T_k) \\ \bigvee_{i=1}^k \mu_T(T_i) & \text{if } T \text{ is a composite term } (T_1 \oplus \dots \oplus T_k) \end{cases}$$

Using this mapping, the  $\psi_\alpha$  for the example of Figure 1 is mapped as follows:

$$\psi_\alpha = ( \{ \text{Availability, MTBR, MTTR, InitCost, Cost, ExtraMTBRCost, ExtraMTTRCost}, \\ \{ [ 1 \dots 100 ], [ 1 \dots \infty ), [ 1 \dots \infty ), [ 1 \dots \infty ), [ 1 \dots \infty ), [ 1 \dots \infty ), [ 1 \dots \infty ), \}, \\ \{ \text{InitCost} = 20, \\ \text{MTBR} \geq 5 \text{ and } \text{MTBR} \leq 60, \\ \text{MTTR} \geq 1 \text{ and } \text{MTBR} \leq 10, \\ \text{Cost} = \text{InitCost} + \text{ExtraMTBRCost} + \text{ExtraMTTRCost}, \\ ((\text{MTBR} \geq 10) \Rightarrow (\text{Availability} \geq 90 \text{ and } \text{Availability} \leq 100)) \Leftrightarrow \\ \Leftrightarrow \neg ((\text{MTBR} \neq 10) \Rightarrow (\text{Availability} \geq 95 \text{ and } \text{Availability} \leq 100)), \\ (\text{MTBR} \neq 10 \Rightarrow \text{ExtraMTBRCost} = 15) \vee \\ \vee (\text{MTTR} \neq 5 \Rightarrow \text{ExtraMTTRCost} = 15) \vee \\ \vee ((\text{MTBR} \geq 10 \text{ and } \text{MTTR} \geq 5) \Rightarrow \text{ExtraMTBRCost} = 0 \text{ and } \text{ExtraMTTRCost} = 0) \}$$

The following table denotes constraints mapped from the example of Figure 1.

**Table 1.** Mapping example terms to CSP constraints

Example term ( $T_i$ ) name	Equivalent mapped ( $\mu_T(T_i)$ )
InitCost	InitCost = 20
MTBRDomain	MTBR $\zeta$ = 5 and MTBR $\jmath$ = 60
MTTRDomain	MTTR $\zeta$ = 1 and MTTR $\jmath$ = 10
CostDefinition	Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost
LowerAvailabilityDomain	(MTBR $\geq$ 10) $\Rightarrow$ (Availability $\geq$ 90 and Availability $\leq$ 100)
HigherAvailabilityDomain	(MTBR $\jmath$ 10) $\Rightarrow$ (Availability $\geq$ 95 and Availability $\leq$ 100)
MTBRIncrement	(MTBR $\jmath$ 10) $\Rightarrow$ (ExtraMTBRCost = 15)
MTTRIncrement	(MTTR $\jmath$ 5) $\Rightarrow$ (ExtraMTTRCost = 15)
CheaperCost	(MTBR $\geq$ 10 and MTTR $\geq$ 5) $\Rightarrow$ (ExtraMTBRCost = ExtraMTTRCost = 0)
ExactlyOne	(LowerAvailabilityDomain $\Leftrightarrow \neg$ HigherAvailabilityDomain) $\wedge$ (HigherAvailabilityDomain $\Leftrightarrow \neg$ LowerAvailabilityDomain)
OneOrMore	MTBRIncrement $\vee$ MTTRIncrement $\vee$ CheaperCost
All	InitCost $\wedge$ MTBRDomain $\wedge$ MTTRDomain $\wedge$ CostDefinition $\wedge$ ExactlyOne $\wedge$ OneOrMore

## 4 Checking and Explaining WS-Agreement\* Inconsistencies

Checking the consistency of WS-Agreement documents involves both syntactic and semantic checking. The former involves checking the document against the WS-Agreement XML Schema. The latter, however, is harder to check on and is thus the focus of this paper.

Figure 4 depicts our consistency checking process. In this scenario, it is imposed by a human error, an MTBR value definition inside the service description term with a value of 61. Thus, the  $\psi_\alpha$  would not be satisfiable by the MTBR domain definition. As depicted in Figure 4, we propose to use the agreement specified with WS-Agreement\* in conjunction with the XML document which defines the variables metrics as the two inputs for a mapping component which implements the mapping defined in section 3.2. Once the agreement is mapped to a CSP, the explanation engine of the CSP explainer component will obtain whether the SLA document is consistent or not. In the latter case, the component sends to a tracing component the explanations for the problem. In this case, the explanations for the problem are  $\epsilon_{MTBR}$ : [MTBR = 61; MTBR  $\geq$  5 and MTBR  $\leq$  60]. The tracing component converts the explanations into the equivalent inconsistent original terms in order to give useful information to users. This is done by naming constraints mapped from an SLO as the name of the guarantee term which includes it; and if the constraint was mapped from a value

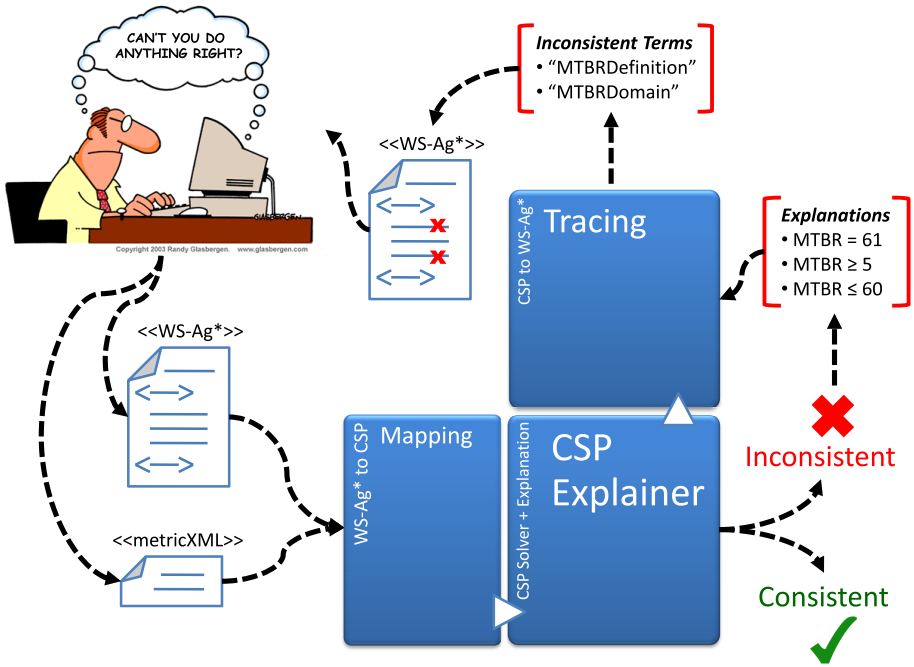


Fig. 4. WS-Agreement compliant process for explaining SLA inconsistencies

definition inside a service description term, we name it “SDT”, concatenate with the name of the variable. Then, the previous explanation for MTBR would be traced by us to the inconsistent term “SDTMTBR” constraint, showing that it is the MTBR value assignment and the domain restriction of the “MTBRDomain” guarantee term. The user should then grasp that the MTBR value definition is inconsistent with the MTBR domain.

## 5 A Proof-of-Concept

We have developed a proof-of-concept with the Choco constraint solver [1] and the Palm explanation engine [3]. This proof-of-concept receives two XML documents as input: the SLA WS-Agreement\* document and the metric XML document. After processing the inputs, our consistency checker returns whether the document is consistent or not and its explanation in the latter case. We have excluded the syntax consistency checking for simplicity.

The current Palm library included in Choco v.1.2.03 only gives complete support to integer variables and constraints with logical operators like  $\geq, \leq, =, \neq, \dots$  Boolean operators and implications like *if constraint1 then constraint2* and assignments like  $\text{var1} = \text{var2} \times \text{var3}$  are excluded. Thus, to test our proof-of-concept, we have simplified our previous SLA example as follows:

- Variables remain equal, unlike in the previous example, because they are all integer variables. But we choose only three of them: availability, cost and increment, and we have added a new initial cost. The unbounded integer domains must be bounded and we assign a maximum value of 10.000 by default as Figure 5 depicts.

```

<metricXML>
  <Availability type='integer' min='1' max='100' />
  <Cost type='integer' min='1' max='10000' />
  <Increment type='integer' min='1' max='10000' />
  <InitCost type='integer' min='1' max='10000' />
</metricXML>

```

Fig. 5. Ad-hoc XML document for the variable domains of Figure 6

- We have removed the qualifying conditions from the guarantee terms because implications are not supported currently. As a consequence, the term compositor elements `OneOrMore` and `ExactlyOne` do not make sense in the simplified example, so an unique `All` term compositor element is included in the new SLA. Finally we have included an inconsistency in the definition of the value of the `InitCost` property. The new SLA is shown in Figure 6.

After mapping the example of Figure 6 to the equivalent CSP  $\psi_\alpha$ , our proof-of-concept processes the explanation problem and returns a minimal subset of inconsistent constraints of the  $\psi_\alpha$ . Then, it traces these constraints to the inconsistent agreement terms. For this example, the proof-of-concept consistency checker returns the following subset  $[CostLET15, CostDefinition, SDTInitCost]$ , because the `InitCost` definition inside the `ServiceDescriptionTerm` is inconsistent according to the `CostDefinition` and `CostLET15` terms. If the user solves the inconsistency of these terms, the checker would check again whether the new agreement document is consistent or not and it would return the minimal subset of inconsistent constraints in the second case.

## 6 Related Work

As far as we know, there are no proposals that deal with providing explanations for the SLA inconsistencies of WS-Agreement documents. Previously, in [6], we proposed mapping SLAs to CSPs, aimed at checking their consistency and conformance. However, in that paper no explanation about the inconsistency of the terms was provided. In addition, [6] dealt with its own SLA specification instead of using a standard format such as WS-Agreement.

Other similar work is [5], in which Oldham et al. create a description logic-based ontology of WS-Agreement that could be used to check consistency and conformance of SLAs using a description logic reasoner. However, the authors do not detail what the consistency checking process is. Furthermore, they do not support the explanations for the inconsistent terms.

```

<Agreement 'id=simplifiedExampleScenario'>
  <Context> ... </...>
  <All>
    <ServiceDescriptionTerm Name='ComputingService'>
      <...>
        ...
        <InitCost> 20 </...>
        <Availability> 95 </...>
        <Increment> 15 </...>
        ...
      </...>
    </ServiceDescriptionTerm>
    <ServiceProperties>
      ...
      <Variable name='Availability' metric='metricXML:Availability'>
        <Location> \\Availability </Location>
      </Variable>
      <Variable name='Cost' metric='metricXML:Cost'>
        <Location> \\Cost </Location>
      </Variable>
      <Variable name='Increment' metric='metricXML:Increment'>
        <Location> \\Increment </Location>
      </Variable>
      <Variable name='InitCost' metric='metricXML:InitCost'>
        <Location> \\InitCost </Location>
      </Variable>
      ...
    </ServiceProperties>
    <GuaranteeTerm Name='CostDefinition'>
      <SLO> Cost = InitCost + Increment </...>
    </GuaranteeTerm>
    <GuaranteeTerm Name='InitCostLET15'>
      <SLO> InitCost <= 15 </...>
    </GuaranteeTerm>
    <GuaranteeTerm Name='CostGET20'>
      <SLO> Cost >= 20 </...>
    </GuaranteeTerm>
    <GuaranteeTerm Name='CostLET30'>
      <SLO> Cost <= 30 </...>
    </GuaranteeTerm>
  </All>
</Agreement>

```

Fig. 6. Simplified example of a WS-Agreement\* document for the proof-of-concept

## 7 Conclusions and Future Work

In this paper we have motivated the need for explaining inconsistencies of WS-Agreement documents and we have presented a first approach to reach this goal. More specifically, we present the problem of explaining WS-Agreement inconsistencies as a constraint satisfaction problem (CSP), and then we use a CSP solver together with an explanation engine to check the consistency and return the inconsistent terms.

To this end, we have defined WS-Agreement\*, which is a subset of WS-Agreement that limits the expressiveness of WS-Agreement, but still allows defining complex SLAs such as the one depicted in Figure 11. Then, we have detailed the mapping of WS-Agreement\* to a CSP and we have described the steps that involve the process of checking and explaining WS-Agreement inconsistencies. Finally, we have presented a proof-of-concept implementation that uses the Choco solver and the Palm explanation engine to perform the explanation of SLA inconsistencies on a simple example.

In summary, the main contributions of this paper are the following:

1. we define a subset of WS-Agreement that can be useful for implementations that do not require the full expressiveness of WS-Agreement;
2. we define a mapping of WS-Agreement\* to CSPs that enables the use of CSP solvers to check the consistency of SLAs;
3. we describe a CSP solver-independent process to check and explain inconsistencies of SLAs.

However, there are still some open issues that require further research: first, extending the mapping to CSPs to full WS-Agreement; second, checking the consistency of WS-Agreement with the temporal extension we detailed in [4], and third, using the CSP solver to check not only the consistency, but also the conformance of an agreement offer with an agreement template.

## References

1. Choco constraint solver web site (2008), <http://choco-solver.net/>
2. OGF Grid Resource Allocation Agreement Protocol WG (GRAAP-WG). Web Services Agreement Specification (WS-Agreement), v. gfd.107 (2007)
3. Jussien, N., Barichard, V.: The PaLM system: explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000, Singapore, September 2000, pp. 118–133 (2000)
4. Müller, C., Martín-Díaz, O., Ruiz-Cortés, A., Resinas, M., Fernández, P.: Improving Temporal-Awareness of WS-Agreement. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSSOC 2007. LNCS, vol. 4749, pp. 193–206. Springer, Heidelberg (2007)
5. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic WS-Agreement Partner Selection. In: 15<sup>th</sup> International WWW Conf., pp. 697–706. ACM Press, New York (2006)
6. Ruiz-Cortés, A., Martín-Díaz, O., Durán, A., Toro, M.: Improving the Automatic Procurement of Web Services using Constraint Programming. *Int. Journal on Co-operative Information Systems* 14(4) (2005)
7. Schiex, T., Verfaillie, G.: Nogood recording for static and dynamic constraint satisfaction problems. In: Tools with Artificial Intelligence, TAI 1993. Proceedings, Fifth International Conference, November 8–11, 1993, pp. 48–55 (1993)
8. Tsang, E.: *Foundations of Constraint Satisfaction*. Academic Press, London (1995)

# Ontology-Based Compatibility Checking for Web Service Configuration Management

Qianhui Liang<sup>1</sup> and Michael N. Huhns<sup>2</sup>

<sup>1</sup>School of Information Systems,  
Singapore Management University, Singapore  
<sup>2</sup>Department of Computer Science and Engineering,  
University of South Carolina, USA  
althealiang@smu.edu.sg, huhns@sc.edu

**Abstract.** Service-oriented systems are constructed using Web services as first-class programmable units and subsystems and there have been many successful applications of such systems. However, there is a major unresolved problem with the software development and subsequent management of these applications and systems. Web service interfaces and implementations may be developed and changed autonomously, which makes traditional configuration management practices inadequate for Web services. Checking the compatibility of these programmable units turns out to be a difficult task. In this paper, we present a technique for checking compatibility of Web service interfaces and implementations based on categorizing domain ontology instances of service description documents. This technique is capable of both assessing the compatibility and identifying incompatibility factors of service interfaces and implementations. The design details of a system model for Web service compatibility checking and the key operator for evaluating compatibility within the model are discussed. We present simulation experiments and analyze the results to show the effectiveness and performance variations of our technique with different data source patterns.

**Keywords:** Web services compatibility, ontology, Web services configuration management.

## 1 Introduction

As service engineering takes hold in large-scale intra- and inter-enterprise service-oriented computing (SOC) settings, how to manage service configurations (or differences) becomes a key enabling task for correct and effective use of dynamic services. Service components, e.g., Web services, resemble traditional independent software systems in being autonomous. However, they have an important advantage over traditional systems: they are network discoverable and accessible via standardized protocols. Comparing service compatibility is the basis of



configuration management for Web services, as it is the key to successful engineering of the third generation of Web services, such as (1) forming a consensus about each other's behavioral differences and similarities, and (2) robust service composition by self-organizing similar services into teams [1].

Configuration management practices in traditional software engineering [2,3,5] [13,14,15] are insufficient for Web services. One major reason is that the autonomous development and deployment of services does not constrain the service engineering and management activities within a tightly-coupled and closely administered environment as is typically expected for traditional local components or controlled distributed components. For example, in a service-oriented environment, two very different service providers may independently evolve the same service interface. They may also develop and change service implementations according to the evolved version of the interface. Given the fact that changes to Web services are not performed by people that are bound by the same organizational constraints, it is important to discover and maintain a consistent mechanism to track the compatibility or incompatibility of the services. Such a mechanism will have a major impact on the reuse of services and on maintaining a consistent understanding of the evolutions made by different parties in a holistic way.

In this paper, we first present a conceptual system model for service objects to facilitate compatibility checking in a service-oriented environment. We then present a technique that conforms to this model to be used for the compatibility checking of Web service interfaces and implementations, which are possibly developed by different teams or completely independent providers. The technique enables consistent Web service configuration management in an enterprise or open service-oriented environment. This technique can be used to (1) retrieve Web service interfaces or implementations that are compatible with a given service object, and (2) identify distinct incompatibility factors of service interfaces and implementations. The technique is based on a domain ontology instance categorization tool, which categorizes terms from service description documents. The technique defines several semantic rules, which are designed to be applied to the term categorization results for assessing compatibility and identifying incompatibility factors of service objects. We present the experiments and results, and demonstrate the effectiveness of our approach on checking compatibilities of services.

The remainder of the paper is organized as follows: Section 2 is related work. Section 3 describes the idea of compatibility checking of Web services and a conceptual system model for organizing service objects and checking compatibility of service objects within this model. Section 4 discusses the technical details of an ontology instance categorization tool, presents the scheme of using ontology categorization for service compatibility checking, and further elaborates how to use the semantic results to distinguish various incompatibility factors. Section 5 illustrates the experiments and provides an analysis of the experimental results, while Section 6 concludes the paper.

## 2 Related Work

Automatic service discovery and selection is a key aspect for composing Web services dynamically in SOC. Current approaches to automating discovery and selection make use of only structural and functional aspects of the Web services. We believe that behavioral selection of Web services should be used to provide more precise results. Service behavior is difficult to specify prior to service execution and instead is better described based on experience with the service execution. In earlier work, we presented an approach to service selection and maintenance-inspired by agile software development techniques-that is based on behavioral queries specified as test cases. Behavior is evaluated through the analysis of execution values of functional and non-functional parameters. The tests can also be used to assess performance and reliability. Therefore, in addition to behavioral selection, our framework allowed for real-time evaluation of non-functional quality-of-service parameters, scalability, and dynamism [4]. Our work reported herein focuses on the compatibility issues of service evolution with multiple providers.

Ontology studies for semantic Web services can be categorized as ontology matching, ontology mapping, and ontology merging. Techniques and algorithms have been proposed to match concepts among heterogeneous ontologies in a tree-like structure or a graph-like structure [9,10,11]. A majority of the matching techniques proposed are schema-based and discover similarities from linguistic or (and) structural characterizations. Platforms and frameworks that provide an integrated environment to facilitate easy use of these matching techniques are also reported. Our compatibility checking system relies on a probabilistic ontology categorization technique. Similar to GLUE [12], the categorization technique is also instance based. In contrast to GLUE, the aim of our ontology categorization technique is to provide a strong evidence for Web service compatibility instead of producing a merged ontology.

As the adoption rate of service-oriented computing and Web services continues to grow, topics pertinent to service engineering are attracting increasing interest. The materialized form of services is software products, which are required to go through a number of general engineering activities. The bottom line of service engineering, therefore, is a process similar to the Software Development Life Cycle (SDLC), which consists of design, build, test, and maintainance. One important aspect of the process is how to record the changes so that consistency and compatibility is guaranteed for interactions between service users and providers. A few industry papers present best practices in Web service versioning [6,7]. Frank et al. [8] also present a proxy-based Web service hosting environment with routing points to handle requests of different versions. They have separate versioning for service interfaces and service implementations. The aim of our work on compatibility checking is to address the most basic issue of Web service versioning and to lay the foundation for research on Web service-specific versioning systems.

All the research issues studied in the above related work can be attributed to one fact, i.e., unlike traditional software, services are published and shared across the Web. As a result, the changes and evolution of service interfaces and implementations are not controlled by a single authority. As more services that serve the same purpose become available, improved versions of the services will be an increasingly complex problem to solve. For example, two important and open issues to be addressed for Web service versioning is how compatibility of services can be determined in the presence of heterogeneous descriptions and how versions of services can be constructed based on compatibility.

### 3 Web Service Compatibility and Versioning

In this section, we introduce our compatibility model of Web service interfaces and implementations.

#### 3.1 Web Service Compatibility Model

To study the compatibility of service interfaces and service implementations, we propose a schema that models the following two categories of information, each of which is represented as a class in the model:

- The classes of objects useful for compatibility checking
- The families that collect related objects.

Object classes include a selection of elements defined in WSDL and UDDI, i.e., `businessService`, `operation`, `tModel`, and `categoryBag`. The classes are defined in the schema in Listing 1. Objects of these classes are referred to as service objects. The first three classes all derive from a class of `Versioned` objects. We treat compatibility of service interfaces and service implementations as two separate but related issues. We also consider versioning of service operations. Therefore, we use the above service objects in our model to differentiate levels of compatibility checking that may contribute to Web service configuration management.

Attributes of a class can be single-valued, such as `release` and `time` of the `Versioned` class. Attributes can also be set-valued, such as `iTypes` of `TModel` class. Compatible objects are linked together. The previous object and the next object in the same link are pointed to by attributes of `nextCompa` and `prevCompa` of a particular object. The number of trailing objects is recorded in `tail`.

The class of `CheckingSystem` represents the entire repository established for the compatibility checking purpose, which contains all objects of all classes. The attribute of `portTypeRoots`, `tModelRoots`, `businessServiceRoots` are the roots of the linked compatible objects. It also has three `Check` constraints in `CheckingSystem`. The first `Check` constraint specifies that each `TModel` object must be categorized with one `CategoryBag` object. The second `Check` constraint specifies that each `TModel` must have at least one `PortType` object defined in it. The third `Check` constraint specifies that each object of `BusinessService` object must implement at least one `TModel` object.

*Listing 1*

```

{
Versioned: class (
  release: Integer,
  build: Integer,
  time: TimeStamp)
NameOwner: class (
  name: String,
  owner: String)
Operation:
  subclass of Versioned, NameOwner (
    operation: XML,
    nextCompa: InterfaceOperationType, //defined in WSDL20
    prevCompa: InterfaceOperationType,
    tail: int)
TModel:
  subclass of Versioned, NameOwner (
    tModel: XML,
    cBags: set of CategoryBag,
    iTypes: set of InterfaceType, //defined in WSDL20
    nextCompa: TModel, //defined in UDDI3
    prevCompa: TModel,
    tail: int)
CategoryBag:
  subclass of NameOwner (
    categoryBag: XML,
    tModel: TModel)
BusinessService:
  Subclass of Versioned, NameOwner (
    bindingTModels: set of TModel,
    nextCompa: BusinessService, //defined in UDDI3
    prevCompa: BusinessService,
    tail: int)
CheckingSystem: class (
  operations: set of Operation,
  tModels: set of TModel,
  businessServices: set of BusinessService
  categoryBags: set of CategoryBag
  check
(for-all tm in tModels)
(there-exists b in categoryBags)
  check
(for-all tm in tModels)
(there-exists i in iTypes)
  check
(for-all bs in businessServices)
(there-exists tm in tModels))
}

```

### 3.2 System Model

We now describe the structure of our compatibility-checking system. Listing 2 defines the system model of the compatibility checking system. `checkingSystem` is an instance of class `CheckingSystem`. It contains version families composed of objects of class `Operation`, `TModel`, and `BusinessService`. We have used the concept of version family when designing our compatibility-checking system. Our purpose is to categorize the object space into a number of compatible groups. A version family is a group of objects of the same class selected based on certain properties [18]. These compatible objects (belonging to the version) are linked together.

The system must be capable of refining the compatibility classification by selection of version families. We use version selection rules for this purpose. These rules operate on version families and select one or more members from the families. In particular, the selection rules of compatibility checking return a set selected from the compatible objects. These objects may or may not belong to the same family, depending on the requirement of the checking task. Several examples are given in Listing 2.

The `select` operation in Listing 2 outputs the latest `tModel` object in a `tModel` version family. `selectCompatibles` operation outputs all `tModel` objects in a version family that are compatible with a given `tModel` object. The `incompatiblewith` operator outputs 0 if the first operand is compatible with the second operand. Otherwise, it outputs some positive number that identifies incompatibility causes if the first operand is incompatible with the second. The technique to implement the `incompatiblewith` operator is discussed in Section 4. `selectCompatibleRoot` finds the `tModel` root that has the largest number of objects in its tail.

#### *Listing 2*

```
checkingSystem: system model from CheckingSystem (
  operations = {"Root1_operations", "Root2_operations", }
  tModels = { "Root1_tModels", "Root2_tModels", }
  businessServices = {"Root1_businessServices", "Root2_businessServices", }
  categoryBags = {"Finance", })
select tModel from f: family of TModel
in checkingSystem.tModels (
  (for-all mf in f)
  (tModel.release>=mf.release))
selectCompatibles set of tModel from f: family of TModel given tm class of TModel
in checkingSystem.tModels (
  (for-all tModel)
  (incompatiblewith(tModel,tm)=0))
selectCompatibleRoot tModel from selectedRoots: family of TModel
in checkingSystem.tModels (
  (for-all root in selectedRoots)
  (tModel.tail >= root.tail))
```

## 4 Compatibility Checking by Ontology Categorization

In this section, we describe how the operator of `incompatiblewith` used in `selectCompatibles` in Listing 2 is realized by (1) using an ontology categorization technique to categorize important terms in the descriptions of Web services, and (2) assessing service compatibility according to semantic rules and the categorization results of terms. In Section 4.1, we first review the ontology categorization technique that has been designed in our earlier work [17].

### 4.1 Basics of Ontology Categorization

Web service information resources are documents that contain descriptive information about the semantics of Web services. We focus on semi-structured documents, including WSDL and OWL-S documents, as sources for checking their compatibility. We are interested in the portion of service descriptions that carries semantics related to the domain and will be useful in checking service compatibility. In WSDL documents, we use the names of *service interfaces*, *operations*, *input*, *output*, and *elements* of input and output, and about implementation descriptions in terms of *binding* protocols. In OWL-S documents, we have used *ServiceProfile* and *ServiceGrounding*, which we consider most relevant to domain knowledge.

Terms are first extracted from the service descriptions. Next, pre-processing is performed on the extracted terms. After that, the terms are used to establish a probabilistic model for ontology instance categorization. The categorization model is adapted from existing classification techniques found in the information retrieval area. It consists of two parts: (1) SVMV-based and (2) co-occurrence-based. Both parts of the model perform categorization based on the similarity ranking of terms in an independent manner. Their results are then merged using a simple voting scheme. The model is used to decode heterogeneous descriptions continuously and enhance the categorization basis by learning new service descriptions.

We have adapted SVMV, as a model of probabilistic text categorization, to derive the probability that a description  $d$  is categorized into a category  $c$ , i.e.,  $p(c|d)$ . We adapt SVMV by introducing relationship patterns that pertain to service descriptions. For example, we consider the relationship between `InterfaceOperationType` and `input`, `InterfaceOperationType` and `output`, and `InterfaceType` and `Operation`.

In co-occurrence analysis, a description document is represented by a matrix of co-occurrence frequencies. We establish asymmetric links for each pair of terms. A cluster function defines the term similarity weights by the combined weight of both term  $t^j$  and  $t^k$  in document  $i$  and the inverse document frequency. To perform co-occurrence analysis, the asymmetric co-occurrence analysis function [17] is adapted for service descriptions. Our adaptation mainly concerns the definition of similarity weights and the calculation of the weighting factor.

## 4.2 Compatibility Checking

The *incompatiblewith* operator is evaluated by our system in the following way: categorize the values of a number of selected key tags in the corresponding XML constructs within the service descriptions and analyze the categorization results according to semantic rules. The key tags are selected from the conceptual definitions of services in a given upper service ontology, e.g., OWL-S. Table 1 lists all the objects and their key tags. Please notice that this forms a hierarchy of relevant tags for **Operation**, **TModel**, and **BusinessService**, which are consistent with the hierarchy of the checking system model described in Section 3.

For **Operation** objects, **operation names**, **input element names**, and **output element names** are the considered tags. For **TModel** objects, in addition to the name itself, all operations referred to by the definition of this **TModel** shall also be considered. For **BusinessService** objects, in addition to the name of this **BusinessService**, all **TModel** objects that are referred to by this **BusinessService** object are also included. We can see the tags are identified in a recursive way.

**Table 1.** Objects and their Key Tags

<i>ObjectClass</i>	<i>KeyTags</i>
Operation	(Operation) name Input element name(s) and types(s) Output element name(s) and types(s)
TModel	(TModel) name Operations
BusinessService	(BusinessService) name TModels

We use the following two sets of recursive rules to check compatibility of two objects of the same class using the ontology categorization tool summarized in Section 4.1. A rule engine based on the semantic service description language SWRL is used for implementing and executing the rules.

**Rule set 1**, which is used to deal with primary objects = {

*Rule 1:* For a particular tag that is simple and single valued, (primary) objects A and B that directly contain this tag are compatible bi-directionally regarding this tag if and only if their corresponding values of that tag are categorized together.

*Rule 2:* For a particular tag that is simple and multi-valued, (primary) object B is compatible to (primary) object A regarding this tag if and only if object B's value set is a subset of object A's value set, where both A and B directly contain this tag.

*Rule 3:* *incompatiblewith* on primary object B and primary object A is evaluated to be 0 if and only if regarding all selected tags object B is compatible with service A. }

**Rule set 2**, which is used to deal with complex objects = {

*Rule 4*: For a particular tag that is of simple type and single valued, (complex) objects A and B that directly contain this tag are compatible bi-directionally regarding this tag if and only if their corresponding values of that tag are categorized together.

*Rule 5*: For a particular tag that is of simple type and multi-valued, (complex) object B is compatible to complex object A regarding this tag if and only if object B's value set is a subset of object A's value set, where both A and B directly contain this tag.

*Rule 6: incompatiblewith* on complex object B and complex object A is evaluated to be 0 if and only if (1) for each selected containing simple tag, object B is compatible to object A regarding the tag, and (2) for each containing complex tags that are present in both objects, *incompatiblewith* on object B and object A is evaluated to be 0 and (3) for all selected containing complex tags, the constituent object set of object B is a superset of the constituent object set of object A.}

We refer to *primary objects* as objects defined in the checking system model that do not contain other objects defined in the model. *Complex objects* are objects defined in the checking system model that contain other objects defined in the model. Referring to Table 1, objects of **Operation** are primary objects and those of **TModel** and **BusinessService** are complex objects. In set 1, rule 1 is used to check the compatibility of each pair of simple single-valued attributes that belongs to the two primary objects to be compared. Rule 2 is used to check the compatibility of each pair of simple multivalued attributes that belong to two primary objects to be compared. Rule 3 is used to determine if one primary object is compatible with another primary object based on the result of rule 1 and rule 2.

In set 2, rule 4 is used to check the compatibility of each pair of simple single-valued attributes that belongs to the two complex objects to be compared. Rule 5 is used to check the compatibility of each pair of simple multi-valued attributes that belongs to the two complex objects to be compared. Rule 6 is used to determine if one complex object is compatible with another complex object based on the result of rule 4 and rule 5. For complex object a to be compatible with complex object b, all simple attributes of a must be compatible with their counterparts in b and all its complex attributes must be compatible with their counterpart in b or such complex attributes in a are not in existence in b. The relationship **compatible with** is unidirectional. To give an example, let us assume there are s1 and s2 both of **BusinessService**, and o1 and o2 both of **Operation** in the checking system. If o1 constitutes s1 and o1 and o2 constitute s2, s2 is compatible with s1. But s1 is not compatible with s2.

### 4.3 Incompatibility Factors

Our compatibility-checking system is designed to not only assess the compatibility of the objects, but also output the incompatibility factor(s) if needed. In Table 2, we summarize independent factors that may affect the compatibility of



**Table 2.** Independent Factors that May Affect the Compatibility of Web Services

<i>ObjectClass</i>	<i>ID</i>	<i>Factor</i>	<i>Compatibility</i>
Operation	CN	Change (Operation) Name	P
	CEN	Change Input and Output Element Name(s)	P
	CET	Change Input and Output Element Types(s)	NP
	RAE	Remove/Add Input and Output Element(s)	NP
TModel	CTN	Change of (TModel) Name	P
	AO	Add new Operation	P
	RO	Remove Operation	NP
	CO	Compatibility of Operation(s) referred by this object	
BusinessService	CBN	Change of (BusinessService) Name	P
	CE	Change of (BusinessService) Endpoint	NP
	CT	Compatibility of TModel(s) referred by this object	

Web services. These factors are referred to as *individual factor*. We have relaxed the definitions of compatibility and incompatibility from traditional software engineering, because services are no longer developed by a single development team as proprietary software assets. P in the table represents compatible and NP represents incompatible. If the compatibility cannot be determined, but is dependent on the compatibility of the factor itself, the cell is left blank.

We have designed the following rule set, i.e., Rule set 3, which is used to judge the factor that has caused incompatibility of two service objects, referred to as *incompatibility factor*.

**Rule set 3** = {

*Rule 7:* For a particular tag that is of simple type and single valued, objects A and B that directly contain this tag are incompatible bi-directionally and this tag is output as a causal incompatibility factor, if and only if their corresponding values of that tag are not categorized together.

*Rule 8:* For a particular tag that is of simple type and multi-valued, object B is incompatible to object A and this tag is output as the causal incompatibility factor if and only if object B's value set is a superset of object A's value set, where both A and B directly contain this tag.

*Rule 9:* Complex object B is incompatible to complex object A if and only if for any containing complex tags that are present in both objects, the constituent object of object B is incompatible to object A and the causal incompatibility factor of the constituent objects will be output as one casual incompatibility factor for objects A and B.}

In set 3, rules 7 and 8 are used to identify incompatibility factors contributed by the simple attributes directly contained in the service objects. In particular, if the service objects are not categorized together on a tag, this tag is one casual incompatibility factor of the two objects. Rule 9 is used to identify incompatibility factors contributed by their containing service objects. In other words, incompatibility factors will propagate from one class of objects to another, as far as the latter has composition or aggregation relationships to the former. Let

us assume again there are  $s1$  and  $s2$  both of **BusinessService**, and  $o1$  and  $o2$  both of **Operation** in the checking system.  $o1$  and  $o2$  are constituent objects of  $s1$  and  $s2$ , respectively. If  $f1$  is a causal incompatibility factor between  $o1$  and  $o2$ , it is also a causal incompatibility factor between  $s1$  and  $s2$ .

## 5 Experiments and Analysis

We have considered in our experiments all the individual factors listed in Table 2 and the factors that are composed of possible combinations of multiple individual factors, referred to as *joint factors*. In our experiments, 600 Web services have been obtained from the following resources. We chose these resources because they are available publicly and are representative of different service domains. From these original Web services, we have formed a collection of 3200 Web services. These services are all mutations of the original services.

The Web service directories and indices are listed below:

- [www.xmethods.com](http://www.xmethods.com),
- [www.bindingpoint.com](http://www.bindingpoint.com),
- [www.webservicelist.com](http://www.webservicelist.com),
- [www.servicesweb.org/rubrique.en.php3?id\\_rubrique=14](http://www.servicesweb.org/rubrique.en.php3?id_rubrique=14)
- Web sites of four companies that publish their Web services: eBay, Amazon, PayPal, and <http://www.xignite.com/>

For all experiments, Web services are selected to form sets of various sizes. For each set, we have selected services in such a way that among all pairs within a set (1) for 20% of the pairs, the first service is compatible with the second service, and (2) for the remaining 80% of the pairs, the first service is incompatible with the second service. We imagine as more and more service providers contribute to the choices for services, which results in an increase of competition of the service market, the ratio of compatible services to incompatible services will also increase. Therefore, we not only test the ratio of 20%-80%, we also test 30%-70%, and 60%-40%. Distributions of both compatible and incompatible objects over all possible factors including individual and joint factors generally follow a uniform distribution. The detailed distributions of service compositions over incompatibility factors are listed in Table 3. Due to space constraints, we only show a part of our experiments and results.

We refer to the sets of service objects for testing the performance of compatibility checking in our experiments as a service set. In our experiments, we always create 10 different service sets so that the same experiment can be repeated 10 times. The result shown is the average performance over all service sets. As part of the preparation of the experiments, we partition a service set into a *base set* and *comparison set*, which comprise 30% and 70% of all the objects in the service set, respectively. A compatibility checking task is defined as the following: for a service picked from the *base set*, retrieve all services in the *comparison set* that are compatible with it and output the incompatibility factors of incompatible services.

**Table 3.** Distributions of Service Compositions over Incompatibility Factors

<i>ObjectClass</i>	<i>IndividualFactorID</i>	<i>JointFactorID</i>	<i>Per -cen -tile (%)</i>						
			A	B	C	D	E	F	G
<i>Operation</i>	<i>CN(notCEN)</i>		7	0	20	5	10	7	10
	<i>CEN(notCN)</i>		7	30	20	5	10	7	10
		<i>CN - CEN</i>	6	0	20	10	0	6	0
		<i>CET</i>	30	23	13	40	40	20	40
		<i>RAE</i>	30	24	13	40	40	20	40
			<i>CET - RAE</i>	20	23	14	0	0	40
<i>TModel</i>	<i>CTN</i>		5	0	15	3	7	5	5
	<i>AO</i>		5	20	15	4	8	5	5
		<i>CTN - AO</i>	5	0	15	8	0	5	5
		<i>RO</i>	40	30	20	60	60	20	60
			<i>CO - P</i>	5	10	20	5	5	5
			<i>CO - NP</i>	40	40	15	20	20	60
<i>BusinessService</i>	<i>CBN</i>		10	15	30	5	15	10	10
	<i>CE</i>		40	35	20	60	60	20	60
		<i>CT - P</i>	10	15	30	15	5	10	10
		<i>CT - NP</i>	40	35	20	20	20	60	20

We perform experiments to measure the performance of the compatibility checking against different compositions of compatible and incompatible factors in service sets of 2000 service objects. We performed 600 (i.e., 30% of 2000) tasks per service set and there are all together 6000 tasks across all 10 service sets. In particular, we study how our checking system performs against different ratios of individual compatibility and joint compatibility factors. We draw *precision* and *recall* in percentile for compatible service retrieving, where Precision is represented as number of true compatible assessments/(number of true compatible assessments + number of false compatible assessments) and Recall is defined as the number of true compatible assessments+/(number of true compatible assessments + number of false incompatible assessments). The result is shown in figure 1. In the left part of figure 1, three lines give the performance over the compositions for column of Percentile A, B, and C in Table 3. As the compatibility ratio increases, the line moves right and up depicting an improved effectiveness of the system (precision of 13% for A compared with precision of 18% for C for recall of 100%). Although this result is straightforward, we use it to show that the system shall show more promising results with the further adaptation of SOA and component based application building. In particular, we observe that our technique favours data sources with a relatively large portion of compatible service objects and that are less skewed. This makes it suitable for service compatibility checking tasks, because service data objects are very likely to follow such a pattern.

In the right part of figure 1, three lines represent the performance over the compositions listed in A, D, and E of Percentile column. We are using this experiment to show that as the incompatibility and compatibility factors move

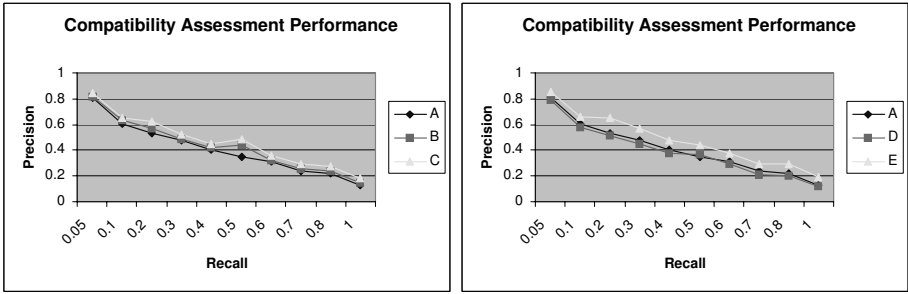
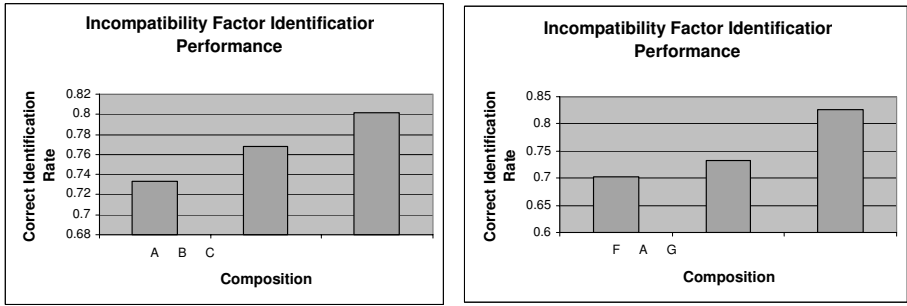


Fig. 1. Compatibility assessment performance for A, B, C, and A, D, E in table 3

from single to joint, or from simple to complex, the performance of the checking system degrades. For example, for recall rate of 100%, precision for E, A, and D are 19.6%, 13%, and 12.7%. As we can see, among E, A, and D, E has the highest percentage of individual compatibility factors and lowest (zero) percentage of joint compatibility factors. D has the highest percentage of joint compatibility factors. Joint compatibility factors on one hand shall increase the possibility that services are assessed as false negative due to the compatibility assessment mechanism of our design, and therefore, result in loss of the recall. On the other hand, it decreases the possibility that services are assessed as false positive and, therefore, results in improved precision. Service users are probably more concerned with precision. We can conclude that with higher percentage of individual factors, our technique tends to provide results that are more likely to satisfy service users in compatibility checking.

We also give the average correct identification rate for incompatibility factor identification in figure 2. The average correct identification rate is defined as (identification tasks percentage of identified factors)/number of identification tasks. The three bars in the left part of figure 2 represent the identification performance over the compositions listed in A, B, and C of Percentile column of in Table 3. As the data sources become less skewed, i.e., the ratio of compatible and incompatible objects increases, the categorization technique performs better; therefore, the correct identification rate improves as well (from 73.3% to 80.1%). In the right part of figure 2, the bars represent the identification performance over the compositions listed in F, A, and G of Percentile column of in Table 3. As the ratio of objects with joint incompatibility factors decreases, the complexity of identification of incompatibility is reduced because fewer tags are required to be categorized exactly. In this case, some identification results that used to earn only partial credits due to the wrong categorization of one joint factor can now earn full credits. Therefore, the system shows a better identification rate. In the figure, the rate increases from 70.2% to 82.6%. This result provides a useful implication to companies practicing service evolution: For the purpose of better service compatibility checking, incompatible changes are recommended to be introduced one-by-one. It is not recommended to wait a long while and then to introduce them in a batch process.



**Fig. 2.** Incompatibility factor identification performance for A, B, C and A, F, G in Table 3

## 6 Conclusions

In this paper, we describe an approach to checking Web service compatibility for Web service configuration management based on categorizing domain ontology instances extracted from service descriptions. The design details of a system model for Web service compatibility checking and the key operator for evaluating compatibility within the model are discussed. The contribution of the paper is to consider the engineering aspect of Web services that are due to autonomous and distributed design and development of the services, i.e., to meet the challenge within a service configuration system framework of engineering service compatibilities that may result from changes or parallel and independent service creation by other parties.

We are in the process of improving the basic categorization tool. Interesting issues include how to make it even more effective for different types of service descriptions, how to transfer such performance improvement to the compatibility checking system, and how service users' preferences can be incorporated into the categorization tool and the compatibility checking system.

## References

1. Huhns, M.N.: A Research Agenda for Agent-Based Service-Oriented Architectures. In: Klusch, M., Rovatsos, M., Payne, T.R. (eds.) CIA 2006. LNCS, vol. 4149, pp. 8–22. Springer, Heidelberg (2006)
2. Roekind, M.J.: The source code control system. *IEEE Trans. on Software Engineering* 1(4), 364–370 (1975)
3. Kramer, J., Magee, J.: The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Trans. on Software Engineering* 16(11) (1990)
4. Gutierrez, R., Mendoza, B., Huhns, M.N.: Behavioral Queries for Service Selection: An Agile Approach to SOC. In: Proc. IEEE International Conference on Web Services. IEEE Press, Salt Lake City (2007)

5. Schmerl, B.R., Marlin, C.D.: Versioning and consistency for dynamically composed configurations. In: Conradi, R. (ed.) ICSE-WS 1997 and SCM 1997. LNCS, vol. 1235, pp. 49–65. Springer, Heidelberg (1997)
6. Brown, K., Ellis, M.: Best practices for Web service versioning, <http://www.ibm.com/developerworks/webservices/library/ws-version>
7. Anand, S., et al.: Best Practices and Solutions for Managing Versioning of SOA Web Services, <http://webservices.sys-con.com/read/143883.htm>
8. Frank, D., Lam, L., Fong, L., Fang, R., Vignola, C.: An Approach to Hosting Versioned Web Services. In: Proc. IEEE International Conference on Services Computing. IEEE Press, Los Alamitos (2007)
9. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Proc. workshop on Web and Databases. IEEE Press, Los Alamitos (2002)
10. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Proc. of the 27th VLDB Conference. Springer, Heidelberg (2001)
11. Huang, J., Dang, J., Huhns, M.N.: Ontology Reconciliation for Service-Oriented Computing. In: Proc. IEEE International Conference on Services Computing. IEEE Press, Los Alamitos (2006)
12. Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., Halevy, A.: Learning to match ontologies on the Semantic Web. The VLDB Journal 12, 303–319 (2003)
13. Nierstrasz, O., Gibbs, S., Tsichritzis, D.: Component-oriented software development. Communications of the ACM 127 (1992)
14. Yellin, D.M., Strom, R.E.: Protocol Specifications and Component Adaptors. ACM Trans. on Programming Languages and Systems 19(2), 292–333 (1997)
15. Georgiadis, I., Magee, J., Kramer, J.: Self Organising Software Architectures for Distributed Systems. In: Proc. the first workshop on Self-healing systems (2002)
16. Liu, Y.D., Smith, S.F.: A Formal Framework for Component Deployment. In: Proc. the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, pp. 325–344 (2006)
17. Liang, Q., Lam, H.: Web Service Matching By Ontology Instance Categorization. In: Proc. International Conference on Services Computing. IEEE Press, Los Alamitos (2008)
18. Wiebe, D.: Generic Software Configuration Management: Theory and Design. PhD thesis, published as Technical Report 90-07-03. Department of Computer Science, University of Washington (1990)

# SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services

Ignacio Silva-Lepe, Revathi Subramanian, Isabelle Rouvellou,  
Thomas Mikalsen, Judah Diamant, and Arun Iyengar

IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne NY 10532, USA  
{[isilval](mailto:isilval@us.ibm.com),[revathi](mailto:revathi@us.ibm.com),[rouvellou](mailto:rouvellou@us.ibm.com),[tommi](mailto:tommi@us.ibm.com),[djudah](mailto:djudah@us.ibm.com),[aruni](mailto:aruni@us.ibm.com)}@us.ibm.com  
<http://www.research.ibm.com/>

**Abstract.** SOAlive aims at providing a community-centric, hosted environment and, in particular, at simplifying the description and discovery of situational enterprise services via a service catalog. We argue that a service community has an impact not only on users and services, but also on the environment itself. Specifically, our position is that a service catalog adds value to users, and is itself enriched, by its incorporation into a community-centric service hosting environment. In addition, analyses of web services directories suggest that a catalog service for enterprise services can be better provided by using a simpler content model that better fits REST, taking advantage of collaborative practices to annotate catalog entries with informal semantic descriptions via tagging, providing a mechanism for embedding invocations of discovered services, and allowing syntactic descriptions to be refined via usage monitoring. The SOAlive service catalog defines a flexible content model, a discovery function that navigates the cloud of tag annotations associated with services in a Web 2.0 fashion, and a service description refinement function that allows the actual use of a service to refine the service description stored in the catalog.

**Keywords:** Service catalog, situational enterprise service, software as a service, service engineering, service assembly, SOA runtime.

## 1 Introduction

As the field of service-oriented computing evolves, we observe a number of trends. In no particular order, one trend is the exposition of Web Services via REST<sup>1</sup> APIs. Another trend is the development of web applications using dynamic programming languages and frameworks, e.g., JavaScript with AJAX, and PHP. These languages enable rapid development and testing, provide expressive and

---

<sup>1</sup> REST (Representational State Transfer) builds on the HTTP protocol principles to define an architectural style where entities, containers and behaviors can be seen as resources that are accessible via a uniform interface consisting of a fixed set of verbs [5,15].

powerful frameworks, and lead to the use of abstractions that are closer to the problem domain [8]. A third trend is the use of Web 2.0-style social and collaborative filtering practices such as bookmarking and tagging, as found in Web 2.0 services such as `del.icio.us` or `flickr`, where tagging is used to annotate shared content.

Situational enterprise services, as instances of situational applications [2], also result from these trends. An enterprise service (sometimes referred to as a situational enterprise service) is a small, primarily browser-based, situational application, typically exposing a REST interface through which it is invoked, for instance, a travel authorization application. More specifically, an enterprise service does not use a rigorous language, such as WSDL, to define its interface. In addition, it seems natural to take advantage of the sort of informal semantics that are conveyed by tagging, as opposed to using rigorous ontologies, to provide semantic descriptions of enterprise services. It is important to make it easy and painless for developers of enterprise services to: (1) publish and advertise services, (2) discover services, and (3) invoke or compose discovered services.

SOAlive aims at providing a community-centric, hosted environment that supports the development, deployment and management of enterprise services, and that provisions the required deployment and execution middleware. In particular, SOAlive aims at simplifying the description, discovery and composition of situational enterprise services via a service catalog. In [14], service communities are introduced as the combination of social and business communities with the purpose of exchanging services. In particular, service communities establish a dynamic platform where services of interest are contributed, grouped, consumed, and managed [3].

We argue that a service community has an impact not only on users and services, but also on the platform itself. Specifically, our position is that a service catalog adds value to users, and is itself enriched, by its incorporation into a community-centric service hosting environment. That is, users benefit more from a service catalog that is part of a service community than from an isolated service directory or even a limited form of service marketplace. Likewise, a service catalog that is incorporated into a service community environment can take advantage of service deployment and usage to improve on the quantity and quality of the information it provides.

Furthermore, the SOAlive service catalog aims at benefiting from some of the lessons learned from web services directories. In [6], Legner presents an analysis of web services directories that are based on the UDDI standard. Some of her observations and conclusions are: (1) “Despite the fact that private UDDI registries allow for advanced categorization and identification schemes, the investigated Web services intermediaries use rather simplistic search and categorization mechanisms”, (2) “Unlike in other electronic markets, we were not able to observe increasing personalization and customization of the Web services offerings”, (3) “Using the StrikeIron Marketplace API, software vendors are able to embed service invocations in their applications”, and (4) “More sophisticated classification schemes which reflect the vocabulary of the target consumers



are, in combination with complete and reliable service descriptions, a prerequisite for the discovery of suitable Web services”.

All of this suggests that a catalog service for enterprise services can be better provided by: (1) using a simpler content model that better fits REST, instead of imposing a more complex model, such as UDDI, which relies on more complete technical specifications of services, (2) taking advantage of collaborative practices to annotate catalog entries with informal semantic descriptions via tagging, (3) providing a mechanism for embedding invocations of discovered services, and (4) using tagging as above, and allowing minimal syntactic descriptions to be initially entered, which can then be refined via usage monitoring. In this paper, we introduce the SOAlive service catalog, which provides a set of functions that satisfy these requirements.

The SOAlive service catalog defines a flexible content model that (1) requires little or no up-front user input, (2) can evolve over time, either automatically or from user input such as user-provided tags, and (3) facilitates the composition of services by generating snippets of invocation code that can be inserted into services under development that invoke discovered services. This content model is designed to contain both syntactic as well as informal semantic descriptions of enterprise services. The service catalog also provides a discovery function that navigates the cloud of tag annotations associated with services in a Web 2.0 fashion. Finally, the service catalog provides a service description refinement function that allows the actual use of a service to refine the service description stored in the catalog.

The remainder of this paper is organized as follows. Section 2 provides a short overview of the SOAlive hosted environment. Section 3 introduces the SOAlive service catalog and its content model. Section 4 provides more details on service discovery, and Section 5 focuses on the service description refinement function of the catalog. Section 6 describes one possible implementation of the catalog. Section 7 discusses related work. Finally, section 8 concludes the paper.

## 2 SOAlive Overview

SOAlive provides a hosted environment for developing, deploying and executing enterprise services. The diagram in Fig. 1 highlights these aspects with a focus on the interactions with the catalog.

At development time, a user can discover services, which can be composed into other services being developed via the use of code snippets generated by the catalog. Here, because the catalog is community-centric, users benefit from the usage and bookmarking of services by other users in the community to improve their discovery experience.

At deployment time, a user relies on the hosted environment, specifically the application manager, to publish his services to the catalog and to install them on the runtime that is provisioned by the hosted environment as well. Here, because the application manager and the catalog are integrated into the same hosted environment, deployed services are automatically published without any more intervention from the user.

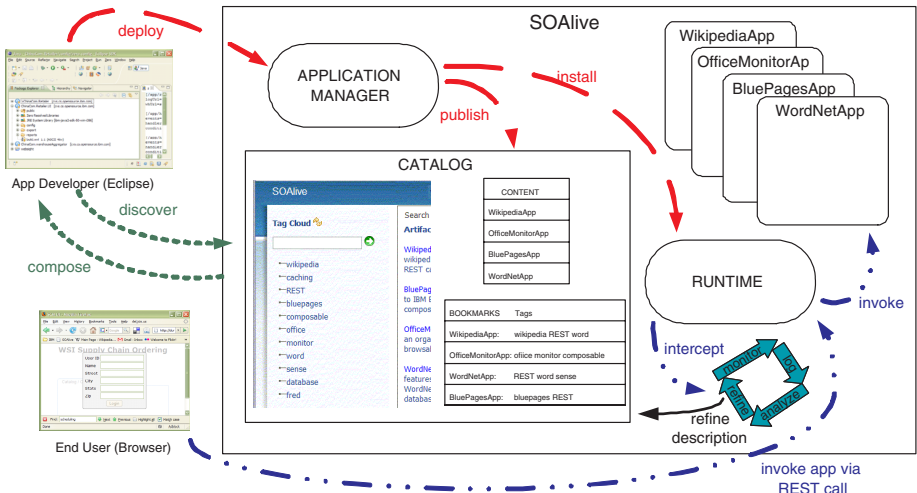


Fig. 1. SOAlive overview

At run time, service invocations, in addition to being routed to the target service, are also intercepted by the runtime. The runtime includes a monitor component that forwards the intercepted invocations to a logging component. Independently, a component that analyzes and processes the logged invocations interacts with the catalog to define and refine service descriptions. Here, because the runtime and the catalog are integrated into the same hosted environment, service invocations can be mined to extract valuable information that can be used in the definition and refinement of the corresponding service’s description.

At any given time, users can also browse the catalog and, in a Web 2.0 fashion, share their discoveries of useful services by adding to the catalog bookmarks that are annotated with tags of their choice. This further improvement on the quantity and quality of the information provided by the catalog is a consequence of its community-centric nature and its integration into a service hosting environment.

### 3 SOAlive Service Catalog

The SOAlive service catalog is a core service in the SOAlive hosted environment. One main function of the catalog is to store descriptions of artifacts it knows about. As we shall see however, the catalog is not just a passive store of descriptive information. As the descriptions of the known artifacts are entered, the catalog can extract additional information from these descriptions in order to improve on them. Also, as the known artifacts are used, the catalog can process logged information about service invocations, to enhance the descriptions of those services.

### 3.1 Catalog Artifacts

As a hosted environment, SOAlive supports not only the deployment, execution and management of services but also their development. As such, the SOAlive catalog is concerned with the description of not only services, but also of other artifacts that are used in the composition and development of services. An artifact that can be described in the SOAlive catalog is either a *service* that is deployed on SOAlive and is being managed by the SOAlive Application Manager, a *module* that is maintained in the SOAlive repository<sup>2</sup>, or an external artifact. Deployed services can in turn be:

- **Callable:** These are services for which the catalog can include documentation which can be used to enable composition, e.g., a credit check service that can be composed into a larger mortgage approval service.
- **Browsable:** These are services that can be discovered in the catalog and invoked by just knowing the endpoint information, e.g., a retailer UI which can be accessed by a simple HTTP GET with the artifact’s URL.

Modules in the SOAlive repository can in turn be:

- **Deployable:** These are self-contained artifacts that can be deployed on their own, such as ProjectZero [11] applications or SCA [13] composites.
- **Composable:** These are artifacts that cannot be deployed in isolation, such as utility classes, ProjectZero application fragments, or SCA component implementations. Composable artifacts are typically composed to form deployable artifacts. As an example, a Bite extension activity [1] is a composable module that must be included as part of a larger flow in order to be deployed.

An external artifact is not hosted on SOAlive, e.g., an enterprise user directory service, but its description can also be in the catalog so that other SOAlive applications can discover and use it. This way, users can also benefit from bookmarking and tagging services that are not hosted by SOAlive. In many cases, SOAlive will host a proxy to an external service. This way, SOAlive can also monitor the use of external services in order to refine their description (see Section 5).

Catalog entries for artifacts are created by system entities such as the SOAlive Application Manager or the SOAlive Repository, or by an admin authority in the case of external artifacts, on behalf of a SOAlive user.

### 3.2 Content Model

The SOAlive catalog maintains two kinds of artifact information: (1) the main description that is initialized at the time the artifact is published, often including information provided by the artifact’s author, and (2) bookmarks (or references to artifacts) on behalf of users other than the artifact’s author.

<sup>2</sup> The SOAlive Repository allows users to store and share application artifacts as modules. Modules are packaged into archive files files (e.g., zips and jars), and the SOAlive Repository provides an interface for uploading and downloading modules.

**Main Description.** The main description of an artifact includes its name, owner identification, version number and visibility rules, all of which the catalog can provide default values for. The visibility rules are used for access control and can default to `public`, meaning any user can view and access the artifact. The description also includes a relative URI that is used as an endpoint reference, either as the URL of a deployed service, or as the URI of the artifact in the SOAlive repository. In addition, the main description of an artifact includes a syntactic as well as an informal semantic description.

*Syntactic Description.* The syntactic description of a deployed or a deployable service is maintained as a stylized RESTful documentation of the interface exposed by the service. This interface enumerates the resources exposed by a service and, for each resource, information about its methods' data format, success and error codes, as well as typical examples of request and response. Representative examples of RESTful documentation that can be used in the syntactic description of a service in SOAlive are Project Zero's RESTdoc [11] and WADL [12]. The syntactic description of a composable artifact depends on the type of the artifact and so the catalog cannot assume any pre-determined schema or grammar. For instance, if the composable artifact represents a Bite extension activity [1], its syntactic description can be given by a piece of textual XML, or by a JSON serialized object containing the attributes that pertain to the extension activity.

*Informal Semantic Description.* To a modern developer it would seem natural to be able to use collaborative filtering practices to publish and discover hosted enterprise services in a hosted environment such as SOAlive. To this end, the SOAlive catalog allows the inclusion of tags as an informal semantic description of a service, or an artifact in general. As in other Web 2.0 services, such as `del.icio.us` and `flickr`, tags used to annotate catalog artifacts induce a tag cloud that can then be used to navigate the space of artifacts.

Unlike other Web 2.0 services however, developers of enterprise services will be interested not only in the services that they have bookmarked but also in any service that is visible to them, regardless of who has published it or bookmarked it. For instance, if I am developing a mortgage application that requires the use of a credit check service, I would like to be able to discover, using a Web 2.0 mechanism such as a tag cloud, any service that may have been published to the hosted environment that may perform a credit check function. Therefore, the tag cloud available to a SOAlive developer should include any tag that annotates any service or artifact in the catalog. And as it is not uncommon for tag clouds in traditional collaborative filtering systems to become very large, it seems reasonable to expect a lower bound on the size of our tag cloud to be in the order of hundreds of tags.

To help in reducing the size of its tag cloud, the SOAlive catalog provides a tag consolidation facility. This facility relies on the ability to collect groups of tags given by some relationship amongst tags and then define the tag cloud as the collection of the representatives for each group of tags. In particular, tag grouping can be given by the hypernym relationship between the meanings or senses of any two tags. A tag group's representative in this case is given by

the tag in the group for which there is no hypernym in the tag cloud. In turn, tag meaning or sense can be obtained from a public lexical database such as *WordNet* [4].

More specifically, when a developer publishes an artifact to the catalog and annotates it with tags, if any such tag  $t_1$  is given a sense  $s$ , then the catalog looks in the tag cloud for a tag  $t_0$  whose sense  $s_0$  is the hypernym of  $s$ . If  $t_0$  is found then  $t_1$  is added to  $t_0$ 's group or hierarchy. If  $t_0$  is not found in the tag cloud but it exists in the lexical database, then both  $t_0$  and  $t_1$  are added to the tag cloud under the same group. We refer to  $t_0$  as a derived tag, given that it was not explicitly entered by a user but rather derived by the tag consolidation facility. In addition, if  $t_0$  did not exist in the tag cloud, and there is a tag  $t_2$  in the tag cloud whose sense is a hyponym of  $s_0$ , then  $t_0$  and  $t_1$  are added under the same group as  $t_2$ . For instance, suppose that a tag *investment* exists in the catalog as an annotation to a service that was published by a user  $u_1$ . When user  $u_2$  publishes a service and annotates it with tag *banking*, the catalog determines that *finance* has a sense that is a hypernym of both *investment* and *banking*, adds *finance*, and groups both *investment* and *banking* under it. Later, when the tag cloud is displayed, it will show *finance* as a top level tag in the tag cloud.

Tag grouping using hierarchies provides a simple way to consolidate the size of the tag cloud, and to generalize the search for an artifact. For instance, after *investment* is grouped under *finance*, a search for artifacts under *finance* will yield all entries that are tagged with any hyponym of *finance*, including *investment* and *banking*, which were entered by different users. Notice that although *funding* is also a hyponym of *finance*, if it does not actually annotate any service in the catalog, it won't be part of the tag cloud and thus it won't be considered during the search. Finally, the SOAlive catalog also defines a number of system-architected tags to annotate artifacts according to their type. These tags include *callable*, *browsable*, *deployable*, and *composable*. These tags also include the corresponding pseudo-derived tags *deployed* and *in-repo* (this last one to annotate any module in the repository).

**Bookmarks.** Bookmarking is another kind of collaborative filtering practice that a modern developer expects to be able to use to collect references to services of interest and to annotate those references with meaningful tags. Bookmarks can enrich the description of a given enterprise service  $S$  by allowing one or more users to refer to  $S$  with their own tags. This way, a hosted, community-centric platform such as SOAlive promotes collective informal semantic descriptions of enterprise services<sup>4</sup>.

The SOAlive catalog allows users to define and maintain bookmarks to collect references to services of interest, and to annotate those references with meaningful tags. Users can also browse other users' bookmarks; this contributes to making the SOAlive catalog a community-centric environment (not unlike *del.icio.us* and *flickr*) where developers share not only service offerings but

<sup>3</sup> Notice that it is also possible not to associate a sense with a tag, in which case such a tag does not participate in any grouping.

<sup>4</sup> Notice that a single user need only maintain a single bookmark per distinct service  $S$ .

also knowledge about other users' services that can be used for a particular purpose. The use of bookmarks also provides increased personalization of the SOAlive catalog by giving a user a view of the catalog contents through his or her bookmarks.

### 3.3 Code Snippet Generation

One goal of including a syntactic description of an artifact in its catalog main description is to provide support in the composition of services that use the artifact. From the RESTful documentation of a service it is straightforward to generate a code snippet that performs a simple invocation of the service from Javascript, Java or Groovy. For example, the following code snippet template can be used to invoke a service from Javascript, provided the fields that have the `<_cg_ . . . >` pattern are filled out. These fields can be supplied by a sufficiently complete syntactic description of the service.

```
var xhr = createXHR();
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == <_cg_successCode>)
      {
        var response = xhr.responseText;
      } else {
        <_cg_comment:_cg_errorCodes>
        alert("Error getting data from the server");
      }
  }
}

xhr.open("POST", <_cg_url>, true);
xhr.setRequestHeader('Content-Type', <_cg_format>);
var post_body = null;
<_cg_populate_body_from_example>
xhr.send(post_body);
```

Notice that, in particular, this code snippet does not perform any processing of the response. In general, there can be enough variability in what such a code snippet can do to make it infeasible for the catalog to attempt at a generic code snippet generation feature. Instead, the approach taken by the SOAlive catalog is to provide a plugin mechanism that allows a catalog-based tool developer to inject any arbitrary code snippet generator that uses the contents of a service's syntactic description as input. As a baseline, the SOAlive catalog includes a default plugin that generates simple code snippets for Javascript, Java and Groovy.

### 3.4 Discussion

As we have seen, the SOAlive catalog uses a simple content model that is suitable for describing enterprise services that are defined in terms of REST resources. This content model places minimal requirements on the author of a service, a

relative URI is enough as an initial description. Given a sufficiently complete syntactic description, in the form of RESTful documentation, the SOAlive catalog can provide support for embedding invocations of selected services by means of code snippet generation. The SOAlive catalog also takes advantage of collaborative practices to annotate service descriptions with informal semantic descriptions via tagging. The collection of all tags that annotate any artifact known to the catalog, published by any user, becomes a catalog-wide tag cloud that can then be used, as we shall see in the following section, to discover any artifact in the catalog. Finally, minimal syntactic descriptions can be made more complete and reliable by the use of a description refinement facility that the catalog provides as a way to enhance the content model. Section 5 elaborates on this refinement facility.

## 4 Discovery

As illustrated earlier, suppose that a credit check service has been published to the SOAlive catalog by Bob, and it can be used in the composition of a mortgage enterprise service being developed by Fred. In order for the credit check service to be effectively and efficiently composed into the mortgage service, the SOAlive catalog needs to provide a service discovery function that considers any service or artifact in the catalog, and is intuitive to Web 2.0 kinds of users.

The SOAlive discovery function is designed to allow a user to discover a catalog entry by drilling down on the tag cloud. That is, a user selects a tag from the tag cloud, which brings up all the catalog entries that contain the selected tag. Notice that if the selected tag  $t$  has any hyponyms in the tag cloud, then all tags in the hierarchy rooted at  $t$  will be considered when selecting catalog entries. That is, the selected entries will be all those that contain any tag in the hierarchy rooted at  $t$ . At this point the discovery function also displays a drill cloud, which is a tag cloud containing only tags in the current entry selection. The user can then select another tag from the drill cloud to further narrow down the contents of the entry selection. This procedure can be repeated until there is only one entry left or there are no more tags in the drill cloud. This procedure is an adaptation of [9].

However, notice that in [9] there is a separate drill cloud for each of a fixed number of categories, e.g., a keyword cloud and an author cloud. In addition, drill clouds are populated from the current selection of search results, which is initially obtained by using a domain-specific search form. In our case, the starting point of a search is the main tag cloud. In addition, tags do not fall under any given set of fixed categories; rather, tags are grouped according to their lexical sense. In some sense, our tag cloud can be thought of consisting of a dynamically changing set of categories, one for each top level tag at a given point in time. Thus, instead of trying to define separate drill clouds, a single drill cloud is used that collects all tags associated with any catalog entry in the current entry selection.

More precisely, the drill down discovery procedure consists of the following steps:

1.  $currentTag \leftarrow \text{Select}^{\text{5}} tag \text{ from main tag cloud}$
2.  $currentTags \leftarrow \{currentTag\}$
3.  $entrySelection \leftarrow \{entry \mid \text{for some } t \in currentTags \text{ hierarchy, } t \text{ annotates } entry\}$
4. do
  - (a)  $\left( \begin{array}{l} drillCloud \leftarrow \{tag \mid tag \text{ annotates some } entry \in entrySelection \\ \quad \quad \quad - currentTags \} \end{array} \right)$
  - (b)  $currentTag \leftarrow \text{Select } tag \text{ from } drillCloud$
  - (c)  $currentTags \leftarrow currentTags + \{currentTag\}$
  - (d)  $\left( \begin{array}{l} entrySelection \leftarrow \{entry \mid entry \in entrySelection \text{ and} \\ \quad \quad \quad \text{for } t \in currentTags, t \text{ annotates } entry\} \end{array} \right)$
 until  $entrySelection$  is singleton or user selects one entry explicitly

Given that the main tag cloud collects all tags that annotate any entry in the catalog, this procedure considers all such entries when starting a search. In our scenario, as Fred knows a credit check service has been published that he would like to invoke from his mortgage service, he looks in the tag cloud and clicks on *deployed*. This brings all services with tags *deployed*, *callable* and *browsable*, given that these last two tags have been entered for several other deployed services, including credit check, and given that these two tags are hyponyms of *deployed*. Fred then selects *callable*, as he knows he wants to incorporate the credit check service by adding a call to it to his code. At that point he notices a *credit* tag in the drill cloud and selects it, narrowing down the entry selection enough to make it easy to find the credit check service. At this point, Fred can also examine further details about the credit check service, including its RESTful documentation, after which he decides this is the service he wants to invoke. Finally, at this point Fred can also obtain the simple JavaScript code snippet that performs the invocation, includes it in his code and makes a few changes to handle the response.

## 5 Service Description Refinement

As Legner [6] suggests, complete and reliable service descriptions are a prerequisite for the discovery of suitable (Web) services. On the other hand, it is important to make it easy and painless for enterprise service developers to publish, advertise and discover services. In addition, since typically an enterprise service exposes a REST interface through which it is invoked, as opposed to using a rigorous language, such as WSDL, to define its interface, there seems to be less of a motivation for a developer to provide an interface, not to mention a complete and rigorous one. Also, although the RESTful documentation of a service may not be mandatory, the more complete it is, the better the code snippet

---

<sup>5</sup> This selection is performed by the user via some appropriate user interface.



that can be generated. This would suggest to a developer a requirement on the development host (and its catalog in particular) that the documentation itself be generated. Furthermore, given the makeup of RESTdoc in particular, it seems feasible to infer the various documentation items (e.g., format, parameters) by example from successful as well as unsuccessful invocations of a service. In other words, given a log of service invocations that include: the URL of the service, invocation method, format, parameter values, and success or error codes, it seems feasible to synthesize the RESTdoc documentation or interface of the service.

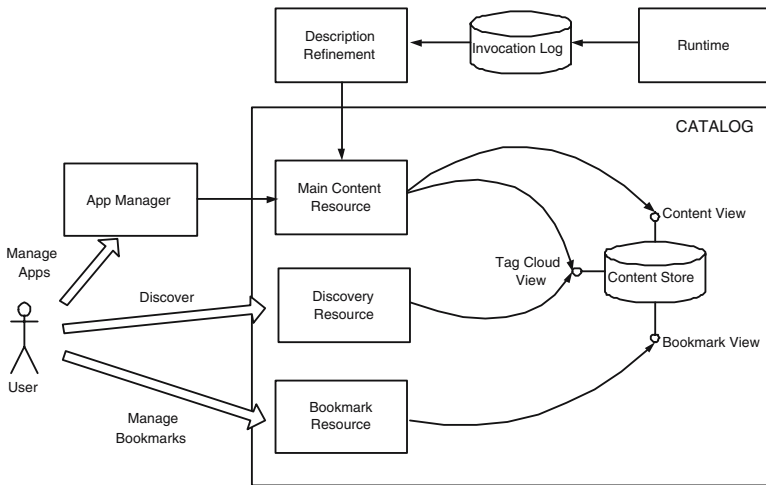
The SOAlive catalog includes a service description refinement function that allows the actual use of a service to refine the service description stored in the SOAlive catalog. This in turn allows the catalog to produce an increasingly accurate service description, without requiring the service provider to specify a highly detailed service description. The basic mechanism of service description refinement is a Monitor, Log, Analyze and Refine loop.

- **Monitor.** This is an interceptor that is registered with the SOAlive infrastructure, and that forwards service requests and responses to the log.
- **Log.** As requests and responses arrive, the log extracts and collects information such as: request and response timestamps, identity of client, request method, request and response formats, parameters the service was invoked with, return value, success or error codes, and parent request correlator (that can be used to trace a chain of requests).
- **Analyze.** On a thread separate to that of monitoring and logging, each log record is analyzed to determine what information in the log, if any, can contribute to the refinement of the service description. This analysis boils down to determining whether or not a log item has been accounted for in the service description. Items such as request format or error code may be as simple as determining whether they are included in a list. Other items, such as parameter values for requests that result in an error may depend on how the refine step accounted for them.
- **Refine.** Service refinement targets both the syntactic and the semantic descriptions of a service. Syntactic description items include data format (such as JSON or XML), success and error codes, example request parameters, and example response values. Data format, and even success and error codes, are typically given by a relatively small (certainly finite) set of values. So in this case it makes sense to simply accumulate logged values into the description. Successful request parameters and response values, on the other hand, would not make sense to simply accumulate. Here, it makes more sense to learn an abstract description of the values seen so far. In the case of XML-formatted requests, an XML schema seems adequate. For JSON-formatted requests, although it is not a type-checked language, a similar descriptive schema could be abstracted from incoming request examples. Parameter values in requests that result in an error are more challenging. In addition to a schema that describes possible error values, it would also be useful for a user to know what actual values resulted in error. So the refinement must strike a balance between collecting too much raw data or abstracting it too much.

Service refinement can also be thought of as either intra-service or inter-service. Intra-service refinement targets the syntactic description of a single service. With inter-service refinement, the logged data pertain to more than one service. For instance, a parent request correlator can be used to refine the semantic description a service. Specifically, if the service that made the invocation is known, then its tags can be used as input to augment the tags of the invoked service. Here, a similarity metric (such as semantic distance) could be used to determine which tags from the invoking service to keep and which to discard.

## 6 Implementation

An implementation of the SOAlive hosted environment is available that supports the main aspects of developing, deploying and executing enterprise services.



**Fig. 2.** SOAlive Catalog Architecture

The diagram in Fig. 2 illustrates the SOAlive catalog architecture. The service catalog is implemented as a number of REST resources that are also accessed locally by the application manager. In addition, the runtime and the service refinement function communicate indirectly via the log. The exposed REST APIs include create, read, update and delete operations for main content and bookmarks. There is also a REST API for discovery that retrieves the tag cloud and that returns a bookmark selection and corresponding drill cloud given a number of selected tags. In the current implementation, the actual drill down procedure is performed on the client via an encapsulated piece of JavaScript. A web-based graphical user interface (GUI) to the SOAlive hosted environment incorporates access to the catalog via its REST APIs. This GUI is illustrated in Fig. 3. Using

a general search tab or a application manager or bookmark-specific search tab, a user can start a tag cloud-based drill down discovery of a desired service. Current selections are shown on the right side of the pane, where actions to perform on each selections include showing the service interface as RESTdoc, from which a code snippet can be generated for any operation of any of the service's resources, as shown in Fig. 3.

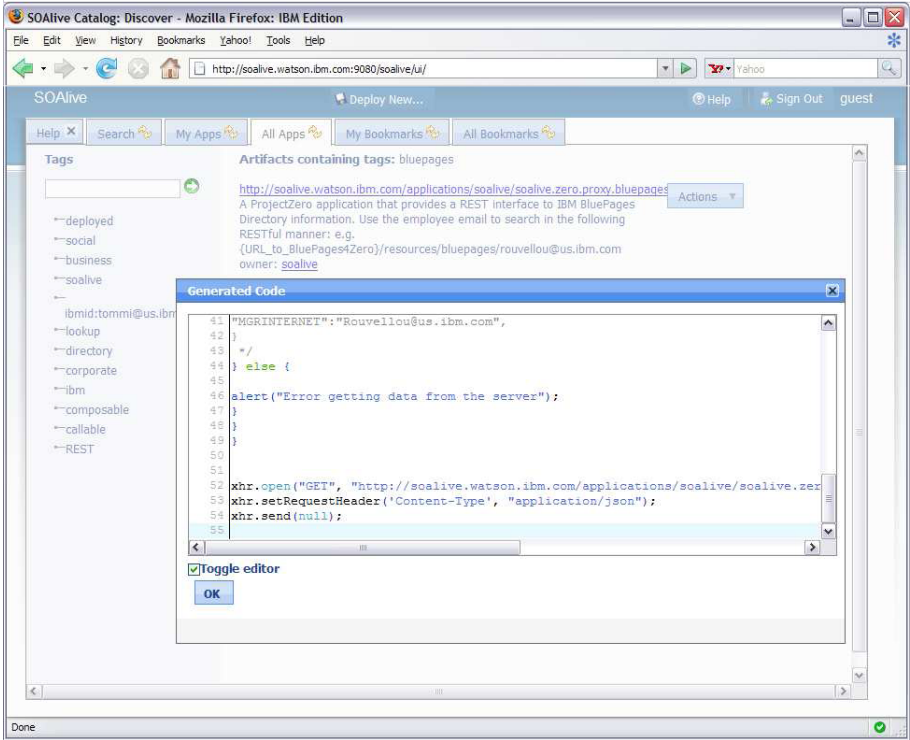


Fig. 3. SOAlive Graphical User Interface

The service refinement function is currently under development. The monitor and log portions of this function have already been incorporated into the SOAlive hosted environment, whereas a design of the analyze and refine portions is being completed, an outline of which is presented in section 5.

To illustrate the functions of the SOAlive catalog end to end, we now elaborate on the credit check service example that we first introduced in Section 3. Bob publishes a credit check service to SOAlive with the tag *credit*. This service gets published to the catalog. Users can now discover this service in many ways. A search on the term *credit* or *finance* (a tag derived from *credit* by the tag consolidation facility, see Section 3) or *composable* (a system-architected tag, see also Section 3) can help users find this service. Jim is composing a mortgage

approval service. He finds the credit check service and asks for a code snippet to include in his application. At this point, the credit check service's description is very minimal, and the catalog is able to provide very minimal code. Jim manages to fill in the gaps. He publishes his mortgage approval service to the catalog. The mortgage approval service is a very popular service and so the usage of the credit check service increases. Jane is composing a car loan approval service and is also in need of a credit check service. When Jane finds the same credit check service in the catalog and asks for help with code, she gets invocation code that is complete with error codes, success codes, examples, etc.

## 7 Related Work

Enterprise service discovery can be thought of as a recommendation system, where the discovery function recommends an initial selection of catalog entries based on a choice of tags. In [16], Zhao et al present a recommendation system based on collaborative tagging behaviors. There, two users are considered similar not only if they rate (or tag) items similarly (i.e., syntactically), but also if they have similar congnitions over these items. For instance if two users tag an item with the tags *photo* and *picture*, respectively, they could be considered similar even if their tags do not match exactly. This idea can also be applied to discovering items (enterprise services in particular) where even if an exact match query on a set of tags yields no results, a similarity-based match may yield a possible result. In this case, the set of tags are selected from the tag cloud. Catalog entries are then retrieved that match not only selected tags but also tags that are similar (via their hypernym/hyponym relationship in particular).

Web 2.0 techniques, such as wiki-based maintenance, are also related to service description. In [10], Paoli et al present an approach to describing (web) services that uses a UDDI registry complemented by a wiki-based semantic annotation subsystem. A developer publishes a service to the UDDI registry and “is encouraged to augment it by intuitive keywords found in the ontology”. This in turn results in the generation of a wiki page containing the developer’s keywords as well semantic links obtained by automatic reasoning from the ontology. This wiki page can then be used by other developer or business analysts for discovery and understanding. We notice however that this work depends “on an already existing and widely used taxonomy developed for the environmental information system of Baden-Wuerttemberg”. We believe that more lightweight Web 2.0 techniques such as social and collaborative filtering are better suited to the cataloguing and discovery of situational enterprise services, given their more dynamic and community-based nature.

Semantic personalization has previously been used in service discovery. Lord et al [7] propose a solution to the task of discovering semantic web services in a Bioinformatics Grid domain. This solution consists of a UDDI registry, augmented by a personalised view service and a semantic find service. The personalised view service provides a way to add user-specific metadata and thus filter the results returned by a query. The semantic find service relies on domain ontologies and a description logic reasoner. The personalised view service can be

used in isolation or in combination with the semantic find service. This work also depends on a rigorous, UDDI-based, syntactic description of services. In addition, while an ontology-based semantic description is suitable to this work, given its specific Bioinformatics Grid domain, it is less suitable to the more generic and Web 2.0-based domain of situational enterprise services. We should point out that by allowing the bookmarking and tagging of catalog entries, given its Web 2.0 motivation, the SOAlive service catalog is in effect providing a personalization approach to describing enterprise services.

## 8 Conclusions

The SOAlive service catalog provides a simplified approach to describing and discovering situational enterprise services. It incorporates support for light-weight description and discovery of enterprise services into the SOAlive community-centric service hosting environment. As we have seen, this incorporation not only improves the functionality of the SOAlive hosted environment but it also enriches the service catalog itself. Specifically, discovery is enhanced by its association with the environment's service deployment function and by the feedback from the service community supported by the hosted environment. Service refinement is enhanced by its integration with the environment's runtime that intercepts service invocations.

The service catalog, by its integration into the SOAlive hosted environment as a number of REST resources, becomes an enterprise service itself, one that is available to the other services hosted by the environment. This has the unintended consequence that services hosted by the SOAlive hosted environment can mash up the function of the catalog and extend its functionality. For instance, a simple enterprise service can provide user information for the developer of a hosted service by invoking the catalog's REST API, looking up the owner of the hosted service, and looking up detailed information for the owner in an enterprise user directory that is registered into the catalog as an external service.

The SOAlive service catalog's content model and code snippet generation function are designed with simplicity and extensibility in mind. This should prove useful as we look towards federation with heterogeneous catalogs, as well as towards integrating the catalog with other environments, which may have specific code snippet generation requirements.

## References

1. Curbera, F., Duftler, M., Khalaf, R., Lovell, D.: Bite: Workflow Composition for the Web. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. Springer, Heidelberg (2007)
2. Wikipedia definition. Situational application, [http://en.wikipedia.org/wiki/Situational\\_application](http://en.wikipedia.org/wiki/Situational_application)
3. Desai, N., Mazzoleni, P., Tai, S.: Service Communities: A Structuring Mechanism for Service-Oriented Business Ecosystems. In: DEST 2007: Digital EcoSystems and Technologies Conference (2007)

4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
5. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
6. Legner, C.: Is there a Market for Web Services? - An Analysis of Web Services Directories. In: Proceedings of Mashups 2007, 1st International Highlight Workshop on Web APIs and Services Mashups, Vienna, Austria (September 2007)
7. Lord, P., Wroe, C., Stevens, R., Goble, C., Miles, S., Moreau, L., Decker, K., Payne, T., Papay, J.: Semantic and personalised service discovery. In: Proc. UK e-Science All Hands Meeting 2003, EPSRC, pp. 787–794 (2003) ISBN 1-904425-11-9
8. Michael Maximilien, E., Wilkinson, H., Desai, N., Tai, S.: A domain-specific language for web apis and services mashups. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 13–26. Springer, Heidelberg (2007)
9. Newton, G.: Drill Clouds for Search Refinement. Blog by Glen Newton (October 2007), <http://zzzoot.blogspot.com/2007/10/drill-clouds-for-search-refinement-id.html>
10. Paoli, H., Schmidt, A., Lockemann, P.C.: User-driven semantic wiki-based business service description. In: 3rd International Conference on Semantic Technologies (I-Semantics 2007), Graz (2007)
11. ProjectZero. RESTful Documentation: <http://www.projectzero.org/~wiki/bin/view/Documentation/CoreDevelopersGuideRESTdoc>
12. WADL Specification, <https://wadl.dev.java.net/#spec>
13. SCA Specification, <http://www.oasis-open.org/sca/>
14. Tai, S., Desai, N., Mazzoleni, P.: Service communities: Applications and middleware. In: SEM 2006: Proceedings of the 6th International Workshop on Software Engineering and Middleware, pp. 17–22. ACM, New York (2006)
15. Multiple wiki authors. REST for the Rest of Us, [http://wiki.opengarden.org/REST/REST\\_for\\_the\\_Rest\\_of\\_Us](http://wiki.opengarden.org/REST/REST_for_the_Rest_of_Us)
16. Zhao, S., Du, N., Nauerz, A., Zhang, X., Yuan, Q., Fu, R.: Improved Recommendation based on Collaborative Tagging Behaviors. In: Proceedings of the International ACM Conference on Intelligent User Interfaces (IUI 2008), Canary Islands, Spain (2008)

# WorldTravel: A Testbed for Service-Oriented Applications

Peter Budny, Srihari Govindharaj, and Karsten Schwan

Center for Experimental Research in Computer Systems  
College of Computing  
Georgia Institute of Technology  
Atlanta, GA 30332, USA  
{peterb,srihari,schwan}@cc.gatech.edu

**Abstract.** This paper describes the “WorldTravel” service-oriented application and testbed. The purpose of the testbed is to provide to researchers an open source venue for experimenting with and evaluating ideas, methods, and implementation options for service-oriented architectures and applications. Built upon standard service technologies, the WorldTravel testbed offers implementations of services and service interactions specific to the WorldTravel application, comprised of (1) a substantive back-end that includes a simple airline pricing/ticketing engine, with a representative flight database, both structured similarly to those used by companies actually offering such services, (2) a front-end for travel services interacting with mid-tier request processing and routing services, and (3) load traces from the corresponding business applications that are used to drive the use of WorldTravel and its services.

We call WorldTravel a testbed rather than benchmark because its design permits extension at both the front-end, e.g., to add interesting new services like weather information about possible travel destinations, and at the back-end, e.g., to add payment services. This paper identifies the need for testbeds like WorldTravel, considers the attributes required of such testbeds, describes our current testbed in detail, and presents an initial testbed evaluation. It also describes the actual production-quality system on which WorldTravel is based.

## 1 Introduction

To pursue research in “service-oriented” architectures and applications (SOA), it is important to have available representative service examples and implementations, an analogous example being the well-known RUBiS eBay-like benchmark used to evaluate ideas and implementation methods for multi-tier web services [12]. RUBiS provides a representative back-end database, implementations of front- and mid-tier services that carry out tasks like searching for items and bidding on them, and load traces for front-ends that use these services. Since RUBiS is open source, researchers can extend or change it, using it to evaluate implementation ideas or methods for alternative ways to invoke services [1] and/or implement them, etc. In contrast, the well-known SPEC

benchmark for evaluating compiler technologies and TPC benchmarks for file systems, transactional services, and web services have as a principal goal to evaluate existing implementations and assess or compare their performance or reliability attributes.

Our goal is to enable research in service-oriented architectures, infrastructures that support their concepts, and applications based on these concepts. Toward those ends, this paper provides a novel testbed, called WorldTravel, which permits researchers to experiment with the SOA technologies of interest to them, construct their own SOA applications extending the front-end or back-end services of WorldTravel, and use WorldTravel load data to evaluate their implementations. Specific functionalities of WorldTravel highlighted in this paper are its methods for synchronous vs. asynchronous service interactions, the distinction of front-end from back-end services and the various service interactions across these different kinds of services, and the testbed's extensibility with respect to adding new services or changing existing ones. In addition, we differentiate the services-based approach used in WorldTravel from that taken in multi-tier web service testbeds like RUBiS [2], and articulate the need for testbeds like it.

In the remainder of the paper, we first define in Section 2 what we mean by “service-oriented architecture” with respect to the WorldTravel testbed, and separate it from related concepts like “Web Services”. Section 3 examines other potential testbeds to determine what qualities and attributes a successful testbed should exhibit. Section 4 explains the system we have chosen to model: an airline fare pricing engine. Section 5 details the implementation of WorldTravel. Sections 6 and 7 demonstrate how WorldTravel meets the previously-defined qualities and attributes of a successful testbed. Section 8 discusses future work to be done on WorldTravel and concludes the paper.

## 2 Service-Oriented Architecture

Service-oriented architecture, or SOA, is an architectural design pattern in which an application is composed of loosely-coupled components that export and import services [3]. A service is a function in which the request is posed as a question, and the response is the answer to that question. In SOA, each service is specialized to only answer certain types of questions. Applications are built by composing services with each other statically or, more interestingly, at runtime.

SOA is the architectural equivalent of abstraction. Since each service provider solves a single problem, this makes it possible to build interesting applications that perform complex tasks by letting service providers at lower levels solve some of the problems, thereby freeing the upper-level from having to worry about these problems.

In the rest of this section we will refine the concept of “service-oriented architecture” as we use it with the testbed we propose. Our intent is not to constrain users to a single definition but rather to clarify the broad mindset taken as we designed the testbed.



## 2.1 SOA and Web Services

Since SOA applications typically run in Internet settings, the term SOA is often linked with “Web Services”, the latter characterizable in multiple ways:

1. web services, *noun* – a collection of standards including SOAP, XML, WSDL, UDDI, and WS-\*;
2. web service, *noun* – an architectural design pattern in which programmatic interfaces allow two applications to communicate directly to each other; [Web service standards (1.) are being created to support web service architectures (2.).]
3. web service, *noun, adj.* – an application constructed using web service standards (1.) *or* web service architecture (2.).

Perhaps most commonly, the term “web service” is used somewhat narrowly to describe “an application designed as a SOA and implemented with ‘web service’ standards”. When used in this sense, web services are a proper subset of SOA.

## 2.2 Defining SOA

Having addressed web services, we next discuss the specific qualities beyond the notion of web services associated with SOA, in part to address some common misconceptions that derive from the fact that SOA implementations often use web services [4].

**Concept.** An application is service-oriented iff it uses certain standards.

**Reality.** SOA indicates the style in which components of a solution are tied together, while standards define specific implementations for doing so. Although standards encourage competition and interoperability by making it easy to switch service providers, this concept is often negated by the use of proprietary standards for the interconnection of services to promote vendor lock-in.

**Concept.** SOAs communicate using XML.

**Reality.** SOAs can be built using any format to exchange messages and data, where such alternatives are often used for reasons of improved performance (e.g., binary formats [5,6,7]) or to support legacy applications.

**Concept.** SOA precludes the use of RPC [8], RMI [9], or REST [10].

**Reality.** Each of these invocation protocols have attributes that make integrating them into a SOA difficult, yet such integration is often done. RPC and RMI are tightly-coupled compile- or link-time protocols integrated into the middleware methods used by distributed applications. This makes them most suitable for somewhat ‘static’ components of these applications (e.g., core back-end services like the database accesses in RUBiS), and while SOA encourages adaptability, services invoked using RPC or RMI can certainly be substituted with only moderate effort. REST, on the other hand, encourages transferring data using state, which makes ensuring idempotency difficult (see [Idempotent requests](#)), but is inherently well-suited for stateful services. With care, it can also be used to implement stateless services.

**Concept.** SOAs implement service discovery or a service registry.

**Reality.** Runtime service discovery and subsequent service use face difficulties in that it is not reasonable to assume that all providers of services are ‘equal’. As a result, there is much recent research on dealing with service equivalence or translation, resulting in ontology-based methods for understanding degrees of similarity or equality and semantic web-based methods for doing so [11,12]. Beyond such functional equivalences, also of interest are performance or reliability differences between services, addressed in part by recent work on standards in the domain of autonomic computing [13,14].

The high level features of SOA described above are implemented using several basic techniques:

**Composition.** SOA is a form of distributed computing in which subtasks are distributed and treated as blackbox operations, possibly even handled by third parties. Applications are structured by abstracting and composing services vertically to provide desired higher level functionality.<sup>1</sup> Composition is intended to be dynamic, but issues persist concerning the viability and generalizability of dynamic composition for commercial applications with required service guarantees.

**Descriptive requests.** Queries in a SOA should describe the problem to be answered, not how to solve it programmatically. This means that queries in SOAs should only be interpretable as questions, not as procedures [16]. (SQL, for instance, is *not* service-oriented; every SQL command contains a verb (e.g., SELECT, INSERT, DELETE, etc.) and describes how to manipulate data. A hypothetical service-oriented database manipulation language would instead have ‘question’ words like “what” or “how many”.)

**Idempotent requests.** Since a service answers a question, it is expected that the answer should not change between requests. However, services will have finite, dynamic resources, which means that a response involving a resource may change when the underlying resource has been modified. For stateful services, responses may differ due to changes of internal states caused by previous requests, but responses must still be idempotent for a combination of a request, the state at the time of the request, and the resources involved.

**Structured responses.** A service’s response must be structured data which answers the question posed by a request. Arbitrary, unstructured data cannot make up the entire response.

### 3 Utility of a SOA Testbed

We developed the WorldTravel testbed for multiple reasons. First, it can provide a useful basis for experimentation, both with new services and service-service interactions and with new methods that improve SOA implementations (e.g.,

<sup>1</sup> In contrast, horizontal composition, as exemplified by load balancing, MapReduce [15], and many other well-known techniques, is used most often as a way of enhancing performance, and is orthogonal to SOA; horizontal composition/distribution can be used both within a service and between equivalent services.

improved discovery methods, improved invocation methods, etc.). We also hope to provide researchers with an environment in which such ideas can be compared, in performance and/or usefulness, but it remains difficult to assess the performance effects of functionality like runtime service discovery and use, since ‘typical’ behaviors for such actions are not yet known. On the other hand, just as compiler developers agree on certain workloads to be representative of certain environments (e.g., selecting certain SPEC benchmarks), for testbeds like WorldTravel, useful request traces and request loads can be derived both from standard Internet measurements (e.g., diurnal changes in request behavior [17,18]) and in our case, from load information provided by our corporate partner, Travelport. A final purpose of the WorldTravel SOA testbed is for it to give rise to a set of testbeds (developed outside our group) that will display common, defining characteristics of SOAs.

### 3.1 Other Potential Testbeds

We are not aware of other SOA testbeds, but point the reader to the following related efforts that have helped shape WorldTravel and its implementation.

*RUBiS* is an online auction simulation modeled after eBay [2]. It is built using a standard three-tier architecture, has a sample database and offers representative load traces. Since RUBiS is focused on a single application, bidding, opportunities exist to extend its back-end (e.g., payment methods) or front-end (e.g., comparative bidding), but such extensions are difficult to perform because they must be intimately integrated into the RUBiS implementation. We note that there are other applications like RUBiS for Java-based multi-tier web service implementations, available from companies like IBM, but since they rely on IBM’s Websphere middleware, they are not suited for constructing a suitable testbed. Finally, earlier versions of applications like RUBiS are even simpler (e.g., PetStore) and are also not useful building blocks for our work.

*Java Adventure Builder* is a sample application demonstrating web service standards on the J2EE platform [19]. We considered using it as a basis for our work, but it does not offer the abstraction necessary to be considered service-oriented; rather, it is structured using a tightly-coupled three-tier architecture. Further, because it is built solely on web service standards, exploring alternative standards in its context would imply a complete re-implementation of its functionality. Lastly, the data set provided with it is quite small and is not representative of a fully functioning SOA.

*Nutch* is a search engine based on Lucene Java, an indexing and search back-end [20]. Search engines certainly constitute an interesting class of service providers, but the open source Nutch implementation does not use service-based interactions with the associated web crawler. Instead, it requires the index to be produced by users based on crawling their own set of websites. It does offer some plug-ins for media-type parsing, data retrieval, querying and clustering, but those plug-ins do not use standard service interfaces or SOA methods. Finally, it does not offer other interesting interfaces, such as those concerning ad generation

and placement, etc. *Hadoop* is a related effort that enables end-users to construct their own MapReduce functions to assist in fast, scalable searching [21].

*Intel Mash Maker* is a tool for allowing non-expert users to create mashups, or queries on two or more related data sets [22]. Although it is primarily designed to take in semi-structured data, it could also take in fully-structured data (i.e., consume services). It could also be construed as a service provider in its own right. However, its implementation within a single web browser reduces it to being a single service consumer/provider, rather than being a complete SOA. The fact that it is not open source discourages its use research environments.

*Yahoo! Pipes* is a web application providing users with simple building blocks to aggregate web feeds, web pages, and other services, manipulate and combine content to create new web applications, and publish the resulting applications [23]. Like Mash Maker, it can take in structured data, and it provides services by offering them in a publicly-available fashion. However, it has a limited set of sources for structured data; data from other sources can only be fetched as unstructured strings. Further, since only basic data manipulation functions are provided, it would be difficult to build a practical business application with the available functions. It is also not open source and is hosted solely by Yahoo!, therefore limiting its use in research environments.

*Apache Tuscany* is an infrastructure for creating SOAs based on specifications defined by the Open SOA Collaboration [24], but is not itself a SOA or a service-oriented application.

*httpperf* and *StreamGen* are tools focused on specific functionality useful for testbeds like WorldTravel, benchmarks like RUBiS, and streaming applications like those developed in the multimedia domain [25] and for database query-structured business monitoring or compliance codes [26,27,28]. Their role is to make it easy to generate request streams using standard loads (e.g., normal or Poisson distributions) and/or to offer non-standard loads acquired from trace files in a standard form. We use both in our research and with WorldTravel.

### 3.2 Criteria for a Testbed

In accordance with the issues raised in the previous section, we articulate the following criteria for our WorldTravel and other useful SOA testbeds. They should:

- be executable, complex, functional applications built from multiple composable units where each unit should be a service provider and/or consumer, and every service provider should be meaningful on its own, separately from the services built on top of it;
- be extensible, particularly because a key element of SOA is its support for composition of services to provide new functionality; thus, for our SOA testbed, it should be easy to construct new applications in its context and with its implementation;
- come with large data sets, to better represent realistic commercial applications that deal with large volumes of data;
- be open source;
- be reusable, not purposed for a single experiment or class of experiments;

- be easy to integrate with other tools, monitors, and benchmarks, and easy to change to permit experimentation with alternative composition methods, new discovery methods, etc.; and
- not rely heavily on specific standards, so they can support and be used to evaluate alternatives.

## 4 Reference System

As a reference domain, we use the airline travel industry; specifically, systems that price and book airline tickets [29], which are termed *global distribution systems* (GDS). A GDS provides services that include pricing and ticket sales for major airlines, independent travel agents, select airline sites (e.g., Delta Air Lines in the case of our corporate partner, Travelport), and travel websites like Expedia or Orbitz, the latter permitting customers to independently search for suitable fares and purchase airline tickets.

### 4.1 About Airline Fares

Prices for airline flights are not static; they are calculated dynamically from rules. These rules are collected and published several times daily. During the  $\sim 11$  months in which tickets for a flight are available, airlines modify the rules governing the pricing of that flight to reflect supply and demand. In addition, pricing rules utilize run-time input such as:

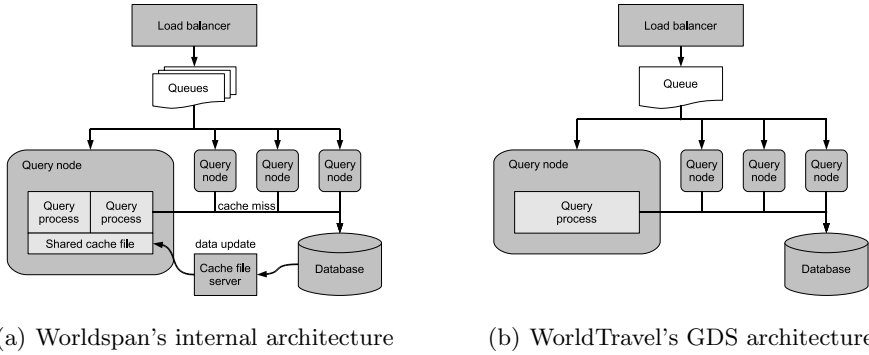
- the date and time of the desired flight;
- the travel class (i.e., first, business, or economy, which are normally broken into as many as 20 “buckets”);
- the number of seats currently available; and
- various discounts and restrictions that may apply (e.g., senior citizen discounts, advance purchase requirements, blackout dates, etc.).

Pricing a flight is a matter of finding the combination of discounts and restrictions that yield the lowest price. In fact, the GDS may be liable for any differences in price from what the airline quotes as the correct price. Responses must be returned within a well-defined time and with well-defined complexity (e.g., number of fare options returned) governed by SLAs negotiated with services that use the GDS (e.g., Priceline).

### 4.2 About Worldspan

Our particular architecture comes from Worldspan by Travelport, a GDS whose users include Delta Air Lines, Expedia, Orbitz, Hotwire, and Priceline. These retailers use Worldspan to calculate ticket prices, check seat availability, and book and purchase tickets. Worldspan in turn relies on airline fare consolidators to provide a data feed for updates to pricing rules, and also communicates directly with airlines to check seat availability and book tickets.

Worldspan serves an average of 11.6 million requests per day, with an average response time of  $\sim 2$  seconds. The data set is 3 GB for domestic (i.e., U.S.



(a) Worldspan's internal architecture

(b) WorldTravel's GDS architecture

**Fig. 1.** WorldTravel uses a simplified version of Worldspan's architecture

and Canada) pricing data, which is updated three times daily, and 13 GB for international, updated five times daily<sup>2</sup>

Worldspan's architecture is based around a 1500-node farm used to process pricing queries; each node runs two query processes. Pricing data is stored in four load-balanced SQL database servers, which are accessed in two different ways. First, after updating the database with new prices, data for frequently-used markets is loaded into a 1.5 GB cache file which is then pushed to the nodes in the farm. This cache file is read into shared memory and is used by the two query processes. Second, the databases serve requests on-line for data not contained in the cache file.

## 5 Implementation

The WorldTravel testbed is based on a simplified version of Worldspan and the systems with which it interacts. It has been designed to meet the aforementioned goals of a SOA testbed.

The WorldTravel design and implementation are deliberately straightforward and limited in scope and complexity. For instance, query processes do not have caches, so that data is pulled from the database on every request. This also makes it easy to experiment with alternative caching methods and implementations. The GDS also does not have the ability to process data updates or sell tickets, which makes it possible to experiment with alternative ticketing or payment services. The front-end uses a standard web server, but the mid-tier request distribution service uses open source web technologies rather than the proprietary middleware from Delta Air Lines used in our earlier work [30] or IBM's commercial MQ middleware used by Worldspan [31].

WorldTravel is currently available at <http://www.cc.gatech.edu/systems/projects/worldtravel/>

<sup>2</sup> International pricing queries also require the domestic data, so the data for international queries is actually 16 GB and updated eight times daily.

## 5.1 Overview

As a service-oriented application, we make a distinction between a travel website, which provides an interface for users to easily search for fares, and a GDS, which performs the task of pricing fares. A GDS may provide the same information to many different websites or to other clients, and the price of a flight can be calculated by anyone with access to the data. The result is a strong incentive for use by multiple end-user services (e.g., those provided by Expedia, Priceline, and the airlines).

One of the complexities of service-oriented applications is that it may be unknown how long a request will take to process. As a result, asynchronous operations are common in SOA. Acknowledging this fact and for generality, WorldTravel implements multiple communication methods between services: (1) asynchronous with polling, (2) asynchronous with call-backs (i.e., event-based), and (3) synchronous, implemented as a wrapper around a polling interface. The services we implemented in WorldTravel all use asynchronous communication with call-backs, which frees the applications from having to keep state about ongoing queries and makes it easier to scale them. It also lets us potentially deliver responses to a different node than the one that originated the request, thus further enhancing scalability. The synchronous wrapper, on the other hand, is provided for simple applications to use, and frees them from having to handle multiple connections, instead enabling a simple call-and-response invocation style much like HTTP or other web protocols.

To protect Worldspan's intellectual property, we have applied transformations to the data they have provided so that it cannot be used to price actual flights. Ersatz prices are generated and can be used by other services (e.g., payment services or price comparison services). Unlike Worldspan's optimized implementation, which is tuned using domain knowledge about the pricing rules, the price search we have implemented is much more straightforward. This gives users the freedom to tune the engine as desired (e.g., by optimizing the search order, multithreading, etc.). This is because our primary goal is to provide a representative workload on a system constructed using SOA principles; while prices calculated by WorldTravel may be artificial due to the data transformations, the system as a whole still exhibits the same characteristics as Worldspan's functioning application.

WorldTravel is implemented as several distinct components. A minimum setup requires five nodes: three for the GDS (database server, query node, and load balancer), one for the travel website, and one for the load generator. The load balancer and travel website are implemented in Apache Geronimo; the query node in plain Java. The database server we use is MySQL, but any compatible database can be substituted.

## 5.2 GDS

The GDS consists of a database, one or more query nodes, and a load balancer. The load balancer acts as the front-end of the pricing service, accepting requests

and returning responses once they have been calculated. Inside the GDS, the load balancer communicates asynchronously with the query nodes via queues. The use of queuing to pass queries from the load balancer to query nodes and back is again modeled on Worldspan's architecture, which uses reliable queues to guarantee message delivery. By using multiple queues and a "partitioning" load balancer, Worldspan can also segment their query nodes for specialization (e.g., differing caches representing different markets).

### 5.3 Travel Website

The WorldTravel travel website is a generic clone of websites like Expedia or Orbitz. The website is also asynchronous, much like a real travel website. Upon submitting a pricing request, users are shown a page asking them to wait, which periodically refreshes to check if a response has arrived at the web server. If so, the response is processed and displayed; if not, the wait page is displayed again.

### 5.4 Customer

The customer is represented by a load generator, which generates requests to the travel website. The loads come from request traces provided by Worldspan, and they reflect some of the complexities particular to travel websites, such as geographical searches that vary based on time of day due to global users being in disparate time zones.

## 6 Testbed Analysis

By choosing a reference application and designing a system with the criteria mentioned earlier in mind, we are able to produce a system capable of serving as a SOA testbed. WorldTravel successfully emulates a complex, multi-layer system, structured using service-oriented architecture, which means the GDS and the travel website are each independent implementations, meaningful and useful without other applications using them or based on them. A multitude of interesting extensions and changes are possible, as discussed next.

WorldTravel can be expanded in many ways, and it offers rich possibilities for composition with other services. The testbed can be extended with front-end services or other well-known tools, such as Java Adventure Builder, Intel Mash Maker, Nutch, etc., or with creative new applications. One idea currently being pursued by our group is the addition of services like event ticketing and hotel booking, to create a system which can handle multiple reservations (e.g., an event ticket, a hotel, and a flight) in a transactional manner, guaranteeing that simultaneous reservations will be booked successfully. In addition, several components used at Worldspan are absent in our current testbed. They include back-end services like seat availability, ticket purchasing, and pricing rule updating. These, as well as internal components like data caching and load segmentation, can be added to expand the scope and depth of the simulation. Its composability



with other services at both the front- and back-ends is what makes WorldTravel suitable to act as a SOA testbed.

A variety of useful experiments can be built on top of WorldTravel. As a whole, WorldTravel can be used for research into the complexities of designing and managing SOA systems, such as SLA management and enforcement, service discovery and equivalency, etc. The load data provided with WorldTravel has interesting properties that are well suited to experiments with load distribution and autonomic management [32]. Finally, the services provided by WorldTravel can be composed with other services, and are suitable for research into dynamic composition, as is done by Yahoo! Pipes, for example.

WorldTravel is built with open source technologies, to enable changes and extensions to its implementation. This includes Apache Geronimo, JMS, MySQL, etc. Its transparent design encourages modification and integration with other tools and applications, such as monitoring infrastructures used to conduct experiments or enable system performance tracking and management.

In order to serve as a testbed for a broad range of SOA implementations, WorldTravel uses a generic architecture and refrains from tightly integrating with any specific platforms, tools, or standards. Specifically, we have not adopted web service standards (WS-\*); the system can be integrated with these standards as validation of their applicability and utility, but does not rely on them, thereby making WorldTravel suitable for evaluating future standards, as well. A few specific standards used in WorldTravel include: XML, to encode requests and responses, which will enable future extensions that analyze and modify the semantic information contained within; SOAP, as a communications protocol; and WSDL, to identify service endpoints. Each of these standards were chosen to maximize utility and simplify the implementation of WorldTravel, but none are essential to its operation; each can be substituted in order to evaluate alternative standards or to integrate with other tools and applications which use different standards.

Perhaps most importantly, WorldTravel is distributed with a large data set generously provided by Worldspan. This includes anonymized pricing data that covers several major airlines, regional airlines, and small competitive low-cost airlines. The total size of this data is > 1 GB. When we implement data updates, Worldspan will also provide us with additional data representative of typical update feeds. Finally, Worldspan has also provided request traces, which are used to generate realistic traffic patterns on the travel website and the GDS.

## 7 Experimental Evaluation

In addition to describing how WorldTravel is designed to meet the abstract criteria for an effective testbed, we must also demonstrate that it represents some of the complexities of commercial applications, making it useful as a SOA testbed. This task is complicated by the fact that some behaviors do not manifest themselves without additional software and architecture, such as caching, load segmentation, and more advanced query processing. However, the data used with WorldTravel has intrinsic properties, and we can demonstrate that the WorldTravel implementation displays behaviors that relate to these properties. This

**Table 1.** Number of fares filed per market

Origin airport	Destination airport	Number of fares
HNL (Honolulu, HI)	SEA (Seattle, WA)	4338
HNL (Honolulu, HI)	PDX (Portland, OR)	3996
OGG (Kahului, HI)	SEA (Seattle, WA)	3840
OGG (Kahului, HI)	PDX (Portland, OR)	3782
HNL (Honolulu, HI)	SAN (San Diego, CA)	3607
OGG (Kahului, HI)	SAN (San Diego, CA)	3540
KOA (Kailua-Kona, HI)	SEA (Seattle, WA)	3538
	⋮	
RNO (Reno, NV)	WRL (Worland, WY)	1
SJU (San Juan, PR)	YQK (Kenora, ON, Canada)	1
STS (Santa Rosa, CA)	TUP (Tupelo, MS)	1
STS (Santa Rosa, CA)	TYS (Knoxville, TN)	1
STS (Santa Rosa, CA)	VPS (Eglin AFB, FL)	1
STT (St. Thomas, USVI)	YQK (Kenora, ON, Canada)	1
YEV (Inuvik, NWT, Canada)	YOQ (Ottawa, ON, Canada)	1

makes WorldTravel suitable for testing solutions pertaining to client request-driven applications with varying response costs and complexities.

One such property is that not all markets (that is, pairs of origin and destination airports) have the same number of fares filed. Airlines may file short-lived fares in small markets as a way of enacting discount sales.<sup>3</sup> They may also file many fares in a single market to implement complex pricing rules. A quick examination of the database, shown in Table 1, shows that this distribution of fares among markets exhibits a strong geographical bias; the markets with the most fares in the database are all Hawaiian airports, while the markets with the fewest fares are mostly small airports.

We construct a simple test in which we measure the time taken by a query node to process a request, indexed by the number of matching fares available in the database. We augment WorldTravel to report the amount of time spent handling a request in the query processor. For the experiment, we randomly choose markets to request, testing both markets which have few airlines filing fares (typically 1 to 3 airlines) and which have many airlines filing fares (increasing as the number of fares increases, up to 7). After an initial request to ensure that the data has been cached by the database, we make several requests and calculate the average processing time. Table 2 on the next page shows the results of this experiment.

The results demonstrate a correlation between the number of fares filed in the requested market and the time required to process a request. Since there

<sup>3</sup> This is possible even when there is no direct flight between the two airports; pricing data and flight data are strictly separate. A single fare may be used to price multiple flights with connections, while a single itinerary may be priced as a combination of fares representing one or more connections.

**Table 2.** Time taken to process a request versus market

(a) Markets with the most airlines publishing fares

Market	Number of fares	Avg. time to process(ms)
AVL-LEX	50	40.8
GEG-IND	100	42.8
GSO-YTO	200	47.3
BOI-SJU	305	51.0
HNL-MEM	402	53.2
EWB-OGG	518	58.5
LIH-PDX	1891	105.2

(b) Markets with the fewest airlines publishing fares

Market	Number of fares	Avg. time to process(ms)
TWF-VLD	50	40.0
SJU-SLK	100	40.5
MBS-OGG	200	43.3
OGG-VPS	302	47.7
BTR-OGG	405	49.7
JHM-PDX	506	52.8
ITO-PDX	1748	99.3

is also a correlation between market geography and number of fares filed, we can conclude that the processing time for a request is affected by the market of the request. This could be exploited to provide different processing methods for requests that are expected to require more processing time based on their geography. Indeed, Worldspan sees this effect on a much greater scale: queries for international markets have up to four times as much data, and their processing times are significantly higher. In their production system, therefore, Worldspan separates international queries from domestic queries and uses faster machines to process international queries in order to avoid service timeouts.

The simple experiment discussed above demonstrates that the WorldTravel testbed exhibits behavior much like Worldspan’s application. This constitutes initial evidence that the testbed is suitable for testing and analyzing techniques and solutions of use to Worldspan and other real-world systems.

## 8 Conclusions and Future Work

This paper presents the WorldTravel testbed for SOA architecture investigations and for constructing and experimenting with SOA applications. The paper defines “service-oriented architecture” and its distinction from “web services”, and it shows that the WorldTravel testbed can act both as a clean model for SOA and as a common ground for promoting and understanding future research and applications in this domain. The WorldTravel simulation system is based on an actual class of commercial applications in the airline travel industry, and it is constructed to exemplify the characteristics of SOA. An analysis of the current testbed demonstrates that it meets key SOA requirements.

Because we have simplified Worldspan’s architecture in order to realize an easily distributed and extensible system, there are many opportunities for extending it. They include both those used in commercial settings like at Worldspan and new creative front- or back-end services that expand the testbed with new features and additional functionality. Also missing for this initial implementation is an evaluation of its composability with existing SOA-based services, includ-

ing those that are publicly available. We are currently undertaking this task. Further, we will also attempt to integrate other applications often cited as being service-oriented, such as Java Adventure Builder or Intel Mash Maker, to demonstrate that many service-oriented applications can be integrated into the testbed. Finally, we will explore how the various standards for managing SOAs (e.g., web service standards, Tuscany, etc.) can be applied to the testbed.

## Acknowledgments

We are most grateful to Sameh Abdelaziz and his team at Worldspan for their partnership and cooperation in allowing us access to their systems, and for providing us with documentation about the airline industry, sample data, and request traces that capture the complexity of the problems occurring in realistic enterprise applications. The authors would also like to thank Richard Bailey for his help in proofreading the paper.

## References

1. Cecchet, E., Marguerite, J., Zwaenepoel, W.: Performance and scalability of EJB applications. In: Proceedings of the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 246–261 (2002)
2. Rice University (RUBiS), <http://rubis.objectweb.org/>
3. Rotem-Gal-Oz, A.: What is SOA anyway? <http://rgoarchitects.com>
4. Kodali, R.R.: What is service-oriented architecture? JavaWorld.com (June 2005)
5. Bustamante, F., Eisenhauer, G., Schwan, K., Widener, P.: Efficient wire formats for high performance computing. In: Supercomputing, ACM/IEEE 2000 Conference, p. 39 (2000)
6. Chiu, K., Devadithya, T., Lu, W., Slominski, A.: A binary XML for scientific applications. In: E-SCIENCE 2005: Proceedings of the First International Conference on e-Science and Grid Computing, pp. 336–343. IEEE Computer Society, Washington (2005)
7. Seshasayee, B., Schwan, K., Widener, P.: SOAP-binQ: High-performance SOAP with continuous quality management. In: Distributed Computing Systems, Proceedings. 24th International Conference, pp. 158–165 (2004)
8. Nelson, B.J.: Remote procedure call. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1981)
9. Sun Microsystems (Remote method invocation), <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>
10. Fielding, R.T.: Representational state transfer (REST). In: Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine (2000)
11. Clerkin, P., Cunningham, P., Hayes, C.: Ontology discovery for the semantic web using hierarchical clustering (2001)
12. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web services. In: Web Semantics: Science, Services and Agents on the World Wide Web, pp. 27–46 (2003)

13. Zhang, L., Ardagna, D.: SLA based profit optimization in autonomic computing systems. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 173–182. ACM, New York (2004)
14. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (WS-Agreement)
15. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: OSDI 2004: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, p. 10. USENIX Association, Berkeley (2004)
16. He, H.: What is service-oriented architecture. XML.com (September 2003)
17. Shannon, C., Moore, D., Keys, K., Fomenkov, M., Huffaker, B., Claffy, K.: The internet measurement data catalog. SIGCOMM Comput. Commun. Rev. 35(5), 97–100 (2005)
18. Sripanidkulchai, K., Maggs, B., Zhang, H.: An analysis of live streaming workloads on the internet. In: IMC 2004: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, pp. 41–54. ACM, New York (2004)
19. Sun Microsystems (Java adventure builder reference application), <https://adventurebuilder.dev.java.net/>
20. Apache Software Foundation (Nutch), <http://lucene.apache.org/nutch/>
21. Apache Software Foundation (Hadoop), <http://hadoop.apache.org/core/>
22. Ennals, R.J., Garofalakis, M.N.: MashMaker: Mashups for the masses. In: SIGMOD 2007: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 1116–1118. ACM, New York (2007)
23. Yahoo! (Yahoo! Pipes), <http://pipes.yahoo.com/>
24. Apache Software Foundation (Apache Tuscany), <http://incubator.apache.org/tuscany/>
25. Li, M.L., Sasanka, R., Adve, S., Chen, Y.K., Debes, E.: The alpbench benchmark suite for complex multimedia applications. Iiswc 0, 34–45 (2005)
26. Kumar, V., Cai, Z., Cooper, B.F., Eisenhauer, G., Schwan, K., Mansour, M., Seshasayee, B., Widener, P.: Implementing diverse messaging models with self-managing properties using IFLOW. In: IEEE International Conference on Autonomic Computing, ICAC 2006, pp. 243–252 (2006)
27. Mosberger, D., Jin, T.: Httperf—a tool for measuring web server performance. SIGMETRICS Perform. Eval. Rev. 26(3), 31–37 (1998)
28. Mansour, M., Wolf, M., Schwan, K.: StreamGen: A workload generation tool for distributed information flow applications. In: ICPP 2004: Proceedings of the 2004 International Conference on Parallel Processing, pp. 55–62. IEEE Computer Society, Washington (2004)
29. Mansour, M., Schwan, K., Abdelaziz, S.: I-Queue: Smart queues for service management. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 252–263. ACM, New York (2006)
30. Kumar, V., Cooper, B.F., Cai, Z., Eisenhauer, G., Schwan, K.: Resource-aware distributed stream management using dynamic overlays. In: ICDCS 2005: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, pp. 783–792. IEEE Computer Society, Washington (2005)
31. IBM (WebSphere MQ), <http://www-306.ibm.com/software/integration/wmq/>
32. Kumar, V., Schwan, K., Iyer, S., Chen, Y., Sahai, A.: The state-space approach to SLA-based management. In: IEEE/IFIP Network Operation & Management Symposium, NOMS (2008)

# TCP-Compose\* – A TCP-Net Based Algorithm for Efficient Composition of Web Services Using Qualitative Preferences\*

Ganesh Ram Santhanam, Samik Basu, and Vasant Honavar

Department of Computer Science, Iowa State University, Ames IA 50011, USA  
{gsanthan,sbasu,honavar}@cs.iastate.edu

**Abstract.** In many practical applications, trade-offs involving non-functional attributes e.g., availability, performance play an important role in selecting component services in assembling a feasible composition, i.e., a composite service that achieves the desired functionality. We present TCP-Compose\*, an algorithm for service composition that identifies, from a set of candidate solutions that achieve the desired functionality, a set of composite services that are *non-dominated* by any other candidate with respect to the user-specified qualitative preferences over non-functional attributes. We use TCP-net, a graphical modeling paradigm for representing and reasoning with qualitative preferences and importance. We propose a heuristic for estimating the preference ordering over the different choices at each stage in the composition to improve the efficiency of TCP-Compose\*. We establish the conditions under which TCP-Compose\* is guaranteed to generate a set of composite services that (a) achieve the desired functionality and (b) constitute a *non-dominated* set of solutions with respect to the user-specified preferences and tradeoffs over the non-functional attributes.

## 1 Introduction

Service-oriented computing [1,2,3] offers a powerful approach to assemble complex distributed applications from independently developed software components in many application domains such as e-Science, e-Business and e-Government. Consequently, there is a growing body of work on specification, discovery, selection, and composition of services. The focus of this paper is on service composition, i.e., the problem of assembling a composite service (goal service) from component services from *functional* and *non-functional* specifications.

Functional requirements specify the desired goal service functionality. Barring a few notable exceptions [4,5,6,7], much of the work on service composition has focused on algorithms for assembly of composite services from functional specifications. Some of the major approaches to service composition based on

---

\* This work is supported in part by NSF grants CNS0709217, CCF0702758 and IIS0711356.

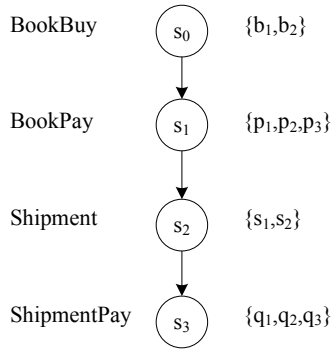
functional specifications include: AI planning [8,9,10,11], labeled transition systems [12,13,14], Petri nets [15], among others. (The interested reader is referred to [16,17,18] for surveys).

Non-functional requirements refer to aspects such as security, reliability, performance, and cost of the goal service. For example, among the composite services that achieve the desired functionality, a user might prefer a more secure service over a less secure one; or one with a lower cost over one with a higher cost. Such preferences may be *quantitative* or *qualitative*. In many settings, a user might need to trade off one non-functional attribute against another (e.g., performance against cost); In others, it might be useful to assign relative importance to different non-functional attributes (e.g., security being more important than performance). Hence, there is an urgent need for principled methods that incorporate consideration of user-specified preferences with respect to the non-functional attributes, and the relative importance of the different non functional attributes. Of particular interest are algorithms that ensure that a set of solutions generated constitute a *non-dominated set*. We say that a set  $N$  of composite services is a *non-dominated set* if there is no composite service that is not in  $N$  that is strictly preferred over one or more of the composite services in  $N$  with respect to a set of user-specified preferences over non-functional attributes (and their relative importance).

Against this background, we present a procedure, **TCP-Compose\*** for generating, given (i) a set of functional specifications; (ii) a set of preferences with respect to non-functional attributes and their relative importance; (iii) a repository of candidate services with specified input-output behaviors and non-functional attributes; and (iv) *any* sound algorithm for assembling, from a repository of component services: a set of composite services that (a) achieve the desired functionality and (b) are non-dominated with respect to the user-specified preferences over non-functional attributes by any other composite service in the solution set of the algorithm used for functional specification based service composition.

**TCP-Compose\*** makes use of Tradeoff-enhanced Conditional Preference Network (TCP-net) [19], a variant of Conditional Preference Network [20], a framework for representing and reasoning with qualitative preferences. *CP-net* provides a compact representation of user-specified preferences with respect to non-functional attributes, by taking advantage of the independence or conditional independence of user preferences with respect to an attribute from preferences with respect to other attributes. *TCP-net* extends the *CP-net* framework by allowing the specification of the *relative importance* of different attributes (e.g., security is more important than cost).

**TCP-Compose\*** uses a heuristic estimate of the preference ordering of alternative partial solutions to a service composition problem that corresponds to different choices of each component service, to improve the efficiency of search for a set of non-dominated solutions. We establish the conditions under which the proposed algorithm is guaranteed to find a set of non-dominated compositions with respect to user-specified qualitative preferences over possible values of each non-functional attribute and the relative importance of different non-functional attributes.



**Fig. 1.** Goal Service

The rest of the paper is organized as follows: Section 2 introduces the problem of service composition from user-specified functional and non-functional specifications; Section 3 describes the key aspects of CP-net and TCP-net formalisms used for representing and reasoning about user preferences with respect to the non-functional attributes and their relative importance, and outlines the application of TCP-nets to guide service composition based on non-functional requirements; Section 4 describes the algorithm TCP-Compose\* and establishes the conditions under which TCP-Compose\* is guaranteed to find the set of non-dominated compositions with respect to user-specified qualitative preferences over possible values of each non-functional attribute and the relative importance of different non-functional attributes; Section 5 concludes with a summary, discussion of related work, and an outline of some directions for further research.

## 2 Problem Specification

We introduce the problem of service composition from user-specified functional and non-functional requirements using a simple example. Suppose a user is interested in assembling a goal service  $G$  shown in Fig. 1 from a repository of services  $R = \{b_1, b_2, p_1, p_2, p_3, s_1, s_2, q_1, q_2, q_3\}$ —where  $b_i$ 's are book buying services,  $s_i$ 's are shipment services and  $p_i$ 's and  $q_i$ 's are payment services that can work with  $b_i$ 's and  $s_i$ 's respectively. Suppose  $(p_3, q_2)$ ,  $(b_2, s_2)$ ,  $(b_2, q_2)$ ,  $(b_2, q_3)$  are functionally incompatible and hence cannot be used together in any valid composition. The goal service should allow the user to buy book(s) from an online store, pay the store through a credit card service, arrange for shipping the book through a shipment service and pay for the shipping. In Fig. 1, each of the steps in the goal service is annotated with the set of services from the repository that provide the respective functionality.

What we have so far is an informal specification of a service composition task based on user-specified functional requirements. We now turn to specification of user preferences with respect to three non-functional attributes: reliability, security, and availability of the goal service denoted by  $R$ ,  $S$ , and  $A$  respectively.



**Table 1.** Domain Definition

Preference Variable	Domain of Preference Variable
Reliability ( $R$ )	$\{L_R, H_R\}$
Security ( $S$ )	$\{L_S, M_S, H_S\}$
Availability ( $A$ )	$\{L_A, H_A\}$



**Fig. 2.** Example CP-net and TCP-net

Suppose the available services can have Low ( $L_R$ ) or High ( $H_R$ ) reliability; Low ( $L_S$ ), Medium ( $M_S$ ) or High ( $H_S$ ) security; and Low ( $L_A$ ) or High ( $H_A$ ) availability as shown in Table 1. Assume that the following non-functional attributes are known of each of the component services:  $b_1 : L_R, b_2 : H_R, p_1 : L_S, p_2 : M_S, p_3 : H_S, s_1 : L_A, s_2 : H_R, q_1 : L_S, q_2 : M_S, q_3 : H_S$ .

Now suppose that the user’s preference with respect to security level is *not* independent of the reliability of the service. Suppose further that when the reliability is low, the user prefers high security; and when reliability is high the user is willing to settle for lower security (say, because of the prohibitive cost of achieving both high security and reliability); Suppose further that the user prefers high availability to low availability irrespective of the reliability and security of the service. Such information can be represented concisely using a CP-net with three nodes denoting the three attributes  $R, S$ , and  $A$ . The single headed arrows (e.g., from  $R$  to  $S$ ) denote dependence among user preferences with respect to the attributes under consideration. The qualitative preferences of the user with respect to each attribute (conditioned on the preferences over attributes that such preference is dependent on) are specified by the conditional preference table (CPT) that annotate each node (Fig. 2(a)). Suppose further that the user attaches greater importance to availability relative to security. Such assertions of relative importance of one attribute over another are represented using double headed arrows in TCP-net shown in Fig. 2(b). The information regarding preferences with respect to  $R, S$ , and  $A$  in Fig. 2(b) are the same as those shown in Fig. 2(a).

Given the preferences with respect to the non-functional attributes and their relative importance, our task is to identify from the solution space, i.e., the set of composite services that satisfy the user-specified functional requirements, a subset that forms a *non-dominated* set with respect to a set of user-specified

preferences over non-functional attributes (and tradeoffs among them) that are captured by a TCP-net. It should be noted that a unique optimal composition exists only when the corresponding TCP-net induces a total ordering over the set of candidate feasible composite services, that is, the set of composite services that satisfy the user-specified functionality. TCP-Compose\* does not assume the existence of a total order induced by the TCP-net over user-specified preferences and relative importance among attributes. Instead, we return a set of feasible composite services that constitute a non-dominated set with respect to the TCP-net that reflects the users preferences and tradeoffs with respect to the non-functional attributes.

### 3 Representing Preferences Using CP-Nets and TCP-Nets

We first introduce the basic notions of preference relation, preferential independence under the *ceteris paribus*<sup>1</sup> semantics and the notion of relative importance among variables. We start with a set of preference variables  $V = \{X_1, \dots, X_n\}$  with finite domains  $D(X_1), \dots, D(X_n)$ .

An *outcome*  $o$  is a complete assignment of all variables  $X_i$  in  $V$ . The set of outcomes is  $O \subseteq D(X_1) \times D(X_2) \times \dots \times D(X_n)$ . A *preference ranking* is a total preorder over the set of outcomes  $O$ . We denote the fact that outcome  $o_1 \in O$  is *at least as preferred (strictly preferred)* to outcome  $o_2 \in O$  by  $o_1 \succeq o_2$  ( $o_1 \succ o_2$ ). We denote the fact that the user is *indifferent* between outcomes  $o_1$  and  $o_2$  by  $o_1 \cong o_2$  if neither  $o_1 \succeq o_2$  nor  $o_2 \succeq o_1$ .

**Preferential Independence.** In order to understand the need for preferential independence, we note that the set of possible outcomes is exponential in the number of preference variables  $n$  (where  $n = |V|$ ). Further, the set of possible total preorders is doubly exponential in  $n$ . A set of variables  $X \subseteq V$  is *preferentially independent* of  $Y = V - X$  if for all possible values of  $Y$ , the *preference order* among various assignments to  $X$  is the same. Formally, a set of variables  $X$  is *preferentially independent* of the set of variables  $Y = V - X$  iff for all  $x_1, x_2 \in D(X)$ ;  $y_1, y_2 \in D(Y)$  (where we use  $D(\cdot)$  to denote the domain of set of variables also), we have:  $x_1 y_1 \succeq x_2 y_1$  iff  $x_1 y_2 \succeq x_2 y_2$ . We say that  $x_1$  is preferred to  $x_2$  *ceteris paribus* (all else being equal).

**Conditional Preferential Independence.** Let  $X, Y, Z$  be a partition of  $V$  and let  $x_1, x_2 \in D(X)$ ;  $y_1, y_2 \in D(Y)$  and  $z \in D(Z)$ .  $X$  and  $Y$  are *conditionally preferentially independent* of each other given  $z$  iff,  $\forall x_1, x_2, y_1, y_2$  we have:  $x_1 y_1 z \succeq x_2 y_1 z$  iff  $x_1 y_2 z \succeq x_2 y_2 z$ .

**Relative Importance.** In Fig. 2(a), we observe that the variables availability and reliability are preferentially independent. Thus, the CP-net, does not assert whether an outcome with high availability and low reliability is preferred to one with low availability and high reliability: all we know from the CP-net is that higher availability and higher reliability are preferred. If we have the additional

<sup>1</sup> Ceteris paribus is a Latin phrase that means "all other things being equal".

information that although reliability and availability are preferentially independent, reliability is *more important* to the user than availability, we can infer that given a choice, the user would settle for lower availability instead of compromising on reliability. Formally, let  $X$  and  $Y$  be a pair of preferentially independent variables given  $V - \{X, Y\}$ . We say that  $X$  is *relatively more important* than  $Y$ , denoted by  $X \triangleright Y$ , if

$$\forall w. w \in D(W), \text{ where } W = V - \{X, Y\}, \forall x_1, x_2 \in D(X), \forall y_a, y_b \in D(Y) \\ x_1 \succ x_2 \Rightarrow x_1 y_a w \succ x_2 y_b w.$$

Note that the preference  $x_1 y_a w \succ x_2 y_b w$  holds even if  $y_b \succeq y_a$ , since any change for the worse in  $Y$  is preferred to any change for the worse in  $X$ . A conditional version of relative importance is defined analogously as follows. Let  $X$  and  $Y$  be a pair of preferentially independent variables given  $V - \{X, Y\}$  and  $z \in D(Z)$ . We say that  $X$  is *conditionally relatively more important* than  $Y$  given  $z$ , denoted by  $X \triangleright_z Y$ , if the following holds:

$$\forall w. w \in D(W), \text{ where } W = V - (\{X, Y\} \cup Z), Z \subseteq W \\ \forall x_1, x_2 \in D(X), \forall y_a, y_b \in D(Y) : (x_1 \succ x_2 \text{ given } zw) \Rightarrow x_1 y_a z w \succ x_2 y_b z w.$$

### 3.1 TCP-Nets

TCP-nets [21][19], extend the CP-net representation by incorporating the *relative importance* among pairs of attributes. The nodes of a TCP-net are the preference attributes  $V$ , and there are three types of edges. The first type of edge is a directed edge (single headed arrow) from  $X$  to  $Y$  used to model preferential dependence of  $Y$  on  $X$ . Such an edge asserts the preferential dependence of an attribute  $X_i$  on the assignment of its parents  $Pa(X_i)$ . Each node (preference attribute)  $X_i$  that has a non empty set of parents  $Pa(X_i)$  influencing its preferences is annotated with the conditional preference relation called conditional preference table  $CPT(X_i)$ . More formally, for each assignment of  $Pa(X_i)$ ,  $CPT(X_i)$  specifies a total order over  $D(X_i)$ . The second type of edge is a double headed arrow which captures the relative importance among a pair of attributes, i.e. if there is such an edge from  $X$  to  $Y$  then  $X$  is relatively more important than  $Y$ . The third type of edge is an undirected edge which captures the conditional relative importance between  $X$  and  $Y$  given  $Z$ . We refer to [19] for formal definitions of TCP-nets.

**Definition 1 (Completion).** [19] *The completion of a partial assignment  $z$  is defined as a complete assignment or an outcome consistent with  $z$ , denoted  $Comp(z)$ . By consistency, we mean that if a preference attribute  $X_i$  has a valuation  $v_i$  in  $z$  then the valuation of  $X_i$  is also  $v_i$  in  $Comp(z)$ .*

**Definition 2 (Most Preferred Completion).** [19] *The most preferred completion of a partial assignment  $z$ , denoted  $PrefComp(z, \mathcal{N})$  is defined as a completion of  $z$  that is preferentially optimal with respect to the TCP-Net  $\mathcal{N}$ , i.e.  $\nexists o \in O : o \succ PrefComp(z, \mathcal{N})$  such that  $o$  is a completion of  $z$  and consistent with  $z$ .*

## Remarks

1. We restrict our discussion to the class of *conditionally acyclic* TCP-nets that have been shown to be satisfiable with respect to a preference relation [19].
2. Given a *conditionally acyclic* TCP-net, it is possible to order the set of all outcomes  $O$  [19]. In other words, there exists a total order (that can be obtained using a topological sort) of the set of outcomes  $O$  that is *consistent with* the given TCP-net. However, several orderings of  $O$  can be consistent with a given conditionally acyclic TCP-net. For example, in a total preorder, there could be an outcome  $o$  such that  $\nexists o' \succ o$  with respect to  $\mathcal{N}$ , but one cannot define  $o$  as the unique most preferred outcome. In our example, considering tuples of valuations of the non-functional attributes of a service, if the user did not give the information that  $R$  is relatively more important than  $A$ , then we would not be able to assert a preference among compositions with outcomes  $o_1 = (H_R, L_S, H_A)$  and  $o_2 = (L_R, H_S, L_A)$  (where subscripts denote the corresponding non-functional attributes reliability ( $R$ ), security ( $S$ ) and availability ( $A$ )). In this case, the user may like the composition system to return both the compositions if both  $o_1$  and  $o_2$  are non-dominated, i.e.,  $\nexists o' \succ o_1$  and  $\nexists o' \succ o_2$ . The algorithm we present, TCP-Compose\* guarantees that in the absence of a *unique total order* over the outcomes, the outcome corresponding to each composition in the solution set is non-dominated by the outcome corresponding to any other feasible composition.
3. We also note that there is another variant of the TCP-net, known as UCP-nets [19] that capture quantitative preferences and relative importance information using utility functions. However, since we are not dealing with quantitative preferences, we stick to the basic qualitative TCP-nets.

## 3.2 Utilizing TCP-Nets in Web Service Composition

We now proceed to describe how TCP-nets can be used to model qualitative preferences during Web service composition. For this we will use *dominance queries* [19] of the form  $o \overset{?}{\succ} o'$  with respect to  $\mathcal{N}$  (in other words whether  $o$  is preferred to or dominates  $o'$ ). The problem of Web service composition is to assemble a composite service that achieves a desired functionality from a set of component services. More precisely, we have:

**Definition 3 (Web service composition problem).** *Given a target or goal service  $G$  and a repository of available services  $R = \{W_1, W_2 \dots W_n\}$ , Web service composition amounts to creating a set of composite services  $C = \{C_1, C_2 \dots C_m\}$  such that  $\forall i \leq m, C_i = W_{i_1} \oplus W_{i_2} \dots \oplus W_{i_k}$  and  $\forall l \leq i_k, W_l \in R$  such that  $C_i$  is functionally equivalent<sup>2</sup> to the  $G$ , denoted by  $C_i \equiv G$ . In the above,  $\oplus$  is the composition operator for composing two services.*

<sup>2</sup> Functional equivalence can be defined in many ways depending on the particular formalism used to describe the services. For example, if labeled transition systems are used for describing the services, checking the functional equivalence of a composite service to a goal service reduces to checking the *bisimulation equivalence* of the corresponding labeled transition systems [12][13].

Note that  $\oplus$  is a *generic* composition operator and  $W_{i_1}, W_{i_2} \dots W_{i_k}$  is an arbitrary ordering of the components in  $C_i$  such that  $W_{i_j}$  is selected before  $W_{i_{j+1}}$  in constructing  $C_i$ . We now proceed to describe an approach for using the TCP-net representation of user-specified *non-functional* requirements to guide service composition using any of the standard methods that can generate compositions that satisfy user-specified *functional* requirements.

## 4 TCP-Compose\*

We present an algorithm, TCP-Compose\*, that uses a preference guided heuristic to come up with the most preferred compositions among the candidates.

### 4.1 Search Space of TCP-Compose\*

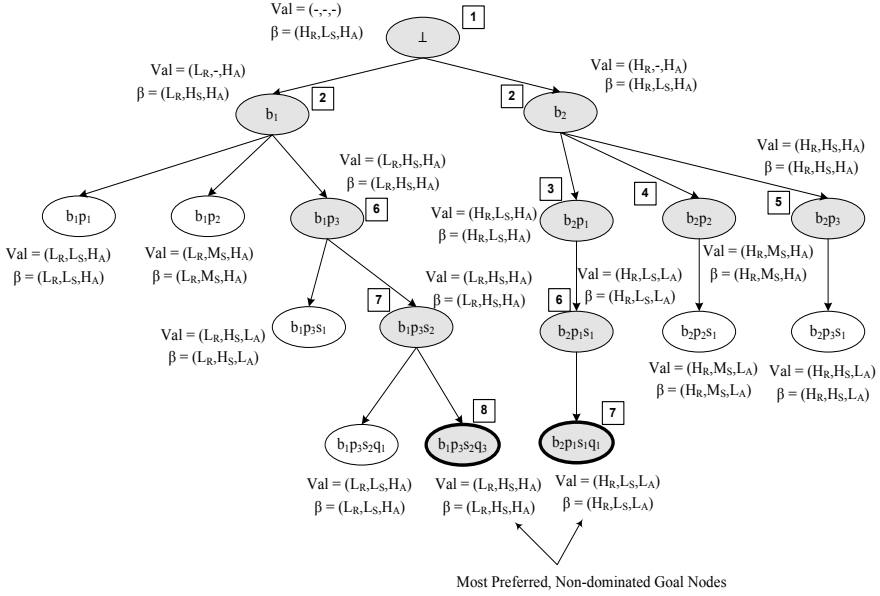
We cast the problem of assembling from a set of available component services, a composite service with the desired functionality as a state space search problem. The empty composition  $\perp$  is the start state; the set of *feasible extensions* using one of the available components from any given state define the successors of that state; and the set of feasible candidate compositions correspond to the goal states. The cost function at any state is given by the *preference valuation* of the partial composition corresponding to that state.

**Definition 4 (Feasible Extension).** A *feasible extension* to a partial composition  $P$  is defined as a partial composition  $P' = P \oplus W_i, W_i \in R$  such that the partial composition  $P'$  is functionally equivalent to a part of the goal service.

Let  $\mathcal{N}$  be a TCP-net with a set of preference attributes  $V = \{X_1, X_2, \dots, X_p\}$  with finite domains  $D(X_1), D(X_2), \dots, D(X_k)$  respectively where each preference attribute corresponds to a non-functional attribute of a composition. We assume that such a TCP-net specification is given by the user as input to the algorithm TCP-Compose\*.

Each of the leaf nodes is a goal node and corresponds to valid or feasible candidate composite services that are functionally equivalent to the goal service. Note that the the nodes of the tree may have varying but finite branching factors. Fig. 3 illustrates the search space for our goal service given in Fig. 1 with respect to the TCP-net given in Fig. 2. The shaded nodes correspond to partial compositions that were actually expanded further. The numbers in the boxes next the nodes show the order in which the corresponding nodes are expanded. The nodes that are not shaded are generated but not further pursued by the algorithm: For example, although TCP-Compose\* explores partial composition  $b_1 \oplus p_1$ , its feasible extension  $b_1 \oplus p_1 \oplus s_1$  is not explored. The annotation *Val* denotes the *preference valuation* and  $\beta$  denotes the *most preferred completion* of the partial composition corresponding to each node. They are formally defined below.

**Definition 5 (Preference Valuation).** Preference valuation is a function  $F : W \times X \rightarrow \bigcup(D(X_i) \cup \{-\})$ , where  $W = \{W_1, W_2 \dots W_n\}$ ,  $X = \bigcup X_i$ . The value  $\{-\}$  denotes that the valuation of the corresponding attribute is unknown. We



**Fig. 3.** Search Space for TCP-Compose\* when the TCP-net does not induce a total order over the set of valuations

denote the valuation of an attribute  $X_i$  in a Web service  $W$  as  $F(W)(X_i) = v_i, v_i \in D(X_i) \cup \{-\}$ . We define the valuation of an attribute  $X_i$  in a composition of two services  $W_i, W_j$  as  $F(W_i \oplus W_j)(X_i) = F(W_i)(X_i) \odot F(W_j)(X_i)$ , where

$$F(W_i)(X_p) \odot F(W_j)(X_p) = \begin{cases} F(W_j)(X_p), & F(W_i)(X_p) \succ F(W_j)(X_p) \\ F(W_i)(X_p), & \text{otherwise.} \end{cases}$$

The preference valuation of a partial composition  $P = W_1 \oplus W_2 \oplus \dots \oplus W_l$  with respect to an attribute  $X_p$  is defined inductively as  $F(P)(X_p) = F(W_1)(X_p) \odot F(W_2)(X_p) \dots \odot F(W_l)(X_p)$ . We also denote the preference valuation (over all attributes) of a partial composition  $P$  as the tuple  $Val(P) = (F(P)(X_1), F(P)(X_2), \dots, F(P)(X_k))$ .

Thus, the preference valuation of a partial composition with respect to a non-functional attribute corresponds to the least preferred valuation of that attribute among the participating component services in the composition. For example, in Fig. 3 the valuation of the partial composition  $b_2 \oplus p_1 \oplus s_1$  with respect to the attribute availability ( $A$ ) is low ( $L_A$ ) because the component  $s_1$  has low availability although the component  $b_2$  has high availability.

**Definition 6 (Most Preferred Completion).** The most preferred completion of a preference valuation of a partial composition  $P$  is defined as a complete assignment to all attributes  $X_1, X_2, \dots, X_k$ ,  $\beta(P) = PrefComp(Val(F(P)), \mathcal{N})$  where the function  $PrefComp$  is as defined in Def. 2.

**Algorithm 1.** TCP-Compose\* ( $\mathcal{N}, \varphi, G, R$ )

---

```

1. Compute heuristic  $\rho \leftarrow h(\varphi)$ 
2. for all  $P \in \rho$  do
3.   if ( $P \equiv G$  and  $\nexists Q \in \theta : \beta(Q) \succ \beta(P)$ ) then
4.      $\theta \leftarrow \theta \cup \{P\}$ 
5.      $\varphi \leftarrow \varphi - \{P\}$ 
6.     for all  $Q \in \theta$  do
7.       if  $\beta(P) \succ \beta(Q)$  then
8.          $\theta \leftarrow \theta - \{Q\}$ 
9.       end if
10.    end for
11.   else
12.     Find the next set of feasible partial compositions expanding  $P$ 
13.      $\psi \leftarrow \{P_i \mid P_i = P \oplus W_i, W_i \in R \text{ and } P \oplus W_i \equiv G\}$ 
14.      $\varphi \leftarrow \varphi \cup \psi - \{P\}$ 
15.   end if
16. end for
17. if ( $\varphi = \phi$ ) then
18.   return  $\theta$ 
19. end if

```

---

Intuitively, it is easy to see why  $\beta(P_i)$  is a heuristic estimate of the most preferred composition that can be realized by extending the given  $P_i$ . In our search for compositions,  $\beta(P_i)$  denotes the estimate of most preferred feasible candidate composite service that is equivalent to the goal service that is realizable from the current partial composition  $P_i$ .

**Definition 7 (Heuristic Function  $h$ ).** We define the heuristic function  $h$  as  $h : 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ , where  $\mathcal{P}$  is a set of partial compositions and  $S := h(\varphi)$ ,  $S \subseteq \varphi \subseteq \mathcal{P}$  is such that  $\forall P_0 \in S$  and  $\forall P_i \in \varphi$ ,  $\beta(P_i) \not\succeq \beta(P_0)$ . We also define  $h(\phi) = \perp$  and  $\perp \oplus W_i \equiv W_i$ .

The above definition of the heuristic function  $h$  makes it clear that, for any set of partial compositions  $\varphi$ ,  $h(\varphi)$  is the set of partial compositions whose valuations are non-dominated by the valuation of any other partial composition in  $\varphi$ .

Algorithm 1 shows the listing for TCP-Compose\*. The main idea behind TCP-Compose\* is to use a best first search strategy to find a set of non-dominated feasible candidate compositions equivalent to the goal service. To guide the search, the algorithm applies the heuristic function  $h$  to the set of partial compositions under consideration. The algorithm is initially invoked with the parameters  $(\mathcal{N}, \phi, G, R)$ . The set of partial compositions under consideration for expansion are maintained in  $\varphi$ , and the algorithm uses  $h(\varphi)$  to select the set of non-dominated compositions  $\rho$  to expand (line 1). Next, for each of the compositions in the non-dominated set  $\rho$ , if there is a candidate composition  $P$  which is functionally equivalent to the goal service, then the algorithm adds  $P$  to the solution set  $\theta$ , provided none of the existing solutions already in  $\theta$  strictly

dominate it (lines 2 - 5). If  $P$  now strictly dominates any of the existing solutions in  $\theta$  then those candidate solutions are removed from  $\theta$  (lines 6 - 10). For partial compositions that are not candidate compositions in  $\rho$ , the algorithm proceeds to compute the set of feasible extensions and adds them to  $\varphi$  (lines 11 and 14). The algorithm terminates with the set of candidate solutions  $\theta$  if there are no more compositions to explore (lines 16 - 18), and finally, the process is repeated (line 19) until there are no more compositions left to explore.

We now proceed to describe how the algorithm explores the search space when the TCP-net does not induce a total order on the set of non-functional attribute valuations. In the search space illustrated in Fig. 3 in the first run, when expanding the root node there are two possible partial compositions namely  $b_1$  and  $b_2$  respectively. Note that the valuations of partial compositions  $b_1, b_2$  are incomparable with respect to the TCP-net  $\mathcal{N}$ , and in the second run, both the partial compositions in  $\varphi$  are expanded. In runs 3, 4 and 5 the compositions  $b_2 \oplus p_1, b_2 \oplus p_2, b_2 \oplus p_3$  are expanded, as their valuations strictly dominated others in their respective iterations. However, in the sixth run, the algorithm again finds that the non-dominated compositions  $b_1 \oplus p_2$  and  $b_2 \oplus p_1 \oplus s_1$  are incomparable, and hence expands both. Notice that when expanding  $b_2 \oplus p_1 \oplus s_1$ , the feasible extensions also have the component  $s_1$  (we assumed that  $b_2, s_2$  are functionally incompatible and cannot be composed together) which has lower reliability and availability, but there is a still a possibility of a service with  $b_1$  ending up with a candidate composition with high availability. In the seventh run, the algorithm identifies  $b_2 \oplus p_1 \oplus s_1 \oplus q_1$  as a solution, and finally in the eighth run the algorithm terminates with both the candidate compositions  $b_1 \oplus p_3 \oplus s_2 \oplus q_3$  and  $b_2 \oplus p_1 \oplus s_1 \oplus q_1$  as the non-dominated candidate compositions. This illustrates how the less preferred compositions like  $b_1 \oplus p_1$  are actually not explored by the algorithm, consequently pruning of the search space.

## 4.2 Properties of TCP-Compose\*

We show that the algorithm TCP-Compose\* is guaranteed to return the set of composite services each of which is functionally equivalent to the user-specified goal service that constitute a non-dominated set with respect to a set of user preferences over the non-functional attributes.

**Lemma 1.** *For any partial composition  $P$ ,  $\beta(P) \succeq \beta^*(P)$ , where  $\beta^*$  gives the valuation of the actual most preferred feasible composition starting with  $P$ .*

*Proof.* Suppose by contradiction, there exists a partial composition  $P$  and a  $\beta^*$  such that  $\beta^*(P) \succ \beta(P)$ . This implies that there is a feasible candidate composition  $C$  starting from the partial composition  $P$  such that  $Val(C) \succ \beta(P)$ , or there is a sequence of feasible extensions from  $P$  to  $C$  such that  $Val(C) \succ PrefComp(Val(P), \mathcal{N})$  with respect to  $\mathcal{N}$ , by the definition of  $\beta(P)$ . This clearly contradicts the definition of  $PrefComp(Val(P), \mathcal{N})$ .  $\square$

**Theorem 1.** *TCP-Compose\* is guaranteed to return the set of feasible composite services that constitute a non-dominated set with respect to a given TCP-net.*



*Proof (Sketch, by contradiction)*

Suppose TCP-Compose\* does not terminate with the set of non-dominated candidate compositions. There are three cases to consider.

1. TCP-Compose\* terminates with a set of compositions such that one of the solutions returned by TCP-Compose\* is a not feasible composition. This contradicts the Step 3 of the algorithm where the terminating condition is clearly only satisfied for feasible compositions.
2. TCP-Compose\* fails to terminate. This is not possible because although the algorithm is recursive, the search tree is finite, and in each iteration, previously examined partial compositions are not revisited.
3. Starting with a partial composition  $P$ , TCP-Compose\* terminates with a set of candidate compositions such that one of the solutions corresponds to a feasible candidate composition  $C'$  with a sub-optimal preference valuation  $Val(C')$ , i.e.  $\beta^*(P) \succ Val(C')$ , where  $\beta^*$  gives the *actual* most preferred feasible composition starting with  $P$ .

By Lemma 1, at each step,  $\beta(P) \succeq \beta^*(P) \succ Val(C') \Rightarrow \beta(P) \succ Val(C')$ . So in the last step just before termination, by the definition of the heuristic function  $h$  and Line 1 of TCP-Compose\*, the algorithm would have chosen to expand the composition  $P$  rather than the partial composition that just preceded  $C'$ . Hence, the algorithm could not have terminated with any composition  $C'$  such that  $\beta^*(P) \succ Val(C')$ , and hence it would return only non-dominated candidate compositions.

This proves that TCP-Compose\* is guaranteed to return the set of feasible composite services that constitute a non-dominated set with respect to a given TCP-net.  $\square$

## 5 Summary and Discussion

Most of the work on service composition has focused on algorithms for assembling, from a set of available component services, a composite service that achieves the user-specified functionality. However, in many applications, preferences over non-functional attributes e.g., availability, performance as well as tradeoffs among them can influence the choice of the component services in assembling a composite service that achieves the desired functionality. Hence, there is a growing interest in techniques that incorporate such non-functional considerations into service composition. For example, Zeng et al. [45] have explored linear programming methods for optimizing the choice of services based on non-functional attributes based on user-specified weights and *utility functions*. Yu et al. [6] have explored a formulation of service composition as a combinatorial optimization (multi-choice multi-dimensional 0-1 Knapsack problem) and as a graph search problem wherein the non-functional constraints are encoded by the edges in the graph. Berbner et al. [7] have proposed a heuristic approach, using simulated annealing and integer programming, for selecting services based on non-functional attributes. Each of these approaches assume a *quantitative*

measure of preference over alternative valuations of non-functional attributes. This is tantamount to assuming the existence of a *cardinal* utility function [22]. Eliciting such a utility function, over multiple not necessarily independent attributes, from users presents a significant challenge in practice. Hence, methods that can utilize *qualitative* information regarding preferences over non-functional attributes are of significant interest.

Against this background, we have presented TCP-Compose\*, a procedure for generating, given (i) a set of functional specifications; (ii) a set of preferences with respect to non-functional attributes and their relative importance; (iii) a repository of candidate services with specified input-output behaviors and non-functional attributes; and (iv) *any* sound algorithm for assembling, from a repository of component services: a set of composite services that (a) achieve the desired functionality and (b) are non-dominated with respect to the user-specified preferences over non-functional attributes by any other composite service in the solution set of the algorithm used for functional specification based service composition. TCP-Compose\* uses TCP-net, a graphical modeling paradigm for representing and reasoning with qualitative preferences and importance of alternative partial solutions to a service composition problem that corresponds to different choices of each component service. An important feature of TCP-Compose\* is that it offers a generic approach to augment *any* of a broad range of available algorithms for assembling feasible composite services that achieve the user-specified functionality with the ability to consider qualitative preferences and tradeoffs over non-functional attributes of the desired goal service.

Schropfer et al. [23] have recently proposed a TCP-net based formulation of qualitative user preferences over non-functional attributes for ranking and selecting *individual* services. In contrast, the focus of TCP-Compose\* is on the assembly of a set of *composite* services that constitute a non-dominated set with respect to a set of user-specified preferences and tradeoffs over non-functional attributes.

Shaparau et al. [11] have proposed an algorithm for contingent planning with goal preferences which can also be used for service composition. The planning algorithm requires the user to specify *explicit* preferences over goals. This is tantamount to explicitly specifying an ordering over all feasible composite services. Hence, this approach is likely to be impractical in settings where the number of component services in the repository is large. In contrast, the approach used in TCP-Compose\* requires the user to specify only the preferences and tradeoffs over the non-functional attributes that in turn *induce* a preference over the feasible composite services.

Work in progress is aimed at the implementation and experimental evaluation of TCP-Compose\* on a range of benchmark problems of varying complexity. Some interesting directions for further research include: investigation of approaches for handling of *global* non-functional constraints (e.g., no composite service which has security level below a specified threshold is acceptable); customized versions of TCP-Compose\* that take advantage of specific representations and algorithms used in the search for feasible solutions.

## References

1. Bichler, M., Lin, K.J.: Service-oriented computing. *Computer* 39(3), 99–101 (2006)
2. Papazoglou, M.: Service-oriented computing: concepts, characteristics and directions. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pp. 3–12. IEEE Computer Society, Los Alamitos (2003)
3. Huhns, M.P., Singh, M.P.: Service-oriented computing: Key concepts and principles. *Internet Computing* 9(1), 75–81 (2005)
4. Zeng, L., Benatallah, B., Dumas, M., Kalaganam, J., Sheng, Q.Z.: Quality driven web services composition. In: *Proceedings of the 12th international conference on World Wide Web*, pp. 411–421. ACM, New York (2003)
5. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalaganam, J., Chang, H.: Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering* 30(5), 311–327 (2004)
6. Yu, T., Lin, K.J.: Service selection algorithms for composing complex services with multiple qos constraints. In: Benatallah, B., Casati, F., Traverso, P. (eds.) *ICSOC 2005*. LNCS, vol. 3826, pp. 130–143. Springer, Heidelberg (2005)
7. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for qos-aware web service composition. In: *Proceedings of the IEEE International Conference on Web Services - ICWS 2006*, pp. 72–82 (2006)
8. Pistore, M., Traverso, P., Bertoli, P.: Automated composition of web services by planning in asynchronous domains. In: *Fifteenth International Conference on Automated Planning and Scheduling*, p. 211 (2005)
9. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) *ISWC 2004*. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
10. Sirin, E., Parsia, P., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. *Journal of Web Semantics* 1(4), 377–396 (2004)
11. Shaparau, D., Pistore, M., Traverso, P.: Contingent planning with goal preferences. In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence*. AAAI Press, Menlo Park (2006)
12. Pathak, J., Basu, S., Lutz, R., Honavar, V.: Selecting and composing web services through iterative reformulation of functional specifications. In: *Proceedings of the 18th International Conference on Tools with Artificial Intelligence*, pp. 445–454. IEEE Computer Society, Los Alamitos (2006)
13. Pathak, J., Basu, S., Honavar, V.: On context-specific substitutability of web services. In: *Proceedings of the International Conference on Web Services*, pp. 192–199. IEEE Computer Society, Los Alamitos (2007)
14. Pathak, J., Basu, S., Honavar, V.: Composing web services through automatic reformulation of service specifications. In: *Proceedings of the 5th IEEE International Conference on Services Computing* (to appear, 2008)
15. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: *Proceedings of the 14th Australasian database conference*, pp. 191–200. Australian Computer Society, Inc. (2003)
16. Dustdar, S., Schreiner, W.: A survey on web services composition. *International Journal on Web and Grid Services* 1(1), 1–20 (2005)
17. Pathak, J., Basu, S., Honavar, V.: Assembling composite web services from autonomous components. In: Maglogiannis, I., Karpouzis, K., Soldatos, J. (eds.) *Emerging Artificial Intelligence Applications in Computer Engineering*. IOS Press, Amsterdam (in press, 2008)

18. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: Nineteenth International Joint Conference on Artificial Intelligence, pp. 1252–1259 (2005)
19. Brafman, R.I., Domshlak, C., Shimony, S.E.: On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research* 25, 389–424 (2006)
20. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21, 135–191 (2004)
21. Brafman, R.I., Domshlak, C., Shimony, S.E.: Introducing variable importance tradeoffs into cp-nets. In: *Proceedings of Uncertainty in Artificial Intelligence*, pp. 69–76 (2002)
22. Keeney, R.L., Raiffa, H.: *Decisions with multiple objectives: Preferences and value trade-offs* (1993)
23. Schropfer, C., Binshtok, M., Shimony, S.E., Dayan, A., Brafman, R., Offermann, P., Holschke, O.: Introducing preferences over nfps into service selection in soa. In: *Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop* (2007)

# A Runtime Quality Architecture for Service-Oriented Systems

Daniel Robinson and Gerald Kotonya

Computing Department, Lancaster University, Lancaster, LA1 4WA, UK  
{robinsdb, gerald}@comp.lancs.ac.uk

**Abstract.** System quality aspects such as dependability, adaptability to a changing runtime environment, and concerns such as cost and provider reputation, are increasingly important in a competitive software service market. Service-oriented system quality is not just a function of the quality of a provided service, but the interdependencies between services, the resource constraints of the runtime environment and network outages. This makes it difficult to anticipate how these factors might influence system behaviour, making it difficult to specify the right system environment in advance. Current quality management schemes for service-oriented systems are inadequate for ensuring runtime system quality as they focus on static service properties, rather than emergent properties. They also offer the consumer only limited control over the quality of service. This paper describes a novel consumer-centred runtime architecture that combines service monitoring, negotiation, forecasting and vendor reputation, to provide a self-managing mechanism for ensuring runtime quality in service-oriented systems.

**Keywords:** Service-Oriented Architecture, Negotiation, Monitoring, Quality of Service, Software Composition.

## 1 Introduction

Service-oriented architectures support dynamic composition and reconfiguration of software systems by making advertised functionality and behaviour available on an “as-needed” basis [1]. This model of software deployment offers significant benefits over the traditional model of software deployment as a product, including reduced capital investment, dynamic integration and rapid deployment of platform and network-independent systems [2,3]. However, as the nature of service-oriented applications continues to vary and the demands on them grow, features such as dependability, adaptability to a changing runtime environment, and concerns such as cost and provider reputation are becoming increasingly important consumer quality considerations.

Current service quality management schemes are largely concerned with predicting system properties based on the static properties of its components [4]. However, the dynamic nature of a system composed from services requires a dynamic runtime approach which is able to detect and respond to emergent problems in the service execution environment, and to the problems that may

arise as a result of different services being composed together. Secondly, current quality schemes offer the consumer only limited control over the quality of a service and therefore the system. In summary, the current software service quality frameworks offer the consumer:

- *Limited consumer control over service quality.* The third-party nature of software services means that a consumer has little control over the quality of services outside the static service level agreement (SLA). SLAs are intended to define the scope, level, and quality of an externally provided service together with associated responsibilities. However, they are difficult to enforce, hard to integrate with specific consumer quality strategies and provide no obvious way of ensuring runtime system quality.
- *Poor support for runtime quality.* Whilst there are initiatives for monitoring and reporting service quality failings [5,6], monitoring alone is inadequate for ensuring runtime quality. To ensure runtime quality, monitoring must be supported with effective (re)negotiation and recovery strategies.
- *Limited support for customisation.* Current quality assurance approaches for service-oriented systems are restricted to specific quality assurance schemes, limiting their scope for experimentation and customisation (i.e. variability in quality contexts).
- *Poor support for resource-restricted systems.* Quality assurance is particularly challenging for systems that operate in resource-restricted environments [7]. Not only must a service have an acceptable level of quality; it must be possible to integrate and orchestrate it within the constraints of the runtime environment.

We have developed a self-configuring quality framework that uses an adaptable service brokerage architecture, to integrate consumer strategies with monitoring, (re)negotiation, forecasting and provider reputation as a means for ensuring runtime quality. The rest of this paper is organised as follows: Section 2 reviews current approaches for addressing quality in service-oriented systems. Section 3 describes the architecture of our quality framework. Section 4 describes service strategy formulation and management. Section 5 uses a small case study to illustrate the framework. Section 6 reviews the framework and provides some conclusions.

## 2 Background

Service-oriented systems are distributed and composed from numerous services which can be discovered and replaced at runtime. It is possible for several service providers to offer services with common functionality, but with different non-functional qualities. Qualities can be considered as constraints over the functionality of a service [8].

A characteristic of distributed systems is the volatility of service quality [9]. It is therefore important that mechanisms are in place for managing the overall

system quality [10]. Traditionally, quality of service (QoS) has been associated with telephony and computer networking, specifying requirements on the data flowing across the network (such as latency, jitter, number of dropped packets etc.). To ensure quality in service-oriented systems, application-level QoS must also be considered [11].

## 2.1 Service Description, Discovery and Selection

There are several initiatives to improve the characterisation of services by including non-functional aspects in their description. These include semantic approaches, such as the Web Service Modeling Ontology (WSMO) [9] and Ontology Web Language for Services (OWL-S) [12], and non-semantic approaches such as WS-Agreement [13].

WSMO and OWL-S both share a similar goal, which is to aid the automation of service discovery, selection, composition, substitution, and invocation through richer semantics [14]. WS-Agreement is a Web service protocol used in industry for establishing an agreement between a service provider and consumer.

When integrated with the service discovery process [12], these initiatives enable service providers to differentiate themselves from other providers of similar services. Service consumers are then able to discover and select providers that best satisfy their non-functional requirements. However, such initiatives are limited if there are no services which satisfy consumer requirements, as consumers must either select the closest match or go without service. Consumers are also required to trust that providers will provide services as advertised.

## 2.2 Service Reputation Systems

Reputation systems, such as feedback mechanisms used by online auction sites, are designed to address issues of trust between parties who have not dealt with one another before. Reputation systems can be used to help manage quality in service-oriented systems, by helping to distinguish between low and high quality service providers [15].

A reputation-based approach to service selection is described in [16] which uses software agents that share QoS information with one another, based on their interactions with the services they are attached to. Initially, each provider has the same (or no) reputation. Over time, poor service providers develop a poor reputation which makes them less likely to be selected for use by the agents. A reputation-enhanced service discovery protocol is discussed in [17]. This enables service consumers to consider QoS issues when making service selection decisions, with fewer assumptions about the trustworthiness and reliability of providers.

Reputation systems enhance service discovery and selection processes, by incorporating feedback on providers as part of the service selection criteria. This requires consumers to expend valuable resources auditing the received QoS of consumed services, and then providing feedback to a reputation system.

Reputation systems are also limited when there are no services which satisfy consumer requirements.

### 2.3 Service Negotiation

Service negotiation can bring software composed from services closer to meeting consumer requirements, through the formation of SLAs between service providers and consumers. Service providers can also benefit from negotiation, by utilising spare resources to provide a better QoS to those consumers who are prepared to pay an additional cost.

The negotiation of SLAs has been an active research area in the Web service community for several years. WS-Agreement [13] enables the specification of an agreement between a service provider and consumer, and provides a protocol for the creation of an agreement using agreement templates. The Web Service Level Agreement (WSLA) [18] is a similar initiative for defining SLAs, and describes how SLAs may be monitored for compliance. SLA negotiation has also seen considerable interest in the agent [16,19] and grid [20] communities.

Current initiatives are primarily concentrated on the negotiation of single services, and have not focused much on the negotiation of end-to-end QoS constraints [19]. Current initiatives also lack the ability themselves to effectively monitor service agreements for compliance.

### 2.4 Service Monitoring

Monitors are required to determine if services actually meet the terms and conditions agreed between service consumers and providers [21]. Monitors are also used to detect emergent properties that arise as a consequence of services interacting with each other through composition. Service providers can also impose conditions of use upon a service consumer, which may be monitored for compliance. The motivation for monitoring is to enable the quality management of services and service compositions, in response to problems such as networking issues, changes in the environment and emergent system properties.

Monitoring approaches used in service-oriented systems include: open and closed-loop control systems [6], assertion-based techniques [22] and approaches using late-binding and reflection [5]. Open and closed-loop techniques are used by service providers to stabilise service-oriented software, by collecting runtime information on services and feeding it to service controllers. In assertion-based approaches, pre-conditions and post-conditions are asserted on services and their non-functional properties, such as business processes, communication protocol preferences, organisational licensing and authentication.

Current initiatives to monitoring are largely manual activities. For example, service compositions specified as BPEL processes and annotated by a system designer with comments describing the monitoring to be performed [22], make it difficult to support quality management in a meaningful way. Such approaches are limited in handling problematic services, and do not support advanced techniques such as service renegotiation.



### 3 Quality Architecture

Fig. 1 shows the architecture of our proposed quality framework. The framework has been developed using the Jini<sup>1</sup> SOA, but is flexible enough to be applied to other SOAs such as Web services. This flexibility is achieved through implementation-specific connection interfaces. The Jini SOA was chosen primarily for its service discovery mechanism, relatively small footprint, and to facilitate the evaluation of the framework in resource-constrained environments.

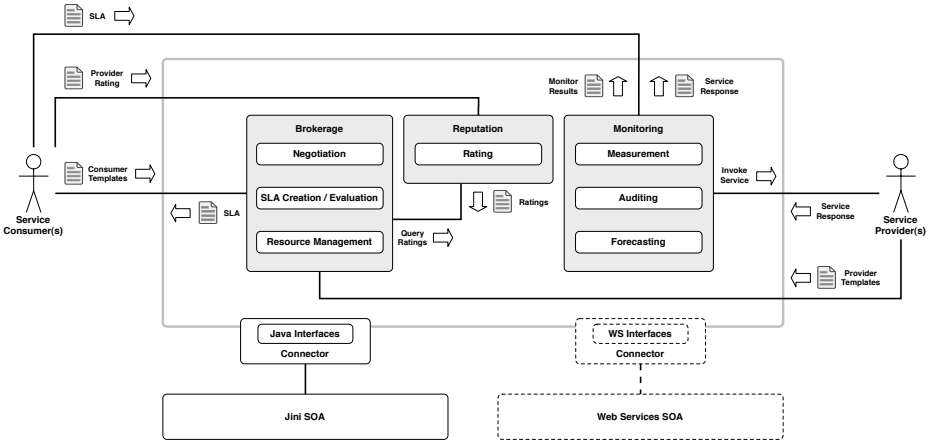


Fig. 1. Framework overview

The quality framework comprises mechanisms for discovering, brokering, monitoring and rating services and their providers (see Fig. 1). The next sections discuss each of these in turn.

#### 3.1 Brokerage Architecture

Existing brokerage models [23,19] focus on particular methods of negotiation, and are not engineered to be integrated with monitoring and reputation processes. We have developed a brokerage approach which provides a structural framework for integrating different methods of negotiation, monitoring and reputation, and supporting the requirements of automated service negotiation and renegotiation in SOA. Our brokerage model, shown in Fig. 2, is based on a factory architecture which creates individual brokers for service consumers and providers on demand.

A service provider uses the service discovery mechanism to locate a brokerage service provider, which in turn supplies it with a broker. The service provider supplies the broker with templates describing the negotiation models to use, decision algorithms and strategies for creating and evaluating proposals. These

<sup>1</sup> Jini – Sun Microsystems SOA: <http://www.jini.org/>

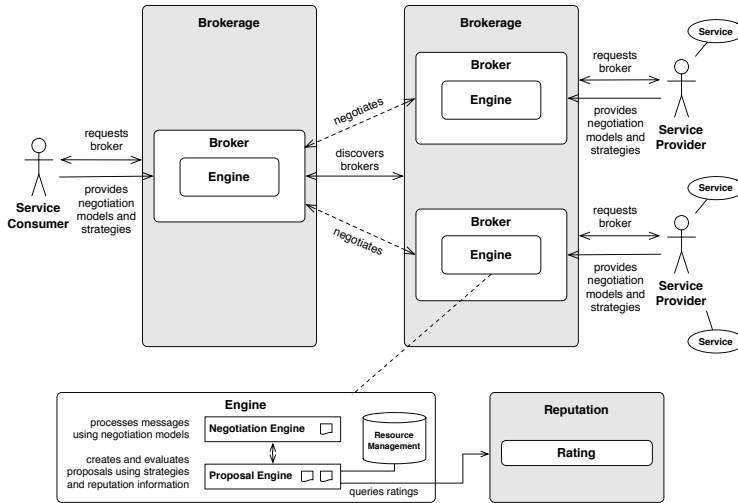


Fig. 2. Service brokerage architecture

templates are provided to an engine builder interface, which provides the broker with an engine for processing negotiation messages and service proposals. Providers also provide additional information enabling their brokers to perform service resource management on their behalf. For most types of negotiation, provider brokers enter a *passive* waiting state until they receive negotiation requests from consumer brokers.

Consumers locate service brokers similarly to service providers. However, there is no service resource management performed on the consumer side. Once initialised, consumer brokers typically enter an *active* state and use the service discovery mechanism to seek out brokerages that contain brokers of the service types required by the consumer, up to a specified limit. There are many different models of negotiation and types of negotiation decision algorithms. Fixed-pricing, auction, reverse auction and bargaining negotiation models are identified in [24]. Barter/bargaining models, request for quotes (RFQs) and auctions are identified in [25].

The framework architecture is pluggable, enabling a variety of negotiation models, decision algorithms and proposal strategies to be used. The framework currently supports two negotiation models. The first is a fixed-price negotiation model (cf. catalogue shopping) with a decision algorithm that accepts only if all qualities are within their respective range as specified by the strategy. The second type is a bargaining model based on static strategies. With the bargaining model, consumer brokers negotiate a single quality at a time. The counter proposal from the provider broker contains not only its offer for that quality, but offers for any other qualities which are related to that quality. To avoid deadlock, consumer brokers must determine any other qualities which have changed since the last proposal, and agree not to negotiate them later in the session. Once the consumer broker has finished negotiating with the set of discovered provider

brokers for each service, each possible composition is ranked, and the service proposals for the most acceptable composition are accepted. The remaining proposals are rejected but are recorded in a negotiation cache. The negotiation cache enables brokers to record the negotiation behaviour and proposals provided by other brokers they have previously interacted with. The negotiation process is initiated and led by the service consumer (see activity diagram in Fig. 3).

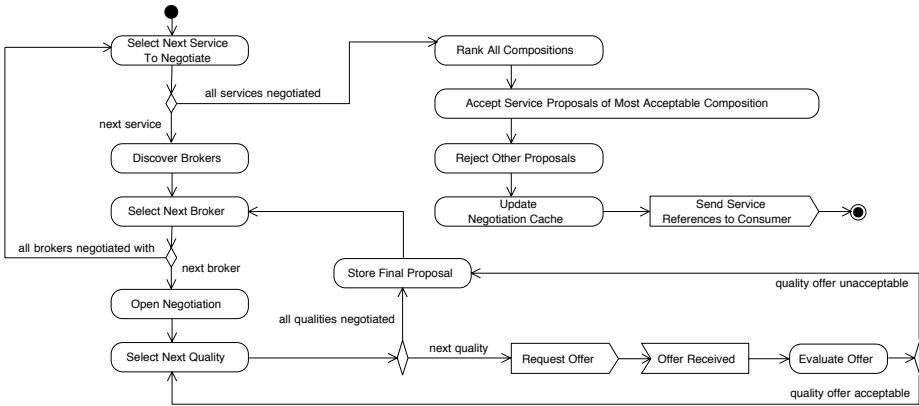


Fig. 3. Negotiation process implemented by engine in Fig. 2

Brokers share a common negotiation protocol for exchanging service proposals. The negotiation protocol currently supported by the framework is based on the primitives and protocol described in [24].

### 3.2 Monitoring Process

The framework actively monitors the quality of negotiated services for violations and failings at runtime. Changes in service quality are continuously evaluated against system composition acceptability levels, and an early renegotiation and replacement automatically initiated for failing services. Monitors are implemented as dynamic proxies (cf. decorator pattern), allowing for the creation of monitors at runtime which transparently intercept requests and responses between consumers and providers.

We have adopted a passive monitoring mechanism, which has the advantage that no additional load is placed on the consumer or provider of a service. In addition, the provider cannot differentiate between consumer and monitor requests (see Fig. 4). The monitoring service also provides a mechanism for auditing data collected by another party. The advantage with this approach is that the consumer does not have to expend additional resources performing auditing. However, the consumer expends additional resources in collecting data and providing it to the auditing service. Another provided approach enables a service to be monitored independently (cf. probed) from the service consumer. This provides an advantage for the consumer, but places additional load on the

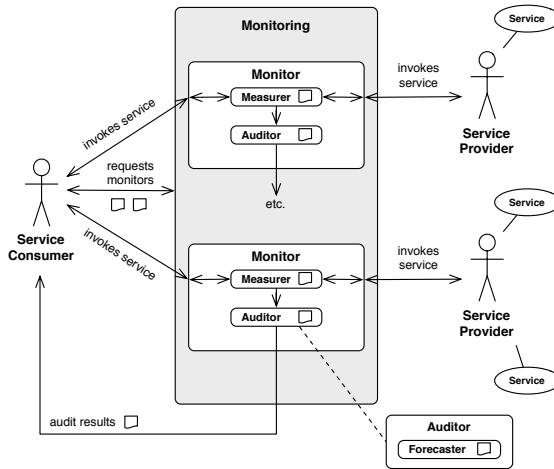


Fig. 4. Service monitoring architecture

provider. Furthermore, the provider may be able to distinguish monitor requests from consumer requests, and respond to each differently.

Auditors compare measured service qualities to those specified in the service contract. If a measured quality does not conform to its contracted value and constraints, the audit signals a *failed* quality contract violation. When consumers are informed of detected problems, they instruct their broker to renegotiate a new contract. If renegotiation fails, brokers attempt to secure service from an alternate provider (see Fig. 5).

The auditing data provided to the consumer has an overall result. If a pre-invocation service audit has passed, the consumer informs the monitor to invoke the service. If it has failed, the consumer can elect not to invoke the service and request its broker to renegotiate the problematic service. If a post-invocation service audit has passed, the consumer informs the monitor to continue monitoring the service. If it has failed, the consumer can again request its broker to renegotiate. When requesting renegotiation, the consumer provides its broker with a service contract created from the auditing data, which forms the basis for any renegotiation attempt.

When renegotiating a failed quality, the consumer broker assumes the provider broker is unable to guarantee a value better than the value which caused the audit to fail. Instead, the consumer broker expects some offer of improvement in another service quality or qualities. Improvements in other qualities should raise the overall acceptability of the renegotiated service to a more acceptable level. The consumer broker compares the acceptability of the renegotiated service proposal, with the proposals made by any other provider brokers in its negotiation cache. If the renegotiated service proposal is still the most acceptable, the consumer broker accepts the renegotiated proposal and continues using the service.

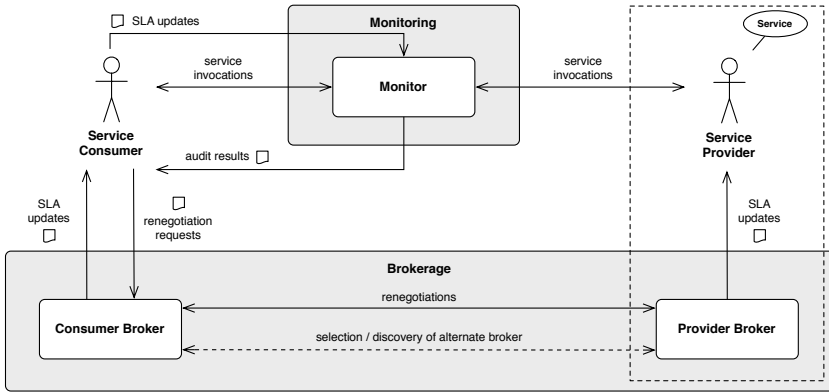


Fig. 5. Monitoring and renegotiation

If the renegotiated service proposal is no longer the most acceptable, the consumer broker attempts to gain service from an alternate provider, and rejects the renegotiated service proposal if successful.

Forecasting is an additional process which complements the auditing performed by service monitors. Service consumers specify the type of forecasting model to be used, and any additional parameters and values. During the audit, trends in measured service qualities are forecast. This enables the auditor to estimate in advance when a given service quality is about to fail. If a particular quality is estimated to fail, the audit signals a *failing* quality. If no contract violation is detected, and no problems are forecast, the audit signals an acceptable service. The framework currently provides forecasting models based on moving averages and exponential smoothing (such forecasting techniques are discussed in [26]).

### 3.3 Reputation Process

The reputation service provides a method for service consumers to rate the services of providers they have used. A service instance is rated once by a consumer, and is done once the service has been unleased. Services are unleased when either a contract is violated, or the service lease expires. All ratings received for the same provider and service type are combined, to develop the overall reputation for the provider’s ability to supply that particular service type in accordance with negotiated contracts.

The reputation service provides a query method to determine the overall rating of a provider for a given service type. Consumer brokers use this method to limit negotiation sessions to those providers which have an acceptable level of reputation (as defined in the strategy provided by the consumer). If a consumer has previously used the service(s) of a particular provider, the consumer’s own rating is combined with the global rating provided by the reputation service, according to weights specified in the consumer’s strategy. Provider brokers also

query for any rating a consumer may have given its provider in the past, before agreeing to provide a service to the consumer.

## 4 Service Strategy and Management

We have implemented strategy templates, based on a quality ontology which enables services to be described and negotiated in terms of their non-functional qualities and constraints. Non-functional attributes are specified using metadata interfaces, which enable a *design by contract* [27] approach.

Both consumer and provider strategies include three attributes which describe how any available reputation information should be used. The *reputation threshold* is used by consumer brokers to limit negotiation to those providers who have a level of reputation above a certain value. The threshold is also used by provider brokers, to limit negotiation to those consumers who have previously rated the provider above a certain value. The other two attributes are *proposal weight* and *reputation weight*, which are used when computing the overall acceptability of an offer. These attributes respectively indicate the importance of a service proposal, when compared to the reputation of the consumer or provider which made the proposal. Consumer strategies include two further attributes, *personal experience* and *global experience*, to weight the consumer's own experience of a particular provider and service against the experience provided by other consumers.

### 4.1 Service Acceptability

Each quality, operation and service in a strategy template is given a weighting from 0.0 to 1.0, so that the sum of all service- and operation-level qualities is 1.0 (the ideal QoS). The acceptability of a single quality proposal  $Q_a$  is calculated using the following formula, based on the acceptability formula given in [25], but extended to factor in the weight of a single quality  $Q_w$  as it pertains to an overall QoS. Let  $Q_p$  be the value proposed for the quality,  $Q_l$  the least acceptable value for that quality and  $Q_m$  the most acceptable value for that quality.

$$Q_a = \left| \frac{Q_p - Q_l}{Q_m - Q_l} \right| * Q_w$$

The formula is used for proposed values which fall within the range of acceptable values defined by the least and most acceptable values. Whether these are high or low values depends on whether greater or lesser values are more acceptable or less acceptable e.g. for a response time quality, a greater number may be less acceptable and a smaller number more acceptable. This equation is extended if reputation information is available for the creator of the proposal. If there exists any prior personal experience of the proposal creator, the reputation  $R$  is computed from both the global rating  $R_g$  provided by the reputation service, and the local personal experience  $R_l$ . Let  $G_w$  be the weight assigned to global experience and  $L_w$  be the weight assigned to local personal experience,

so that  $0 \leq G_w \leq 1$  and  $0 \leq L_w \leq 1$ , and  $G_w + L_w = 1.0$ . The overall rating  $R$  is then defined as:

$$R = (R_g * G_w) + (R_l * L_w)$$

The total quality acceptability  $Q_{ta}$  is then calculated as follows. Let  $P_w$  be the weight of the proposal and  $R_w$  the weight of the reputation information, so that  $0 \leq P_w \leq 1$  and  $0 \leq R_w \leq 1$ , and  $G_w + P_w = 1.0$ .

$$Q_{ta} = (Q_a * P_w) + (R * R_w)$$

## 4.2 Service Composition

The framework is capable of negotiating a composition, but is not responsible for the actual composing of services (this activity is left to the service consumer). Each service in the required composition is individually-weighted, enabling the specification of compositions where one service is more critical than another. Calculating the acceptability of every possible composition is an NP-hard problem. For example, if a composition with 3 different services is required and there are 10 providers for each type of service,  $10^3$  composition comparisons are required in order to calculate the acceptability of every possible composition. Provider reputation can be used to limit negotiation to a subset of the available providers, but the linear programming approach still does not scale. Every additional service in a composition introduces another order of magnitude to the number of compositions which must be compared. There are several proposed solutions to this optimisation problem [23].

## 5 Case Study

We have developed a small case study to evaluate the framework and to visualise the framework processes and system quality at runtime. Simulated consumer devices, each with different resource constraints and requirements, execute a navigation application comprised of location, traffic, weather, street maps, and information services.

First, the system invokes the location service to obtain the location of the consumer. The location data is passed on to the maps, weather and traffic services to obtain graphical maps, weather and traffic information within an  $n$  metre radius of the consumer. Traffic and map information are integrated to highlight traffic conditions on the roads. The map information can also be used to request information on places of interest (e.g. restaurants, shops, parking, tourist attractions etc.). The consumers of this application each simulate a navigation device, such as a mobile phone, internet tablet, or automobile navigation system. Variances in consumer requirements mean that different providers are more acceptable to different consumers. In addition, the runtime environments of the different consumers vary from remote locations to busy metropolitan areas, meaning that invoking services will result in widely different responses.

Several providers of each service type are registered with the Jini discovery service. Each provider offers services with arbitrary differences in levels of QoS and cost, making some providers more acceptable than others for the different consumer devices. Once SLAs have been negotiated for each service type in the navigation composition, monitors are attached and each service in the navigation process is invoked as required.

Providers are doped in a variety of ways, so that they occasionally violate negotiated SLA qualities. Monitors detect these violations by auditing service invocations and forecasting trends. Once notified of a violation, consumers instruct their brokers to renegotiate the SLA if the monitored QoS is still within acceptable limits. If the monitored QoS is not within acceptable limits, if renegotiation is unsuccessful or if the service continues to deteriorate, the consumer may switch to an alternate provider if available, depending on its strategy.

### 5.1 Visualising Framework Processes

Software has been developed for visualising the framework processes at runtime. The negotiation viewer displays the consumers actively negotiating a service or service composition. The viewer provides a means to view negotiation sessions between each consumer broker and the provider brokers they negotiate with.

Fig. 6 shows a negotiation session between a consumer device and a *MapService* provider. The final proposal was rejected as another provider was more

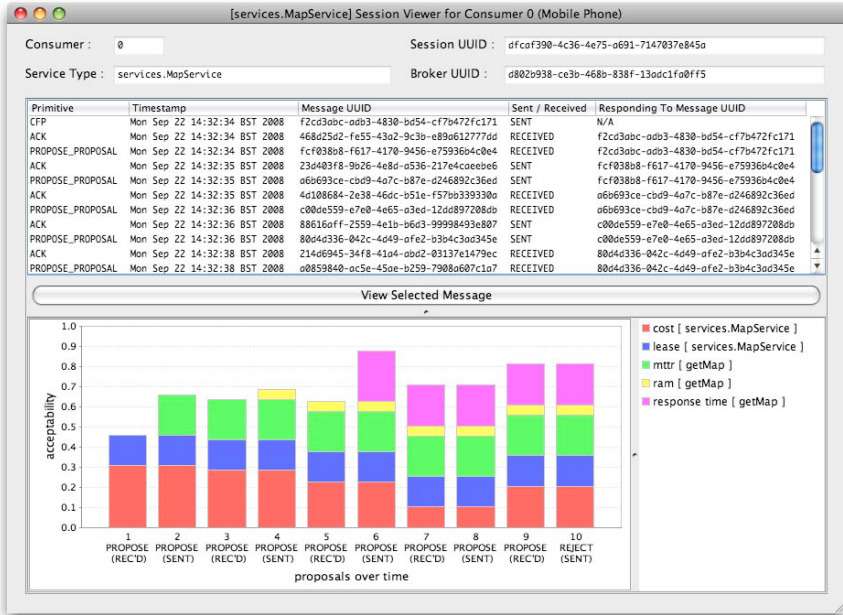
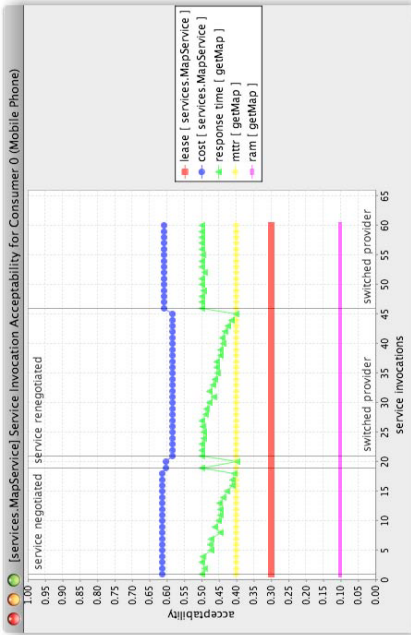


Fig. 6. Negotiation session viewer





(a) Service invocation acceptability



(b) Monitor event viewer

Fig. 7. Example simulation views

acceptable. The first table contains the negotiation primitive of each negotiation message; the second column contains the timestamp; the third column contains the universally-unique identifier (UUID) of the message; the fourth column indicates whether the message was sent or received by the consumer broker; and the fifth column contains the UUID of the message being responded to (if any). The user can view the contents of any message and service proposal.

The user can also view and compare negotiated service proposals from each provider of a particular service, and view the invoked acceptability of individual services (see service example in Fig. 7a) and service compositions.

The monitor viewer (see Fig. 7b) shows each consumer currently executing services. Each table entry represents a single monitoring event. The entry includes the UUID and type of the service monitored, the service operation invoked, the quality audited (profiled RAM usage and response time in the example), the SLA quality value, the observed quality value, when the audit occurred, and audit result. The observed and forecast values for different qualities are plotted on a chart where trends in service qualities can be observed over time. Consumers react to failed/failing services by requesting their brokers to renegotiate, and may switch provider if renegotiation fails (as shown on Fig. 7a).

The user is also able to view the current and historical reputation of each provider of each service type, based on the ratings from consumers.

## 6 Conclusions

This paper has presented a runtime quality architecture for service-oriented systems, that uses a novel service brokerage model to provide a framework for integrating consumer strategies with different negotiation and monitoring techniques. The framework enables consumers to negotiate service agreements which are closer to their requirements, and compensates providers accordingly. Services are monitored during runtime for compliance with the negotiated agreements. Problematic services are renegotiated, or alternate sources of service provision are sought. Our approach does not provide a definitive solution to all quality assurance problems that plague service-oriented systems. However, we believe it offers a real alternative to current quality frameworks and provides some answers to the problems that we set out at the beginning of this paper.

Current quality management initiatives allow the consumer only limited control over the quality of provided services; we have developed a framework that allows consumers to specify quality-weighted services and to associate these with consumer strategies. Effective runtime quality assurance must combine monitoring with effective recovery and self-management strategies. We have developed a portable self-managing quality architecture, that uses a lightweight service brokerage model to integrate pluggable monitoring and negotiation with consumer quality strategies for ensuring runtime quality. Lastly, we have demonstrated the effectiveness of the quality architecture with the simulation of a small resource-constrained example. We are currently investigating improvements to the framework to support runtime quality more efficiently in resource-constrained system environments.

We are also looking at better ways of expressing and incorporating fallback mechanisms as part of the consumer strategy, and better ways of integrating a quality of service ontology. Fallback mechanisms would provide the consumer with the ability to function in the event that certain framework components or services are unavailable.

## References

1. Turner, M., Budgen, D., Brereton, P.: Turning software into a service. *Computer* 36(10), 38–44 (2003)
2. Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River (2005)
3. Sommerville, I.: 31. In: *Software Engineering*, 8th edn. Addison Wesley, Reading (2006)
4. Lüders, F., Flemström, D., Wall, A.: Software component services for embedded real-time systems. In: *Proc. Fifth Conference on Software Engineering Research and Practice in Sweden, Västerås, Sweden, Mälardalen University, October 2005*, pp. 123–128 (2005)
5. Baresi, L., Ghezzi, C., Guinea, S.: Smart monitors for composed services. In: *IC-SOC 2004: Proceedings of the 2nd international conference on Service oriented computing*, pp. 193–202. ACM Press, New York (2004)
6. Hoffman, B.: Monitoring, at your service. *Queue* 3(10), 34–43 (2005)

7. Milanovic, N., Richling, J., Malek, M.: Lightweight services for embedded systems. *Wstfeus* 00, 40 (2004)
8. O'Sullivan, J., Edmond, D., Hofstede, A.T.: What's in a service? Towards accurate description of non-functional service properties. *Distrib. Parallel Databases* 12(2-3), 117–133 (2002)
9. Toma, I., Foxvog, D., Jaeger, M.C.: Modeling QoS characteristics in WSMO. In: *MW4SOC 2006: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pp. 42–47. ACM Press, New York (2006)
10. Menascé, D.A., Ruan, H., Gomaa, H.: QoS management in service-oriented architectures. *Perform. Eval.* 64(7-8), 646–663 (2007)
11. Woodside, C.M., Menascé, D.A.: Guest editors' introduction: Application-level QoS. *IEEE Internet Computing* 10(3), 13–15 (2006)
12. Martin, D.L., et al.: Bringing semantics to web services: The OWL-S approach. In: Cardoso, J., Sheth, A.P. (eds.) *SWSWPC 2004*. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
13. Andrieux, A., et al.: Web services agreement specification (WS-Agreement), version 2006-09-07. Technical report, Global Grid Forum (2006)
14. O'Sullivan, J.: Towards a Precise Understanding of Service Properties. PhD thesis, Queensland University of Technology (2006)
15. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. *Decis. Support Syst.* 43(2), 618–644 (2007)
16. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: *ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing*, pp. 212–221. ACM Press, New York (2004)
17. Wishart, R., Robinson, R., Indulska, J., Josang, A.: SuperstringRep: reputation-enhanced service discovery. In: *ACSC 2005: Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pp. 49–57. Australian Computer Society, Inc., Darlinghurst (2005)
18. Ludwig, H., et al.: Web service level agreement (WSLA) language specification (2003), <http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf>
19. Yan, J., et al.: Autonomous service level agreement negotiation for service composition provision. *Future Gener. Comput. Syst.* 23(6), 748–759 (2007)
20. Czajkowski, K., Foster, I.T., Kesselman, C., Sander, V., Tuecke, S.: Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Feitelson, D.G., Rudolph, L., Schwiiegelshohn, U. (eds.) *JSSPP 2002*. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002)
21. Benjamim, A.C., Sauvé, J., Cirne, W., Carelli, M.: Independently auditing service level agreements in the grid. In: *Proceedings of the 11th HP OpenView University Association Workshop, HPOVUA* (2004)
22. Baresi, L., Ghezzi, C., Guinea, S.: Towards self-healing service compositions. In: *PRISE 2004, First Conference on the PRinciples of Software Engineering*, Buenos Aires, Argentina (November 2004)
23. Menascé, D.A., Dubey, V.: Utility-based QoS brokering in service oriented architectures. *ICWS* 0, 422–430 (2007)
24. Li, H.: Automated E-business Negotiation: Model, Life Cycle and System Architecture. PhD thesis, University of Florida (2001)
25. Lock, R.: TRANSACT (Tool for Real-time Automated Negotiation of Secure Authorisation ContractTs). PhD thesis, Lancaster University (2005)
26. Wolski, R.: Dynamically forecasting network performance using the network weather service. *Cluster Computing* 1(1), 119–132 (1998)
27. Meyer, B.: Applying “design by contract”. *Computer* 25(10), 40–51 (1992)

# QoS Policies for Business Processes in Service Oriented Architectures

Fabien Baligand<sup>1,2,\*</sup>, Nicolas Rivierre<sup>1,\*</sup>, and Thomas Ledoux<sup>2</sup>

<sup>1</sup> France Telecom - R&D / MAPS / AMS,  
38-40 rue du general Leclerc, 92794 Issy les Moulineaux, France  
fabien.baligand@gmail.com, nicolas.rivierre@orange-ftgroup.com

<sup>2</sup> OBASCO Group, EMN / INRIA, Lina  
Ecole des Mines de Nantes,  
4, rue Alfred Kastler, F - 44307 Nantes cedex 3, France  
thomas.ledoux@emn.fr

**Abstract.** The advent of Service Oriented Architectures tends to promote a new kind of software architecture where services, exposing features accessible through highly standardized protocols, are composed in a loose coupling way. In such a context, where services are likely to be replaced or used by a large number of clients, the notion of Quality of Service (QoS), which focuses on the quality of the relationship between a service and its customers, becomes a key challenge. This paper aims to ease QoS management in service compositions through a better separation of concerns. For this purpose, we designed QoS4BP, a domain-specific language which allows QoS policies specification for business processes. More specifically, the QoS4BP language is designed to allow an architect to specify QoS constraints and mechanisms over parts of BPEL compositions. This language is executed by our ORQOS platform which cooperates in a non-intrusive way with orchestration engines. At pre-deployment time, ORQOS platform performs service planning depending on services QoS offers and on the QoS requirements in QoS4BP policies. At runtime, QoS4BP policies allow to react to QoS variations and to enact QoS management related mechanisms.

## 1 Introduction

QoS management in service compositions presents multiple challenges both statically and at runtime. Static time occurs before a composition is deployed on the orchestration engine. This step requires the selection of services whose QoS offers can satisfy the QoS requirements of the composition [18]. Because the number of functionally equivalent services is likely to grow larger over the Web, it becomes crucial for architects to have methods and tools allowing them to specify, compute and guarantee QoS of their compositions [7]. Runtime occurs while the composition is executed on the orchestration engine. At this step, QoS

---

\* This work was partially supported by the FAROS research project funded by the French RNTL.

of services may vary. Such variations are likely to violate guarantees and lead to variations of the composite service level that need to be dynamically counter-balanced [4]. Furthermore, in addition to performance, QoS requirements such as security, reliable messaging and transaction, which rely on WS-\* protocols, are major QoS features in service composition that must be addressed [8]. Many solutions offer some interesting methods and tools for QoS management, but we think that none of them takes into account five criteria that seem particularly relevant to the issue of QoS management:

- **Reuse of existing standards** criterion focuses on the capacity of approaches to leverage standards like BPEL language or Service Level Agreement (SLA) [4]. On the whole, although BPEL is almost always used, few approaches reuse works around SLA.
- **Separation of concerns** criterion aims to evaluate how service composition and QoS management logics are isolated one from another. Although many approaches promote a better separation of concerns, very few concretely achieve a precise analyze of roles as well as their domains of concern. This often leads to tangled logics or intrusive platforms [16][2], hence reducing maintainability and reusability.
- **Coverage of QoS management** criterion evaluates how much of QoS domain the solution handles. More specifically, QoS domain includes performance properties [18][7], such as availability or throughput, as well as non functional properties, like security or reliable messaging [9]. We noted that approaches tackle a wide variety of properties in regard to QoS but few or none actually consider both performance properties and non functional mechanisms (such as security). With emerging works around SLA, taking into account these various properties tends to become necessary for anyone who deals with QoS in business processes.
- **Dynamicity** criterion shows how approaches can handle QoS management statically, at runtime or both. The study of this criterion tends to show that approaches are fairly dedicated to either static time [7] or runtime [18], hence reducing the scope of applications of these solutions.
- **Expressivity** criterion evaluates how rich and complex the interfaces offered by the approaches for QoS specification are. Although some approaches include a language to specify QoS management, the expressivity of these languages can be either restrictive [12], or complex to handle [16].

Our approach aims to bring solutions to the challenges of QoS management in SOA by creating a Domain-specific Language (DSL). This language, called “QoSL4BP” (Quality of Service for Business Process), is executed by our platform, namely “ORQOS” (ORchestration Quality Of Service), that performs both at pre-deployment time (for static QoS verification) and at runtime (for dynamic QoS adaptation), as detailed in [1]. The remainder of the paper is organized as follows: Section [2] presents the key decisions for the design of our approach. Section [3] presents a scenario from telecommunication world. Section [4] describes the QoSL4BP language, and Section [5] details its execution model. Section [6] discusses the related works. Finally, Section [7] concludes and outlines future work.

## 2 Motivation

**Study of roles.** To materialize QoS management and service composition concerns, we identify roles surrounding service compositions, as depicted in Figure 1. Among these roles, the “Composition Architect” role is in charge of the development and of maintaining the functional part of the service composition using BPEL language. The “Integrator Architect” role is in charge of managing the QoS of the service composition by binding the potential Service Providers to the service composition and by managing the relation with each Service Provider. In particular, the Integrator Architect has to negotiate contracts with each Service Provider in order to find appropriate candidates implementing the interfaces provided by the Composition Architect. He can also specify fine-grained requirements on certain parts of the composition and perform QoS related mechanisms like security or replanning strategies on some limited scopes of the composition. For that matter, he requires a “grey box” vision of the service composition, consisting in workflow activities and messages being exchanged. This role is bring to additional QoS requirements to improve QoS or gain profits. For services whose SLA is missing, he can attach QoS offers so that service planning can still be performed. Also, he can specify specific strategies for service planning statically and at runtime, hence reducing the computation cost.

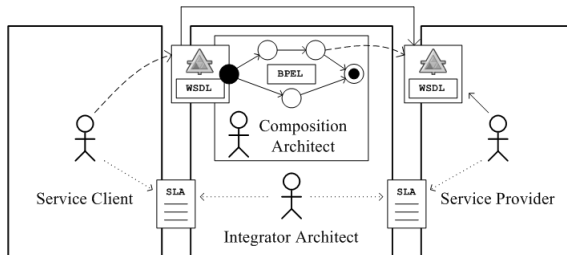


Fig. 1. Roles in service compositions

**Domain Study of QoS Management.** QoS management requires a variety of treatments and data that characterizes the Integrator Architect role and encompasses the study domain we are interested to capture.

*Data related to QoS Management in Business Processes.* As depicted in Figure 2, the data related to our domain encapsulates the data belonging to the **SLA** domain, to the **WS-\* mechanisms** domain, and to the **BPEL** domain. A SLA specifies the QoS between a client and a provider. In particular, we focussed on the WS-Agreement standard which is composed of terms referencing a party and a SLO acting as an offer guaranteed by the targeted party. A SLO specifies either a performance constraint or WS-\* mechanisms. WS-\* mechanisms domain refers to the standards described in the Web Service Architecture such as WS-Security or WS-ReliableMessaging. Finally, BPEL domain is reified into basic or composite BPEL activities which may be linked to partners as well as may contain

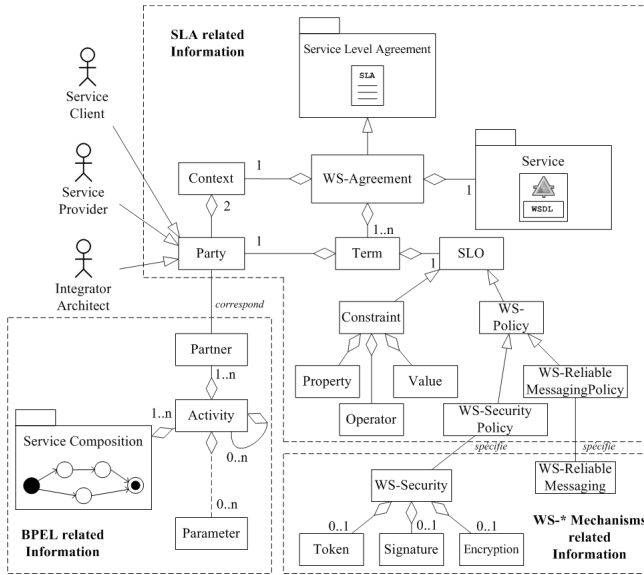
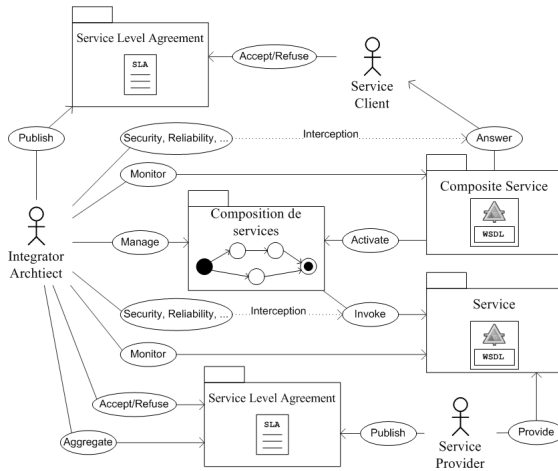


Fig. 2. Data related to QoS Management in Business Processes

additional parameters useful for QoS computation (e.g rate of path selection in *Switch* activities or the average number of loops in *Loop* activities).

*Treatments related to QoS Management in Business Processes.* The Integrator Architect role is bound to a variety of tasks that are described as treatments in our domain study. Such treatments are depicted on Figure 3. First, the Integrator Architect is in charge of **SLA management** with the partners of the business process (client and Service Providers). This implies selection of QoS offers that guarantee the service level specified in the business processes. The Integrator Architect also deals with **QoS observation** through QoS monitoring and BPEL parameters supervision. Finally, he manages **WS-\* mechanisms** enactment as described by the WS-\* standards.

**Domain Specific Language Orientation.** To provide the Integrator Architect role with a programmatic interface, we chose to design a Domain Specific Language (DSL) [10] capturing the domain of QoS management in business processes. A DSL is a high-level language providing constructs appropriate to a particular class of problems. Domain expertise is made explicit in the language abstractions and directly supported through its implementation rather than coded by the programmer. The avoidance of low-level source code in itself improves program robustness. More importantly, the use of domain-specific constructs enables or facilitates precise, domain-specific verifications of properties, such as termination properties or critical safety properties, which would be impossible or costly to perform on code written in a general-purpose language. By integrating the concepts of the domain we analyzed, our DSL aims to provide the Integrator Architect role with high level descriptions for QoS management in business processes.



**Fig. 3.** Activities related to QoS Management in Business Processes

**Non Intrusivity.** Our contribution lays on the principle of separation of concerns by isolating the logic responsible for QoS management in a DSL from the business process logic. However, the execution of these two logics requires a recomposition method. In order to be able to reuse both the BPEL language as well as the BPEL platforms, we chose not to modify neither the BPEL language nor an existing BPEL platform. Thus, our strategy consists in binding the two languages via loose coupling. Indirections are inserted into the code of the primary concern (i.e BPEL document) in order to call the QoS logic. Such a strategy aims to be non intrusive with existing languages and platforms.

### 3 Case Study

**Scenario.** Depicted in Figure 4, the “Urban Trip Planner” (UTP) scenario is an example<sup>1</sup> of Web Service orchestration, that we used to exemplify our approach. The UTP service aims to plan trips in big cities by delivering the complete transportation route and commutes list as well as a map showing the path from the last station to the final destination. The UTP Service is composed of multiple services. It requires both a destination and a device identification number as inputs. Next, the request is sent to two different services in parallel. These services belong to a flow activity named “LocationScope”. The first service uses the device identification number and returns the client current location (for instance, using a Wifi access point location service). The second service takes the destination in input and returns the exact address, using the Yellow Pages

<sup>1</sup> Although being an imaginary use case, this scenario corresponds to issues that companies like France Telecom have to deal with.



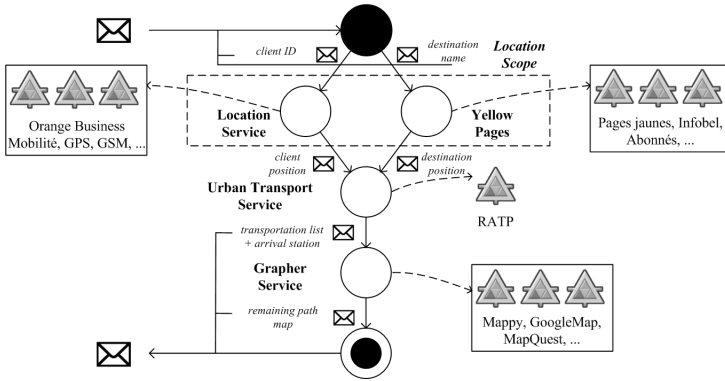


Fig. 4. UTP Service Composition

service. Upon reception of both replies, the UTP service sends both addresses to a Transportation service that returns the route details. The final station address and the destination address are sent to a Grapher service that delivers a map of the path from the station to the destination. Eventually, both the route details and the map are returned to the user.

**QoS Requirements.** The Integrator Architect may specify multiple requirements related to QoS management. For example: **Global QoS Offers.** The Integrator Architect wants to guarantee the UTP service QoS to UTP clients. For service selection, the Integrator Architect choose to apply more constrained requirements to make a profit and to ensure that the QoS offers are respected. He also wants to set up a reliable messaging mechanism with the client. If these requirements are not maintained at runtime, replanning must be triggered. **Missing SLA.** The transportation service is unique (RATP service) and does not provide QoS offers. However, the Integrator Architect found out empirically that its QoS exhibits a response time less than 5 ms, a throughput over 200 requests per minute and a cost equal to 10 cents per call. He has to isolate this service and make up QoS offers. Should the QoS of the service vary, then UTP service should fail. **LocationScope Specific Management.** In order to reduce the computation task of guaranteeing the global QoS, the Integrator Architect requires that the composite activity “LocationScope” handles its own QoS. In particular, it should handle service planning and replanning. Also, an encryption mechanism is required with the partners of the activities included in “LocationScope” to preserve confidentiality of exchanged messages. **Grapher Service Rationing.** In order to reduce the selection process for this service, the Integrator Architect asks an additional constraint to set a maximum cost for the Grapher service.

## 4 A Language for QoS Policies Specification

**Design.** The QoSL4BP language was elaborated, based on our domain study. Thus, QoSL4BP language includes a data model that reifies the information

associated with this domain, as well as primitives which capture the mechanisms described in this domain. To design the QoSL4BP language, we focus on a couple of properties.

First, QoSL4BP is conceived as a high-level and declarative language. Such a property implies that the programmer manipulates concepts close to the domain idioms and is not required to deal with complex control of algorithms. To ensure that, QoSL4BP enables specification of QoS objectives and its expressivity is limited to a set of relevant mechanisms and data. Thus, QoSL4BP language aims to bring an appropriate balance between rich expressivity and complexity.

Secondly, QoSL4BP language brings a clear separation between static and dynamic specifications by isolating the specifications into two sections. This hybrid approach helps bringing more dynamicity in order to handle QoS management effectively both at pre-deployment time and at runtime. QoSL4BP language provides support for QoS management at diverse granularity levels (whole business process, basic or composite activities), which helps to rationalize QoS management. To increase homogeneity and reusability, the QoSL4BP specifications are contained in modules called “policies”. Finally, QoSL4BP language possesses some guarantees relevant to QoS management in business processes. For brevity’s sake, we do not provide details about them in this paper. However, these guarantees ensure BPEL and QoSL4BP logics synchronisation, BPEL activity types verification (basic or composite activities), policy composition (when two policies interact, specifications of the policy whose target granularity is the finest override those of the policy whose rather granularity is coarser), exception handling and termination.

**Structure.** The structure of QoSL4BP policies is shown in Figure 5. They are made of three sections: the section SCOPE describes the BPEL activities target whose QoS management logic is attached to, whereas the sections INIT and RULES respectively contain the static and dynamic logics specification. Thus, INIT specifications are executed at pre-deployment time and RULES specifications are executed at runtime. Instructions of the section RULES can be written as “condition-action” rules.

The data model of QoSL4BP language emerges from the domain study, depicted in Figure 2. Based on this study, we choose BPEL activities and SLA information as structuring elements for the data model of QoSL4BP language. **BPEL activities** are used both to specify the SCOPE section of the policy and to designate the target instructions in the INIT and RULES sections. In particular, in QoSL4BP, a BPEL activity may be linked to one or several partners,

```

POLICY "policy_name" = {
  SCOPE = { BPEL_activities_selection }           // Target Specification
  INIT  = { ( static_instr ) + }                 // Static Logic Specification
  RULES = { ( dynamic_instr ) +                 // Dynamic Logic Specification
            ( condition -> dynamic_instr ) +
  }
}

```

Fig. 5. QoSL4BP Policy Structure

hence allowing to easily attach some QoS offers, requirements or mechanisms. To enable the programmer to efficiently point out BPEL activities in a business process, QoSL4BP offers a specific semantic including some activity composition operators. **SLA information** enables to specify QoS data related to QoS performance or mechanisms. In QoSL4BP, a SLA consists in a collection of terms. A term is made of one QoS constraint and of a party responsible for the QoS constraint. A QoS constraint can be considered either as an offer or as a requirement depending if the constraint is guaranteed to an external party or if an external asks for the constraint to be applied. Constraints can be specialized either into a performance constraint (*e.g* Response time less than 10 ms) or into a QoS mechanism (*e.g* encryption using the “RSA algorithm”).

Similarly to the data model, the primitives emerge from the domain study and aim to provide the Integrator Architect with expressivity to perform the activities depicted in Figure 3. Primitives appear in Figure 6 and are categorized depending on their concern and on the parties involved: The primitives SET\_OFFER and SET\_REQUIREMENT makes it possible to set QoS constraints which are processed at service selection, performed using primitive PLANNING. Services can be manually set or obtained using the BIND and BOUND primitives. READ\_CLIENT and READ\_PROVIDER allow to read constraints from accepted SLA. The primitives SENSOR and MONITOR return monitored QoS values at runtime. VIOLATION\_ACTIVITY and VIOLATION\_PROVIDER allows to verify whether the requirements of a SLA are not violated. The primitives PERFORM\_CLIENT and PERFORM\_ACTIVITY allow to specify a WS-\* mechanism to perform either with the client, or with one to several activities. THROW and CATCH are means to manage exceptions.

**Illustrations with UTP Scenario.** For brevity’s sake, we exhibit only two out of the four policies. Figure 7 exhibits the policy “selection\_RATP” corresponding to the QoS requirement called **Missing SLA**. This policy applies to the “Invoke-Transportation” activity. At pre-deployment time, some QoS offers are attached to this activity and the service is bound to the partnerlink “WS\_RATP”. At

		Integrator-Client	Integrator-Activity	Integrator-Provider
SLA Management	Constraints	SET_OFFER	SET_OFFER SET_REQUIREMENT	SET_REQUIREMENT
	Planning		PLANNING	PLANNING BIND BOUND
	Access	READ_CLIENT		READ_PROVIDER
QoS Observation	Monitoring	SENSOR	SENSOR MONITOR SET/GET_RATE SET/GET_LOOP	MONITOR
	Violation	VIOLATION_ACTIVITY	VIOLATION_ACTIVITY	VIOLATION_PROVIDER
QoS Mechanism	WS-*	PERFORM_CLIENT	PERFORM_ACTIVITY	PERFORM_ACTIVITY
	Exception	THROW	CATCH (LOG)	

Fig. 6. QoSL4BP Primitives

```

POLICY "selection_RATP" = {
  SCOPE = { INVOKE["InvokeTransportation"] }
  INIT = {
    SET_OFFER(([RESPONSETIME < 5], [THROUGHPUT > 200], [COST < 10]));
    BIND(JOIN_ACTIVITY, "WS_RATP");
  }
  RULES = {
    VIOLATION_ACTIVITY(JOIN_ACTIVITY) -> {
      THROW("UTP Service QoS cannot be maintained any longer.");
    }
  }
}

```

**Fig. 7.** “selection\_RATP” Policy

runtime, if the QoS requirements for the “LocationScope” activity are violated, then the UTP service should send an exception.

Figure 8 exhibits the policy “location\_security” corresponding to the QoS requirement **LocationScope Specific Management**. This policy applies to the “LocationScope” flow activity. At pre-deployment time, some QoS offers and requirements are attached to this activity. Service planning is performed on the target activity using a constraint programming algorithm (CP) [11]. At runtime, two encryption mechanisms are performed for the invoke activities of “LocationScope”. Should a security exception be raised, then it should be logged and the UTP service should send an exception. Also, if the QoS requirements for the “LocationScope” activity are violated, then service replanning is to be performed. Should this replanning fail, then it should be logged and the UTP service should send an exception.

## 5 Execution Model

**Global process.** Figure 9 depicts the global process of QoSL4BP policies execution, including three steps. Before this process takes place, the Composition Architect must provide a BPEL document and a set of generic WSDL describing the functional interface of each service involved in the composition. The Integrator Architect specifies QoSL4BP policies and a list of Service Providers for each service of the composition.

(1) First, the static specifications (from the INIT section) are executed by ORQOS. The values are set for the initial states of the system (offers and requirements of activities, BPEL parameters). At this step, ORQOS performs service planning to find a set of initial Service Providers. This results either in a failure or a successful attempt for service planning. (2) The second step also occurs before the deployment of the BPEL composition. It aims to modify the BPEL document by weaving indirections (or “hooks”) calling ORQOS platform at specific points requiring QoS policies execution. (3) Final step occurs at runtime, when the composition is executed by clients and calls Service Providers. At this step, ORQOS platform is called by the hooks from the BPEL document. It also intercepts messages between the composition and its partners to perform the appropriate treatments (monitoring, security, service replanning, etc.).

```

POLICY "location_security" = {
  SCOPE = { FLOW["LocationScope"] }
  INIT = {
    SET_OFFER(([RESPONSETIME < 13], [THROUGHPUT > 110],[COST < 15]));
    SET_REQUIREMENT([RESPONSETIME < 13],[THROUGHPUT > 110],[COST < 15],[SECURITY:ENCRYPTION]);
    PLANNING(JOIN_ACTIVITY, CP);
  }
  RULES = {
    PERFORM_ACTIVITY(INVOKE["InvokeLocation"],[SECURITY : ENCRYPTION],("3DES", "key1.."));
    PERFORM_ACTIVITY(INVOKE["InvokePhoneBook"],[SECURITY : ENCRYPTION],("RSA", "kek2.."));
    CATCH("SECURITY EXCEPTION") -> {
      LOG("Encrypted message failure.");
      THROW("UTP service fails.");
    }
  }

  VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
  PLANNING(JOIN_ACTIVITY, CP);
  CATCH("PLANNING EXCEPTION") -> {
    LOG("LocationScope services planning fails.");
    THROW("UTP Service QoS cannot be maintained any longer.");
  }
}
}

```

Fig. 8. "location\_security" Policy

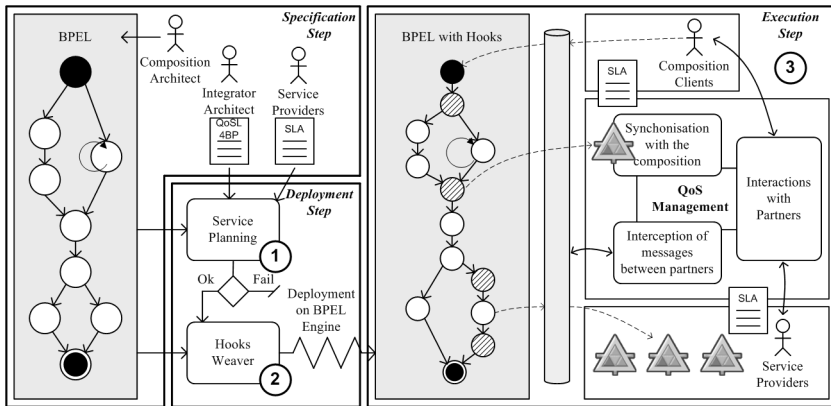


Fig. 9. Global Process

**Static planning.** First, the BPEL composition is translated into a tree, as shown in Figure 10. Composite activities are translated into nodes whereas basic activities become leaves. The values specified in the INIT section of policies (QoS offers, requirements, etc.) are attached to the elements of this tree. Using a tree as a model to reflect on QoS of the composition allows to efficiently associate QoS information and structure of this information.

Next, the tree is decomposed into multiple sub trees: for each element containing QoS offers, this element and its potential leaves are detached to become a new sub tree. This transformation is also depicted in Figure 10. Such decomposition allows to reduce the planning computation effort due to the fact that activities containing QoS offers act as guarantees to the rest of the composition.

Thus, the maximal computational complexity is  $\sum_{i=0}^n s^{c_i}$  which is in  $\Theta(s^{\max(c_i)})$  (with  $n$  the number of sub trees,  $c_i, i \in [1, n]$  the number of leaves in the sub tree  $i$ , and  $s$  the average number of Service Providers).

Finally, service planning is performed in each sub tree using QoS offers and requirements, SLA data of each Service Provider, as well as composition rules as described in [5]. Although ORQOS may implement any existing algorithm, it offers so far two algorithms, using constraint programming and backtracking. Constraint programming is performed by translated each sub tree into a constraint network then by using a constraint solver tool. Backtracking algorithm is performed by exhaustively trying each SLA offer of Service Provider and testing against QoS requirements from the bottom to the top of each sub tree, back and forth when it fails.

**Preparation of the composition.** At pre-deployment time, some hooks are weaved before and after each BPEL activity. These hooks consist in invoke activities calling an ORQOS platform Web service interface. This enables to execute policies without modifying the BPEL engine. Next, in order to perform some mechanisms (e.g monitoring or security), ORQOS platform requires to intercept the messages exchanged between the composition and its partners. For this purpose, ORQOS platform acts as a proxy. In particular, partnerlinks of the BPEL document are redirected to the ORQOS platform and includes header identification information so that ORQOS can bind received messages with their activities.

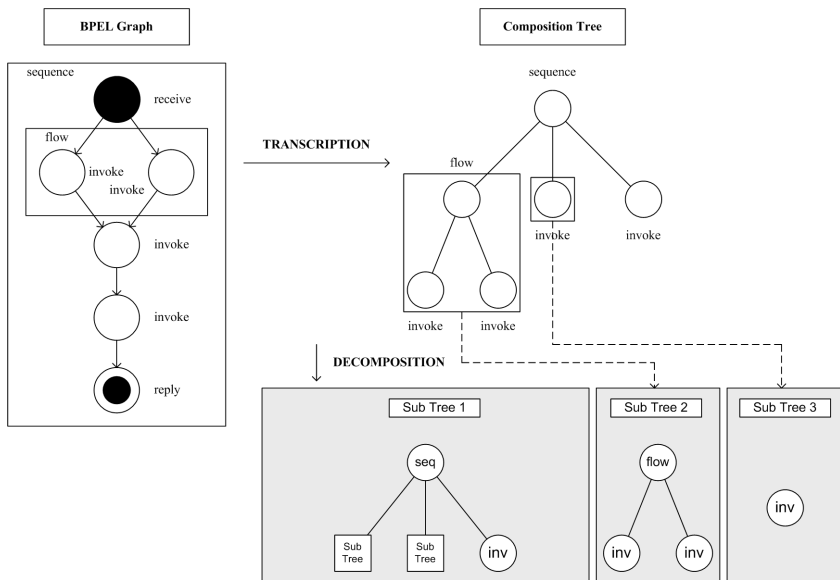


Fig. 10. Transcription and Decomposition

**Dynamic Execution.** The execution of QoSL4BP policies processes the RULES sections of the policies at runtime. Synchronization between BPEL and policies is achieved by the hooks weaved at pre-deployment time. Rules are executed before and after each activity since the QoS state of the system is likely to change after the execution of the targeted activity, called “Join Activity” similarly to “Join Point” in Aspect Oriented Programming [13]. To perform the mechanisms specified in the rules, ORQOS platform integrates already existing components such as Apache WSS4J, Sandesha and WSAG4J which implements WS-Security, WS-ReliableMessaging and WS-Agreement SLA management logics respectively.

## 6 Related Works

**Reuse of existing standards.** Except AgFlow [17,18] and WS-Binder [14,4], few approaches reuse SLA works to handle QoS in compositions. In [18] the authors tackle the issue of replanning as a graph search problem, using integer programming technique for service selection. Their method allows to select, for each abstract service of a workflow, a concrete service so that their QoS aggregation is optimized and that the global constraints are satisfied. WS-Binder performs service planning, and it allows to specify and to compose domain specific QoS properties. Our approach reuses both SLA semantic (QoS offers and requirements integrated in QoS L4BP language) and tools (WS-Agreement protocol implementation). It provides the Integrator Architect with flexibility to decide about the triggering decision and about the BPEL activities where service replanning should occur.

**Separation of concerns.** Each platform that handles QoS in service compositions has to deal with QoS management concern and business process concern collaboration. Some approaches, such as DYNAMO [2,3] and MASC [16], focus on how to isolate the specifications of these logics. DYNAMO provides two languages, WSCoL (Web Services Constraint Language) and WSReL (Web Service Recovery Language), that aim to allow the specification of monitoring and reactions in BPEL processes. In MASC (Manageable and Adaptive Service Compositions), the authors bring an Event-Condition-Action based language called WS-Policy4MASC allowing to specify adaptation strategies in service compositions. However, the tools in charge of the execution of these specifications are tangled up which prevent from reusing BPEL engines without modification. Other works, like TRAP/BPEL [12] or WS-Binder, isolate both the specifications and the execution tools. TRAP/BPEL is based on *Transparent shaping* [15] which aims to bring new behaviors in a non intrusive way by inserting hooks calling the logic of these behaviors at runtime. Our approach is based on a similar strategy that allows both specifications (BPEL and QoS L4BP) and implementation platforms (BPEL engine and ORQOS platform) to be kept isolated. This strategy increases logic and platform maintenance, evolutivity and reusability.

Furthermore, because separation of concerns is a major focus in our work, we created the Integrator Architect role and emphasized its domain of concern.

**Coverage of QoS management.** On the whole, platforms focus either on performance properties concerns (AgFlow and ORBWork [7,6]) either on non functional concerns such as security or reliable messaging (AO4BPEL [8,9]). ORB-Work propose mathematical models for workflow QoS computation to predict QoS, using an algorithm consisting of a set of graph reduction rules which are defined for time, cost, reliability and fidelity metrics. In AO4BPEL, the authors have elaborated a language named “Aspect Oriented for Business Process Execution Language” (AO4BPEL) to bring AOP mechanisms to the BPEL language. By using the SLA as a fundamental basis for our language, our approach aims to provides answer to any QoS concern that can be described by a Service Level Objective (either performance properties or WS-\* mechanisms). Also, because policies are attached to BPEL activities, this enables to apply QoS management to various granularity levels (whole composition, set of activities, basic activity).

**Dynamicity.** Many approaches (WS-Binder, AO4BPEL, DYNAMO) perform QoS management only during the execution of the business processes and do not consider QoS at pre-deployment time. However, ORBWork brings QoS guarantees or perform service planning at pre-deployment time but do not consider QoS at runtime. Also, AgFlow and WS-Binder platforms offer static solutions for service planning at pre-deployment time, as well as strategies for adaptation at runtime. By handling both predeployment time and runtime for QoS management, our approach aims to provide the Integrator Architect with abilities to compute global service planning statically and to specify fine-grained strategies for dynamic service replanning.

**Expressivity.** Solutions like AO4BPEL, DYNAMO or MASC provide languages, while some others, such as WS-Binder or AgFlow, have the QoS management logic directly integrated in their platforms. DYNAMO offers to manipulate supervision and recovery with its two languages in a declarative mode. MASC provides a language that aims to answer to a wide variety of concerns, which introduces complexity. AO4BPEL reuses AspectJ syntax, XPath and BPEL language to allow the user to specify BPEL adaptation aspect in a imperative mode. To provide a relevant expressivity and to encapsulate efficiently QoS management concerns, QoSL4BP language was designed as a declarative language based on a precise study of the domain of QoS management in business processes. In particular, it enables to specify using SLA and BPEL activities information, and its expressivity is limited to a reasonable amount of primitives, making it efficient to use yet relatively easy to learn.

## 7 Conclusion

In this paper, we presented a language and a platform addressing QoS management in business processes, both at predeployment time and at runtime. Our language called “QoSL4BP” (Quality of Service Language for Business Processes) allows an architect to specify QoS policies over scopes of BPEL compositions.



Static and dynamic execution of the QoSL4BP language are performed by our platform “ORQOS” (ORchestration Quality of Service), designed to be non intrusive with already existing infrastructures and languages. At pre-deployment time, ORQOS selects a set of services whose QoS offers match the QoS requirements attached to the BPEL activities of the composition. At runtime, ORQOS platform monitors the QoS of services and performs actions such as selecting different Service Providers, throwing BPEL exceptions, or performing WS-\* mechanisms, as specified in QoSL4BP policies.

Although solutions to deal with QoS management in business processes already exist, our approach aims to answer to some issues often encountered by these solutions. First, it focuses on reusing most of the already existing SOA standards such as BPEL, SLA and WS-\* norms. It aims to provide a better separation of concern by achieving a precise study of the domain of QoS management and by encapsulating this domain in QoSL4BP language as well as in ORQOS platform (which is non intrusive with BPEL engines). Our approach tends to allow the architect to manipulate both QoS performance properties and QoS non functional properties such as security, at pre-deployment and at runtime. Finally, QoSL4BP is designed as a declarative policy-based language that allows to manipulate concepts close to the domain idioms without being required to deal with complex control of algorithms.

In future works, we plan to enhance QoSL4BP language by addressing a larger number of QoS dimensions and mechanisms, and by studying the issue of policy composition that occurs when at least two policies scopes intersect. We also plan to investigate formal semantics of QoSL4BP language, as well as its re-design using WS-Policy standard.

## References

1. Baligand, F.: PhD thesis, Une Approche Déclarative pour la Gestion de la Qualité de Service dans les Compositions de Services (June 2008)
2. Baresi, L., Guinea, S.: Dynamo and self-healing bpel compositions. In: ICSE COMPANION 2007: Companion to the proceedings of the 29th International Conference on Software Engineering, pp. 69–70. IEEE Computer Society, Washington (2007)
3. Baresi, L., Guinea, S., Plebani, P.: Ws-policy for service monitoring. In: Bussler, C.J., Shan, M.-C. (eds.) TES 2005. LNCS, vol. 3811, pp. 72–83. Springer, Heidelberg (2006)
4. Canfora, G., Di Penta, M., Esposito, R., Perfetto, F., Villani, M.L.: Service composition (re)binding driven by application-specific qos. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 141–152. Springer, Heidelberg (2006)
5. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Modeling quality of service for workflows and web service processes (2002)
6. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Modeling quality of service for workflows and web service processes. Technical Report UGACS-TR-02-002, Computer Science Department, University of Georgia (2002)
7. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web* 1(3), 281–308 (2004)

8. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In: Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 168–182. Springer, Heidelberg (2004)
9. Charfi, A., Schmeling, B., Heizenreder, A., Mezini, M.: Reliable, secure, and transacted web service compositions with ao4bpel. In: Proceedings of the 4th IEEE European Conference on Web Services (ECOWS) (December 2006)
10. Consel, C.: Charles Consel. In: Lengauer, C., Batory, D., Consel, C., Odersky, M. (eds.) Domain-Specific Program Generation. LNCS, vol. 3016, pp. 19–29. Springer, Heidelberg (2004)
11. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers, San Francisco (2003)
12. Ezenwoye, O., Sadjadi, S.M.: Trap/bpel: A framework for dynamic adaptation of composite services. Technical Report FIU-SCIS-2006-06-02 (2006)
13. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
14. Penta, M.D., Esposito, R., Villani, M.L., Codato, R., Colombo, M., Nitto, E.D.: Ws binder: a framework to enable dynamic binding of composite web services. In: SOSE 2006: Proceedings of the 2006 international workshop on Service-oriented software engineering, pp. 74–80. ACM, New York (2006)
15. Sadjadi, S.M., McKinley, P.K., Cheng, B.H.C., Stirewalt, R.E.K.: Trap/j: Transparent generation of adaptable java programs. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3291, pp. 1243–1261. Springer, Heidelberg (2004)
16. Tasic, V., Erradi, A., Maheshwari, P.: Ws-policy4masc - a ws-policy extension used in the masc middleware. In: IEEE SCC, pp. 458–465. IEEE Computer Society, Los Alamitos (2007)
17. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW 2003: Proceedings of the 12th international conference on World Wide Web, pp. 411–421. ACM, New York (2003)
18. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.* 30(5), 311–327 (2004)

# Deriving Business Service Interfaces in Windows Workflow from UMM Transactions

Marco Zapletal

Institute of Software Technology and Interactive Systems, Vienna University of  
Technology, Austria

marco@ec.tuwien.ac.at

**Abstract.** Modeling inter-organizational business processes identifies the services each business partner has to provide and to consume as well as the flow of interactions between them. A model-driven approach to inter-organizational business processes allows abstracting from the underlying IT platform and, thereby, guarantees to survive changes in technology. UN/CEFACT's Modeling Methodology (UMM), which is defined as a UML profile, is currently one of the most promising approaches for modeling platform-independent business collaborations. However, well defined mappings to most of the current state-of-the-art candidate platforms are still missing. A candidate platform of growing interest is the Windows Workflow Foundation (WF). In this paper, we outline a mapping from the basic UMM building blocks, i.e. business transactions, to business service interfaces (BSI) implemented in WF.

## 1 Motivation

Business-to-Business (B2B) electronic commerce presupposes the integration of inter-organizational systems. In recent years, service-oriented computing has become the next evolutionary step in connecting autonomous enterprise systems. Service-orientation is considered as an enabler for aligning services in a business sense with their technical implementation. If each business partner, however, defines the service interactions with other partners in isolation, interoperability is unlikely. Consequently, B2B requires an approach that describes business collaborations from a global perspective. Furthermore, business logic should be abstracted from implementation specifics. UN/CEFACT's Modeling Methodology (UMM) [1] is a UML-based modeling language following this approach. It describes business collaborations from a neutral point of view by specifying the services each partner has to provide and to consume as well as the flow between them. A UMM model is not bound to any specific implementation platform. However, in order to realize a business service interface based on UMM, mappings to specific target platforms have to be provided. A typical candidate are Web Services based on the Business Process Execution Language (BPEL), which we already discussed in [2]. In addition to the pure Web Services stack, the Windows Workflow Foundation (WF) is a strong candidate for the implementation of business service interfaces. In this paper we show the transformation of UMM

business transactions to business service interfaces (BSI) realized in WF. The flow within the BSI is defined using WF's sequential workflow language. The interface to the workflow, however, is composed of well-defined business services implemented using Web Service specifications. Our model-driven approach yields three major benefits: First, business partners may agree on a global model serving as a kind of contract on the process. Second, the resulting business service interfaces of collaborating roles are complementary to each other ensuring their interoperability. Third, the graphical UMM representation abstracts from the complexity of a business service interface, which becomes evident in the workflow model.

Due to space limitations, we do not cover UMM business transactions in this paper, but refer to its long version that is published as a technical report [3].

## 2 The Transformation Process

In this section, we elaborate on the transformation from a global UMM business transaction model to partner-specific BSI's implemented in WF. For describing the mapping, we concentrate on the initiators's part of the process. Evidently, the responder's business service interface is complementary to the one of the initiator. In other words, when the initiator invokes something, the responder has to receive something. Thus, the order of sending and receiving business documents has to be reversed. The same applies for the handling of acknowledgments.

We detail the mapping by means of figure 1 depicting the derived business service interface for the initiator implemented in WF. The WF process is defined as a sequential workflow resulting in 11 major steps (A-K), whereby steps F to J

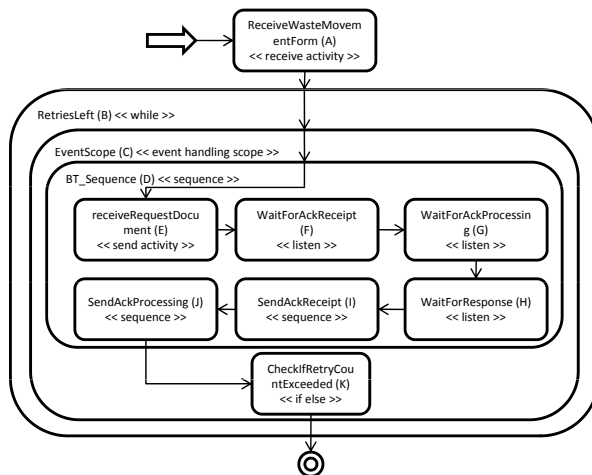


Fig. 1. The initiator's business service interface implemented in WF

contain nested activities. If not explicitly noted else, all used activity types are contained in WF's basic activity library.

**Step A: Interacting with the business application.** At the very beginning, the initiator's BSI receives the request document from the business application. Receiving the document is implemented by a *handle external event* activity. This presupposes that the business application is implemented in .NET as well. If this is not the case, the *handle external event* activity may be substituted by a *receive activity* for enabling cross-platform communication - for example realized by Web Service calls.

**Step B: Checking the available retries.** According to UMM business transaction semantics, the initiator has to re-start a business transaction in case of time-outs. A time-out occurs if a business document or a business signal is not received within an expected time frame. The maximum amount of retries is specified by the tagged value *retry count* in UMM. In WF, we use a *while activity* to repeat the execution of the business transaction if required. The *while activity* (B in figure 11) has to be executed until either the retry count is exceeded or the business transaction is considered as successful. Thus, we define the loop's condition as `retryCount >= 0 || businessTransactionSuccessful`, whereby both parameters are defined as normal .NET variables within the workflow. In case the business transaction is successful, the last action within the while loop sets the variable `businessTransactionSuccessful` to *true* (step H).

**Steps C and D: Listening to business signals during the regular process flow.** The only activity within the *while activity* is an *event handling scope activity*. This activity type allows to act upon events concurrently to the execution of the regular process flow. In UMM business transactions, business partner's may receive time-out exceptions or failed business control exceptions from their counterpart at any time during the course of a business transaction. Consequently, we use the *event handling scope activity* for receiving and processing business signals concurrently to the regular process flow. The *event handling scope activity* may have several event handlers attached - one for each event (i.e., business signal). Due to space limitations, we do not discuss event handlers in this paper, but refer to its long version [3]. Within the *event handling scope* the *sequence activity* (D in figure 11) serves as a container for the activities realizing the message exchange with the responder's BSI (steps E to J).

**Step E: Sending the request document.** Step E communicates the request document from the initiator's to the responder's BSI. On the initiator's side, the service call is implemented using the *send activity*. Note, that **receive request document** (E in figure 11) is indeed a *send activity*, which refers to the operation offered by the responder. The call is performed asynchronously, which means that the workflow continues immediately. The semantics of an asynchronous operation call by a *send activity* correspond to a truly fire-and-forget behavior. This entails that the client does not even receive a fault message from the service in case of an exception.

This behavior is in line with the semantics of asynchronous UMM business transactions patterns. Thereby, business document exchanges are completely

asynchronous in order to avoid blocking behavior of business service interfaces. Nevertheless, interacting business service interfaces share the same understanding about the state of a business document exchange by communicating business signals as shown in the following steps.

**Step F: Waiting for the acknowledgment of receipt.** After sending the request document, the initiator waits for a business signal of type *acknowledgment of receipt* from the responder's BSI. According to UMM business transaction semantics, an acknowledgment of receipt is issued after a received business document passes grammar-, schema-, and sequence validation.

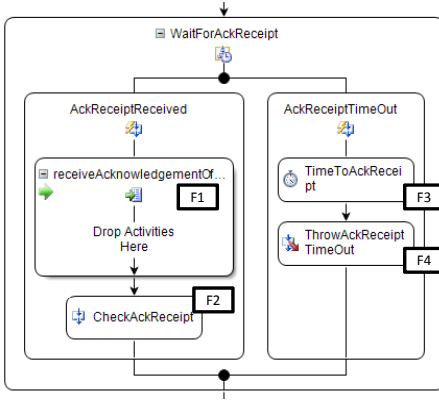
Figure 2 shows the required activities of step F in detail. The initiator expects the acknowledgment of receipt from the responder's BSI to confirm that the business document passed the syntactical checks. The *listen activity* in step F has two branches. The left branch is activated when the initiator's business service interface receives the acknowledgment of receipt. If the acknowledgment, however, is not picked up within the agreed time frame, the right branch is activated. The *listen activity* is responsible for activating that branch, whose trigger event occurs first. The remaining branches are canceled.

In order to expose a service for receiving the acknowledgment, the first activity in the left branch is a *receive activity* (F1). The *receive activity* is bound to an operation called **receive acknowledgment of receipt**. We define this operation in a service contract particularly for business signals. This service contract is not restricted to any business context and may be globally defined for business transactions. In this paper, we assume that at least the initiator and the responder bind their BSI's to this service contract for exchanging business signals.

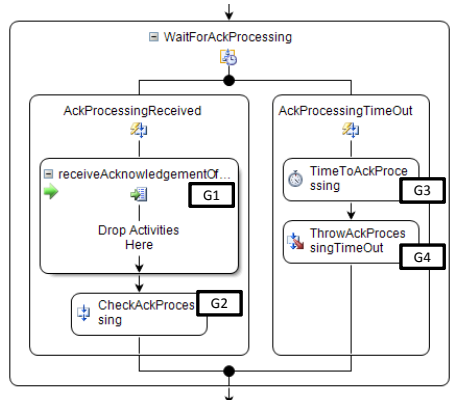
The *receive activity* is followed by an activity that is responsible for checking the contents of the received acknowledgment (F2). Similar to the service contract for business signals, these checks may be identical for the same type of business signal across different business transactions. Thus, we propose to implement the required checks and constraints in a custom activity type. The *custom activity check ack receipt* may then be re-used in different WF business service interfaces. If, by any reason, checking the acknowledgment of receipt fails, the custom activity throws an exception.

In the right branch, the first activity is a *delay activity* (F3). It monitors the agreed time to acknowledge receipt. If exceeded, the *delay activity* triggers a time event which makes the *listen activity* activating the right branch (and consequently deactivating the left branch). In this case, the business transaction has to be re-started due to a time-out exception. In order to re-start the business transaction the current run has to be canceled and the condition of the *while activity*, which monitors the retries, has to be evaluated again. This behavior is accomplished by the *throw activity* (F4) following the *delay activity*. The *throw activity* actuates a *time-out detected exception* that is caught by a fault handler attached to the *while activity*. Please note that handling faults is not covered in this paper.

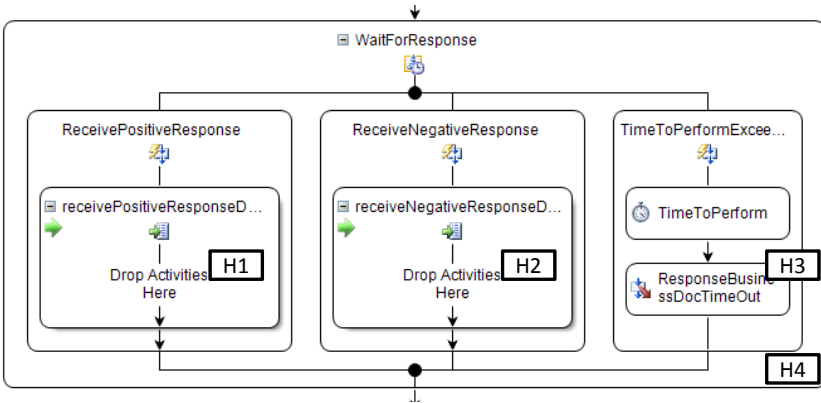
**Step G: Waiting for the acknowledgment of processing.** In this step the initiator expects an *acknowledgment of processing* as shown in figure 3. It



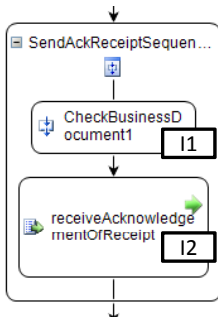
**Fig. 2.** Step F: Waiting for the acknowledgement of receipt



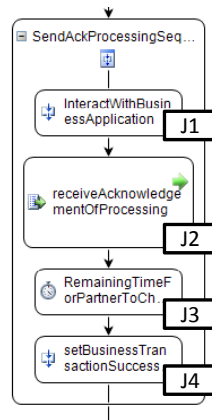
**Fig. 3.** Step G: Waiting for the acknowledgement of processing



**Fig. 4.** Step H: Waiting for the response document



**Fig. 5.** Step I: Sending the acknowledgement of receipt



**Fig. 6.** Step J: Sending the acknowledgement of processing

confirms that the request document was successfully handed over to the responder's business application for further processing. This implies that the business document was delivered to the business application, where it passed additional validation rules.

In terms of the activity flow, handling acknowledgments of processing and their contingent time-outs is similar to the tasks processing acknowledgments of receipts. In the left branch, the steps G1 and G2 model the reception and the checks for a received acknowledgment, whereas the right branch (G3 and G4) handles the time-out. The agreed time-out monitored by the *delay activity* (G3) corresponds to the time to acknowledge processing as defined by the UMM business transaction. The acknowledgment of processing affirms the initiator that the responder is able to process the request document and will respond to it.

**Step H: Waiting for the response document.** Similar to steps F and G, waiting for the response document is implemented by a *listen activity* (H). In case of handling response documents two or more branches are required. Since we may expect time-outs for business documents as well, we define one branch for monitoring the maximum agreed time limit as agreed in the UMM model. The cutout in figure 4 shows that the right branch keeps track of the time limit. If no response document is received within the agreed time to perform, the *delay activity* (H3) triggers a time event and the *throw activity* (H4) terminates the current cycle.

Two-way UMM business transactions support one to many response document types. Consequently, a business service interface requires one to many branches for the receiving business documents - one for each business document type. In our example, we specify two possible response documents - one representing a positive response and the other one a negative response to the request document. Accordingly, the business service interface requires two branches to receive both business document types (see figure 4). A positive response triggers the execution of the left branch containing the *receive activity* H1. Similarly, the *receive activity* H2 listens to negative response documents.

**Step I: Sending the acknowledgment of receipt.** Before the receipt of the response document is acknowledged, the business service interface needs to perform grammar-, schema-, and sequence validation. Since these are generic validation routines we employ the concept of custom activities. If the business document passes the checks in step (I1 in figure 5), the *send activity* (I2) confirms the successful receipt by communicating an acknowledgment of receipt to the responder's BSI.

**Step J: Sending the acknowledgment of processing.** After the proper receipt of the response document is affirmed, the initiator's BSI hands over the document to the business application for further processing. We assume that the business application hosts the business service interface. Therefore, we implement the communication between those systems using a *call external method activity* (J1 in figure 6). Once the business application is delivered, the business application verifies that the document is processable according to pre-defined business rules.



If no exception is thrown by the business application, the BSI sends an acknowledgment of processing (J2) denoting that the verification was successful. The following *delay activity* (J3) keeps the business transaction alive in order to allow the responder to issue a time-out exception or a failed business control exception. The former is communicated by the responder, if the acknowledgment of processing is not received in time by its business service interface. The latter one is thrown, if an acknowledgment is received, which is not processable. How long the business transaction is kept alive is calculated by adding the *time to acknowledge processing* of the response document to the time when the receipt of the response document was acknowledged. Finally, the custom activity J4 sets the variable `businessTransactionSuccessful` to true, so that the condition of the *while activity* (C) is not met any longer.

**Step K: Checking the retry count.** Before the business transaction is eventually finished, the business service interface must check if the *retry count* has not exceeded. Note, the loop continues until the retry count is equal or greater than zero. If the retry count is decremented to -1 at the end of the last attempt, the condition of the *while activity* is not met any longer and the control flow reaches step K. Therefore, the *if/else activity* in step K queries if the retry count is greater or equal to zero. If true, the business transaction was evidently successful and the execution of the business service interface finishes. Otherwise, a *retry count exceeded exception* is thrown and the business transaction failed. Due to space limitations, we do not include a figure for this step.

### 3 Conclusion

UMM is a platform-independent modeling language for collaborative business processes. In order to deploy UMM models to specific target platforms corresponding mappings have to be defined. This paper contributes a UMM to Windows Workflow binding. The generated code is in fact ready to compile. Before executing the workflow, the following tasks remain: (i) declarative configurations of service endpoints, (ii) hosting the workflow, (iii) binding its internal interfaces to a business application.

### References

1. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module, Public Draft V2.0 (2008)
2. Hofreiter, B., Huemer, C., Liegl, P., Schuster, R., Zapletal, M.: Deriving executable BPEL from UMM Business Transactions. In: Proc. of the IEEE Intl. Conf. on Services Computing (SCC 2007). IEEE CS, Los Alamitos (2007)
3. Zapletal, M.: Deriving business service interfaces in Windows Workflow from UMM transactions - long version. Technical report, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria (2008), [http://publik.tuwien.ac.at/files/PubDat\\_166624.pdf](http://publik.tuwien.ac.at/files/PubDat_166624.pdf)

# From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL

Man Zhang and Zhenhua Duan

Institute of Computer Theory & Technology, Xidian University,  
Xi'An, 710071, P.R. China  
zhangman705@gmail.com, zhhdian@mail.xidian.edu.cn

**Abstract.** The Business Process Execution Language for Web Services (BPEL) has emerged as the de facto standard for implementing business processes. At the same time, Model Driven Architecture (MDA) is being applied to the field of business process engineering by separating business logic from the underlying platform technology. However, due to the challenge of mapping graph-oriented modeling languages to block-structured ones and the informal description of UML 2.0 Activity Diagram (AD) and BPEL, transforming AD models to executable BPEL code is not trivial. This paper proposes an approach to transform AD to BPEL and paves the way for further general transformation between graph-oriented and block-structured process modeling languages.

## 1 Introduction

The Business Process Execution Language for Web Services (BPEL) [1] has emerged as the de facto standard for implementing business processes. At the same time, Business Process Management (BPM) has made traditional process-aware information systems completely distributed, global and closely integrated with Web services. It can be concluded that in the near future a wide variety of process-aware information systems will be realized using BPEL. On the other hand, Model Driven Architecture (MDA) is being applied to the field of business process engineering by separating business logic from the underlying platform technique. The core paradigm of MDA is the model transformation from Platform-Independent Models (PIM) to Platform-Specific Models (PSM). Particularly, in BPM's perspective, PIM is established through business process modeling languages while PSM is represented as runtime platform executable code, such as BPEL.

Complete business process models consist of a process model to describe the execution logic, an information model for the data types, an organizational model with some involved roles, and possibly other models [2]. In this paper, we focus on the execution logic, i.e. a model's control flow.

Graph-oriented BPM languages have become mature and been utilized by many existing systems. Among them, UML 2.0 Activity Diagram (AD) [3] has

attained wide adoption and is supported by abundant software tools. There have been some research work on transforming AD into executable BPEL code [2,4,5,7,8,9]. However, they are suffering from the following limitations:

1) They need human intervention to identify activity patterns in AD models. Some approaches define UML profiles specific for structured activities in BPEL. For instance, UML 2.0 profile for BPEL proposed in [4] enables the graphical representation of BPEL with UML 2.0. However, it uses heavily stereotypes and requires quite amount of manual work. Another UML 2.0 based mapping of mobile processes to executable BPEL described in [5] can automatically identify two structured activity patterns, but for others it still needs human intervention.

2) They are not applicable to AD models with arbitrary topology. Most of such methods can handle only a limited subset of AD. Various restrictions are imposed on the structures of models, of which the most restrictive one is *structured limitation* [6]. Techniques for translating unstructured flowcharts into structured ones have been used to translate unstructured process diagrams into equivalent structured ones in [2,7,8]. However, these methods only address a piece of the puzzle of the transformation. Once parallel split and synchronization are involved these techniques are not efficient.

3) They do not make full use of two modeling structures of BPEL: block-structured and graph-based ones. Mendling et al. summarized four strategies of transformation of graph-oriented process modeling languages into BPEL in [9]: the first and second ones transform acyclic models by relying intensively on flow and control links, and the third one identifies block-structured patterns and folds them incrementally. Although the fourth strategy tries to use both of them, no concrete algorithms are presented. In addition, all of the four strategies have restrictions on the structures of source models.

These limitations are not surprising since transforming AD models to BPEL code is not trivial work. First, AD and BPEL belong to two fundamentally different classes of languages: AD is graph-oriented, allowing links between nodes in arbitrary topology, while BPEL is mainly block-structured; Secondly, model transformation of full AD requires formal semantics to express its meaning with greater precision than it is available today. Similarly, formalization of BPEL is also in the initial state of development [7].

In this paper, we propose a novel transformation to achieve completeness, automation, and full exploitation of the two modeling structures of BPEL under some conditions. Here completeness means that any meaningful AD can be transformed into BPEL code. It should be noted that our AD models only capture the basic control flow patterns defined in [10] in order to avoid the subtleties of full AD. The automation is realized by decomposing an AD model into regions and identifying structural patterns separately. By identifying block-structured patterns as often as possible, the readability of generated BPEL code is improved. With regards to the BPEL behavior model, we simplified the BPEL specification to contain only those elements needed to describe the execution logic extracted from the process model.

This paper is organized as follows. Section 2 introduces the subset of full AD we can handle as the basis of our process modeling language. Section 3 describes the transformation technique in detail. Finally conclusion and future work are outlined in the last section.

## 2 Semantically Sound AD Models

Here, we select a subset of AD's meta-model, in which unnecessary elements for modeling the control flow have been removed. According to the meta-model, an AD model consists of ControlFlows, ExecutableNodes (which refer to the necessary basic activities in BPEL), and ControlNodes. Six types of ControlNodes are distinguished: InitialNode, FinalNode, DecisionNode, MergeNode, ForkNode and JoinNode. Here the semantics of all the elements are inherited from the token flow semantics UML 2.0 adopts [3]. Furthermore, we impose some restrictions: DecisionNodes can only start a single flow and MergeNodes always have only one active flow arriving; ForkNodes must start all its concurrent flows and JoinNodes can only be triggered by the arrival of all its incoming flows. These restrictions imply that our AD models only capture the basic six control flow patterns, as mentioned above. In the following, without loss of generality, we assume that all AD models are structurally sound, i.e., they contain exactly one InitialNode and one FinalNode and for every node, there is a path from the InitialNode to the FinalNode going through this node.

When trying to achieve completeness, we should be convinced that transforming process models which cannot be executed normally in real world is meaningless. Based on this point, we introduce the property "semantically sound".

**Definition 1 Semantically sound.** *A structurally sound AD model is semantically sound if and only if all its possible executions terminate successfully. An execution terminates successfully if no other tokens are present in the process model as soon as the FinalNode consumes a token [11].*

To capture the common structural characteristics of semantically sound AD models, we first induce the concept *single-entry-single-exit(SESE) region*. Informally speaking, an SESE region is a set of nodes and edges such that there is exactly one entry edge entering the region, and exactly one exit edge leaving the region. We can decompose an AD model using the technique of SESE decomposition [12] into SESE regions. For an SESE region R, we represent the regions immediately enclosed in R as the child regions of R. For the decomposition to be unique, every region should be canonical. Roughly speaking, a canonical region requires that if a region contains child regions all in sequence it must contain child regions as many as possible; see cf. [12] for precise definition for SESE and canonical region.

Given the unique decomposition of a semantically sound AD model, every region can fall into one of the following three categories: 1) **sequential region**: a region having no ForkNodes and JoinNodes as its ControlNodes; 2) **parallel region**: a region having no DecisionNodes and MergeNodes as its ControlNodes, and no ControlFlows constructing a cycle; 3) **overlapped region**: we use the

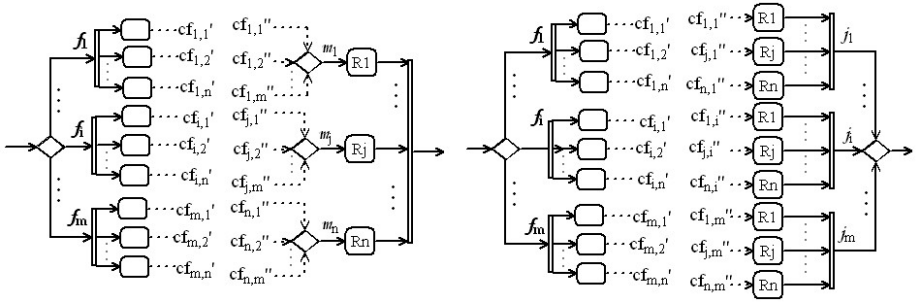


Fig. 1. A pseudo overlapped region and its functionally equivalent structured form

left part of Figure 1 to depict it, which shows a pseudo region whose partial control flows are under determination. It should be noted that when we identify the category of a region, its child regions are treated as ExecutableNodes and their internal structures are out of account.

The round rectangles represent child regions. As the figure shows, there are  $m$  ForkNodes and  $n$  MergeNodes, denoted by  $f_i (i = 1, 2, \dots, m)$  and  $m_j (j = 1, 2, \dots, n)$  respectively. Two types of dotted edges exist: edges without an arrow denoted by  $cf'$  and edges with an arrow denoted by  $cf''$ . Let  $CF'_i$  be the set of  $n$  outgoing edges of  $f_i$ ,  $CF''_j$  be the set of  $m$  incoming edges of  $m_j$ ,  $CF' = \cup_{i=1,2,\dots,m} CF'_i$  and  $CF'' = \cup_{j=1,2,\dots,n} CF''_j$ . The set of control flows under determination, denoted by  $CF$ , are decided by connecting one of edges in  $CF'$  to one of edges in  $CF''$ . If  $CF$  is formed according to a family of bijective functions  $g_i : CF'_i \rightarrow \{cf''_{1,i_1}, cf''_{2,i_2}, \dots, cf''_{j,i_j}, \dots, cf''_{n,i_n}\} (i = 1, 2, \dots, m, j = 1, 2, \dots, n)$  such that for any  $g_k$  and  $g_l (1 \leq k \leq m, 1 \leq l \leq m) k_j \neq l_j$ , the generated region is overlapped.

### 3 Pattern Based Transformation from Semantically Sound AD Models to BPEL Code

Given the unique decomposition of a semantically sound AD model, the transformation can be iteratively operated on the model. For a region, the transformation only focuses on it and its child regions. Once its pattern has been identified, the relevant BPEL code for it will be generated, and then the region will be reduced to an abstract node. For the clarity of representation, we call the abstract node medi-node, to which the corresponding BPEL code is attached.

For the readability of generated code, we try to transform as many regions as possible into block-structured BPEL activities, while the remaining regions could be isolated to limited ranges. Therefore we establish four kinds of patterns: structured pattern for block-structured BPEL activities, unstructured sequential, unstructured parallel and overlapped pattern. The last three ones are applied to those regions the first pattern cannot handle. In the following, transformation methods for the four patterns will be explained in turn.

**1. Structured pattern.** This pattern is defined for those regions that can be suitably mapped to one of the five structured constructs: Sequence, If, Flow, While and RepeatUntil. They all have directly relevant block-structured activities in BPEL 2.0, hence the mappings are straightforward.

**2. Unstructured sequential pattern.** Those sequential regions which the structured pattern cannot tackle with are handled by this pattern. Informally speaking, this kind of regions contains either improperly nested DecisionNodes and MergeNodes or unstructured cyclic flows, even both. In this paper we use an approach proposed in [13] based on continuation semantics for this pattern. Although this solution is intended to untangle unstructured cyclic flows, study shows that it works well for the unstructured sequential regions without unstructured cycles. With the assistance of continuation semantics, an unstructured sequential region can be mapped to its functionally equivalent BPEL code, which could yield the same output as the original model when provided with the same input data. For space limitation, the detailed algorithm is omitted here.

**3. Unstructured parallel pattern.** This pattern is applicable to those parallel regions which cannot be transformed to Flow activity using the structured pattern. Informally speaking, they have improperly nested ForkNodes and JoinNodes. We could easily tame them by mapping all its child regions to BPEL activities nested in a Flow activity and mapping all the edges to control links. However, analysts or programmers may be confused with the generated code full of control links owing to lack of readability. In order to generate as few control links as possible, we try to preserve two structured patterns: Sequence and Flow as much as possible.

For space limitation, only the main idea is briefly described. We first generate a link for each edge connecting a ForkNode to a JoinNode (The ForkNode's immediately predecessor acts as the link's source and the JoinNode's immediately successor as its target), and then remove the edge from the region. Afterwards we apply SESE decomposition on the altered region over again. If new child regions are produced, they can be matched to Sequence or Flow pattern.

**4. Over-lapped pattern.** As the name shows, this pattern is for over-lapped regions. We turn an over-lapped region into its functionally equivalent structured form by duplicating the medi-nodes between MergeNodes and the JoinNodes and switching these two kinds of ControlNodes. Recall the pseudo region in the left part of Figure 1, its functionally equivalent structured region is shown in the right part. Apparently the premise of transformation is that those control flows under determination are formed according to valid connecting functions. Afterwards the new region can be easily handled using the structured pattern. This method is easy to practice but at the expense of code redundancy.

The whole algorithm starts transformation from those regions only having one ExecutableNode by turning them into medi-nodes, and then gradually folds outer regions into medi-nodes by matching them to the four patterns above, until there is only one single medi-node left. At that time the BPEL code attached to it is the desired resulting code.

## 4 Conclusion and Further Work

In this paper, we presented the transformation of AD process models into executable BPEL code. First, we make our transformation complete by exploiting the characteristic of semantically sound AD models. Secondly, by adopting SESE decomposition and separate pattern based analysis the transformation process is automated. Thirdly, two modeling structures of BPEL are fully utilized in the transformation. Furthermore, we emphasize the readability of generated code throughout the process. In the future, we will try to extend our method to more general control-flow models.

## References

1. OASIS: Web Services Business Process Execution Language Version 2.0 (April 11, 2007), <http://docs.oasis-open.org/wsbpel/2.0/0S/wsbpel-v2.0-0S.html>
2. Hauser, R., Koehler, J.: Compiling Process Graphs into Executable Code. In: Kar-sai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 317–336. Springer, Heidelberg (2004)
3. OMG: UML 2.0 Superstructure (8/8/05), <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>
4. Ambühler, T.: UML 2.0 Profile for WS-BPEL with Mapping to WS-BPEL. Universität Stuttgart (2005)
5. Pajunen, L., Ruokonen, A.: Modeling and generating mobile business process. In: Proc. ICWS 2007 (2007)
6. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789. Springer, Heidelberg (2000)
7. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal 44(1), 47–65 (2005)
8. Zhao, W., Hauser, R., Bhattacharya, K., Bryant, B.R., Cao, F.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. IJWGS 2(1), 68–91 (2006)
9. Mendling, J., Lassen, K.B., Zdun, U.: Transformation strategies between block-oriented and graphoriented process modelling languages. In: Multikonferenz Wirtschaftsinformatik 2006. Band 2, pp. 297–312 (2006)
10. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPM-center.org (2006)
11. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: An incremental approach to the analysis and transformation of workflows using region trees. IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and reviews 38(3) (May 2008)
12. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through sese decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
13. Koehler, J., Hauser, R.: Untangling Unstructured Cyclic Flows – A Solution Based on Continuations. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 121–138. Springer, Heidelberg (2004)

# Batch Invocation of Web Services in BPEL Process

Liang Bao, Ping Chen, Xiang Zhang, Sheng Chen, Shengming Hu, and Yang Yang

Software Engineering Institute, Xidian University.

Xi'an, 710071, China

{baoliangbox, zximportant, kkchensheng, yyqinghuan}@gmail.com,

{chenping, shmhu}@sei.xidian.edu.cn

**Abstract.** This paper presents our approach for optimizing the execution of BPEL (Business Process Execution Language) process by leveraging the features of enterprise intranet and introduces *batBPEL*, a tool for batch invocations of web services in BPEL process. The approach focuses on the decrease of connections by forming batch invocation request of web services in BPEL process. Some empirical experiments and evaluations show and prove the efficiency of our method and related algorithms.

## 1 Introduction

Nowadays, Web services are emerging as a prevalent paradigm for implementing the SOA [1] concept by developing and deploying business processes within and across enterprises. Since the agility and flexibility of business processes become more and more important, the Web Services Business Process Execution Language [2] (BPEL for short) is in the very act of becoming the industrial standard for modeling Web services based business processes.

As an increasing number of business processes are modeled using BPEL, it is critical to guarantee the execution performance of a business process. Many optimization methods have been introduced, such as IBM Symphony project for decentralized process execution [3] and our earlier work for data-race free optimization [4].

However, all of these methods mentioned above are process-centered and fail to leverage the special features (such as operating system, networking etc.) that exist within the environment of an enterprise or organization to gain better performance. One of these features is that most applications are running on a fast and reliable intranet network. This implies that the conclusion “the time-cost to establish a new connection is far more expensive than that of transmitting a large trunk of data” holds under this circumstance. In this paper, we design and implement a framework named *batBPEL* (batch BPEL) that leverage this straightforward but important conclusion to speed up the execution of a BPEL process. The approach focuses on the decrease of connections by forming batch invocation request of web services in BPEL process. Some actual experiments prove that this solution can increase the throughput and decrease the execution time of different processes significantly.



## 2 Overview

In this section, we introduce our overall design of *batBPEL*. The architecture illustrated in Fig. 1 describes the basic structure and interactions of *batBPEL*, which comprises the following components:

*Static Analyzer*: this component applies analysis techniques to a given BPEL process and forms the batch groups.

*Invocation Interception*: its task is to intercept the invocation of Web services in BPEL process, and to look up the batch groups and notify the client side service agent with the invocation and batch information.

*Client Side Service Agent (Client Agent)*: after receiving the information sent by invocation interception, the client side service agent forms a batch invocation request.

*Server Side Service Agent (Server Agent)*: when the request is received, the server side service agent un-marshals the request, identifies, separates and dispatches the request into many web service invocations.

*Invocation Result Cache*: once receives the response, the client agent un-marshals and separates it. The invocation result required at this time is returned to the BPEL engine directly, and other results are put into the invocation result cache for later use.

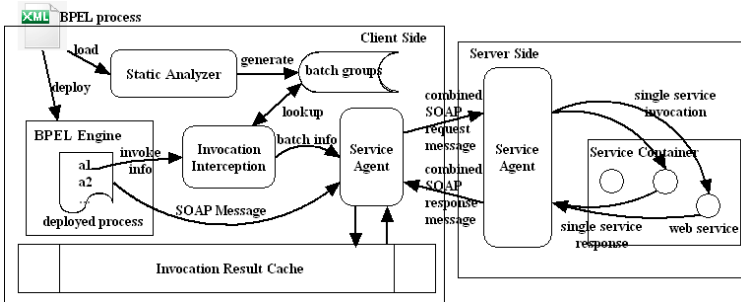
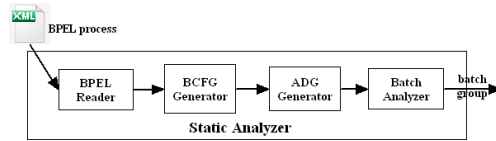


Fig. 1. The architecture of *batBPEL*

## 3 Static Analysis

Before a BPEL process is executed in the process engine, it is loaded into a static analyzer first. The analyzer consists of four modules, namely “BPEL Reader”, “BCFG Generator”, “ADG Generator” and “Batch Analyzer” respectively, which is shown in Fig. 2. When a BPEL process is loaded, the BPEL Reader will read it and transform it into our memory structure of BPEL process (i.e. customized Java classes), which is implemented by JAXB (Java native XML APIs in JDK 6.0). The whole analysis procedure is totally sequential, the detailed discussion can be found in [5].

*BCFG Generator*: The key task of this component is to load the memory representation of BPEL process generated by BPEL reader and transform it into a corresponding BCFG (BPEL Control Flow Graph) representation.



**Fig. 2.** Static analyzer

*ADG Generator:* Load the memory representation of BCFG and transform it into a corresponding ADG(Activity Dependence Graph) representation.

*Batch Analyzer:* Enforce static slicing algorithm [6] to the ADG formed above, generate a legal sequence of execution(a sample) and apply our batch algorithm to it [5].

## 4 Invocation Interception

When an *invoke* activity in a batch group is executed, *batBPEL* need to be notified with related invocation information. This is archived by introducing an invocation interception module in it. The implementation of such interception is based on a modified version of BPEL engine named ActiveBPEL [7], which uses AspectJ [8] to introduce invocation interception as cross-cutting concerns(inspired by [9]).

Since we are only interested in the Web service invocation, we have decided to intercept the process when it performs *invoke* activities. In order to do so we set pointcuts before and after the engine calls the execute method on these activities.

## 5 Service Agent

By definition, the main responsibilities of the service agents(both client and server agent) are two folds: (1) pack one or more invocation information up and form a single invocation request and (2) read and parse the packaged invocation request, dispatch the different invocations(for server agent) or save the different results in the invocation cache(for client agent).

### 5.1 Client Side Service Agent

Client side service agent(client agent for short) is located in the same host as the process engine is. The client agent is structured around four key components: (i) request interception, (ii) batch marshaller, (iii) batch un-marshaller and (vi) service invocation cache reader/writer.

The core function of request interception component is SOAP request monitoring and interception.In *batBPEL*, it is implemented by Apache Synapse, which is an open source XML and Web services management and integration broker that can form the basis of an SOA and Enterprise Service Bus (ESB). We use it here to perform SOAP message interception.

## 5.2 Server Side Service Agent

Server side service agent(server agent for short) is located in the same host as the service container is. Fig. 3 gives a detailed architecture of it. The batch un-marshaller and batch marshaller are both mediators implemented in Java class, just like their counterparts in client agent. When a batch invocation arrives, the batch un-marshaller and invocation dispatcher are added to separate and dispatch the batch invocation request to desired web services. The whole process is totally concurrent. Once all of these invocations are returned, the related results are collected and formed a single response by marshaller. It is then pushed to the intranet as a batch invocation result.

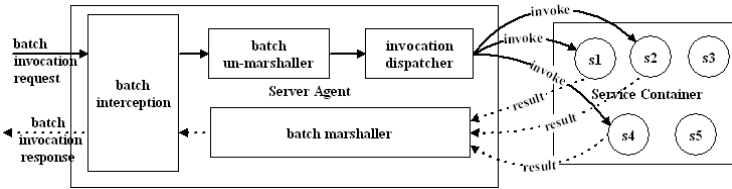


Fig. 3. Server agent

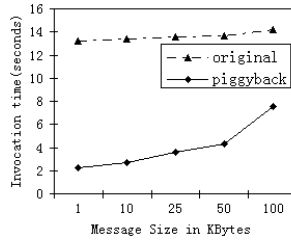
## 6 Experiment and Evaluation

In this section we report the comprehensive performance gain that brought by batch invocation from both the service and BPEL process level.

The typical setting for evaluation is a cluster of Intel Pentium based Windows machines(2.8G, 512MB RAM) connected by a 100 Mb/s LAN. Fig. 4 shows the comparison of the execution time duration on the service level between the batch method and the conventional invocation method. The duration presented in the figure marked *original* includes fifty invocations. Every five invocations are packed into one invocation when batch method is exploited, and accordingly, the client invokes the server for ten times. When the message size is less than 50 KBytes — the most case for message size, the batch method is about four times fast than the traditional method. The acceleration factor deteriorates slightly when the message size is bigger than 100 KBytes as data marshal and un-marshall take relatively longer time.

The duration for executing a service, which varies from many seconds to many hours or even days, and the impact of it that will be explored statically, is not taken into account in Fig. 4. It is worth nothing at this point that *batBPEL* that utilizes batch invocation can speed up the execution of BPEL process appreciably.

Table 1 displays the process level comparisons between these two invocation paradigms with the focus on invocations and execution duration. Among all the processes we have investigated, six of them are represented. These processes can be mainly categorized into three types: computation-intensive, service-invocation-intensive and the compound of them. The Office Automation(OA) and Draining System(DS) process belong to the first type; the Tool Integration(TI) and Travel Reserve(TR) fall into the second category; and the other two processes — Online Book Purchase(OBP) and Train Tickets(TT) pertain to the last type.



**Fig. 4.** Service level comparison

**Table 1.** Process level comparison

process	original invokes	batch invokes	original duration(s)	batch duration(s)
OA	79	63	231.23	207.5
DS	103	81	287.93	242.7
TI	183	144	436	315
TR	94	71	108.9	97.4
OBP	23	22	34.63	33.1
TT	52	39	60.83	45.2

As what can be observed in table 1, while the batch invocation opportunities for the first type is minor, it increases pronouncedly for the second type, which is what BPEL targets for. Take the TI process for example, when it is executed conventionally, which incurs one service invocation when an *invoke* activity [2] is interpreted, 183 invocations occurred. The batch invocation paradigm, however, requires only 144 invocations. The performance gain for this process is around 28 percent.

By analyzing all the data gathered between our users and *batBPEL*, we found that the performance gain, which is process dependent, ranges from 5 percent to 30 percent. Moreover, the complex and sophisticated process, such as banking and telecommunication systems that contain thousands of activities and have twisted control flow, enjoy more performance improvement.

## 7 Related Works

So far as we know, there are no related works on this topic. However, there are some works on execution optimization of BPEL process, such as IBM Symphony and our earlier work on *data-race free* optimization of BPEL process. In Symphony project, some techniques partition a composite web service written as a single BPEL program into an equivalent set of decentralized processes, with the goal of minimizing communication costs and maximizing the throughput of multiple concurrent instances of the input program. Our earlier approach focuses on the adapted static optimization methods of BPEL process.

Besides, there has been considerable research effort paid to BPEL. WofBPEL [10] translates BPEL processes to Petri nets and imposes existing Petri nets analysis

techniques to perform static analysis on processes. [11] modifies the CWB to support BPE-calculus by means of PAC to ensure that each *link* has one source and target activity exactly, and to guarantee that the process is free of deadlocks. Mads [12] describes some region-based memory techniques for programs that perform dynamic memory allocation and de-allocation.

## 8 Conclusion and Future Works

In this paper, we advocate batch invocations of Web services in BPEL process by static analysis and dynamic invocation interceptions. Furthermore, we implement our *batBPEL* that support high performance BPEL execution in the enterprise intranet environment. Our further work includes improvement of our static analysis algorithm and a careful study of the system.

## References

1. Papazoglou, M.: Service-oriented computing: Concepts, characteristics and directions. In: 4th International Conference on Web Information Systems Engineering (WISE), New York, pp. 3–12. IEEE Press, Los Alamitos (2003)
2. Jordan, D.: Web services business process execution language version 2.0. OASIS Specification (2007)
3. Nanda, M.G., Karnik, N.: Synchronization analysis for decentralizing composite web services. ACM, New York (2003)
4. Sheng Chen, L.B., Chen, P.: Optbpel: A tool for performance optimization of bpel process. LNCS. Springer, Heidelberg (2008)
5. Bao, L., Zhang, X.: Batch invocation of web services in bpel process(detatiled version) (2008), <http://www.soacn.org>
6. Krinke, J.: Static slicing of threaded programs. In: ACM SIGPLAN/SIGSOFT, pp. 35–42 (1998)
7. Endpoints, A.: Activebpel engine architecture(version 4.1) (2008), <http://www.activebpel.org/docs/architecture.html>
8. Kiczales, G., Griswold, E.H., W.G.: An overview of aspectj. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 327–353. Springer, Heidelberg (2001)
9. Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: International Conference on Software Engineering(ICSE), pp. 69–77. ACM Press, New York (2005)
10. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Wofbpel: A tool for automated analysis of bpel processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
11. Koshkina, M., B.F.: Modelling and verifying web service orchestration by means of the concurrency workbench. TAV-WEB Proceedings/ACM SIGSOFT 29–5(5) (2004)
12. Mads Tofte, J.P.T.: Region-based memory management. Information and Computation 132, 109–197 (1997)

# Formation of Service Value Networks for Decentralized Service Provisioning

Sebastian Speiser, Benjamin Blau, Steffen Lamparter, and Stefan Tai

Karlsruhe Service Research Institute (KSRI), Universität Karlsruhe (TH), Germany  
firstname.lastname@ksri.uni-karlsruhe.de

**Abstract.** The provisioning of complex services requires tight collaboration between diverse service providers and their customers harmonizing supply and demand chains to a highly flexible, dynamic and decentralized service value network. Peers in such a network autonomously delegate (sub-)tasks which cannot be done efficiently by themselves to other more suitable peers in their community. In this paper, we propose an architecture for such service communities that features decentralized service provisioning based on current Web technologies. In this context, we present an algorithm for efficient service value network formation and show by means of a simulation that sufficiently sized service networks can fulfill practically all customer requests. When compared to the optimal (central) case, there is a modest price increase for the customers but the overall welfare decreases only insignificantly.

## 1 Introduction

Complex (or *composite*) services "typically involve the assembly and invocation of many pre-existing services possibly found in diverse enterprises [1]," and thus, a network of service providers and consumers. In services-led economies, these networks increasingly are loosely-coupled configurations of legally independent firms.

The formation of such a network is driven by the value that it generates for its customers. With increasing competition and specialization in the services sector, and the continuous introduction of new services offerings, value-driven network formation and transformation is of predominant importance. Dynamic *service value networks* are often considered as the only strategic alternative to provide complex services [2][3][4].

Service networks describe the possible cooperations between legally independent actors that enable co-generation of value by fulfilling complex customer requests. Due to low lock-in and lock-out costs, service value networks are characterized by a high rate of fluctuation of service providers, making it difficult to maintain a consistent, centralized service repository.

Consequently, this paper analyzes the formation of service value networks in a decentralized service provisioning environment. Based on customers' requests for the completion of complex tasks, peers in such a network autonomously delegate (sub-)tasks to other service providers within their partner networks. This iterative process fosters the evolution of a network-based value generation driven by customers' needs and co-opetition of specialized service providers.

In this paper, we propose an architecture for such service networks that features decentralized service provisioning based on current Web technologies. In this context, we present an algorithm for efficient service value network formation and show by means of a simulation that sufficiently sized service networks can fulfill practically all customer requests. When compared to the optimal (central) case, there is a modest price increase for the customers but the overall welfare decreases only insignificantly.

Several decentralized service discovery mechanisms have been presented in literature. Most of them distribute service descriptions over several peers and use an indexing mechanism to efficiently route the queries through the P2P network (e.g. [5,6]). However, building and maintaining such indexes contradicts our assumption that each peer in the network has only knowledge about his direct neighbors and usually peers are not willing to share this knowledge since it can be an important business asset.

There are a few approaches that explicitly consider a network of providers, e.g. [7]. However, the algorithms assume peers with different capacity (i.e. available resources) but the same homogeneous functionality (such as computing power). Since in our network the services are highly specialized and therefore provide different functionality, these algorithms are not directly applicable.

The paper is structured as follows. In Section 2 we define our model of composed services and service value networks. Based thereon in Section 3 we introduce our proposed algorithms for decentralized service provisioning. Section 4 describes a simulation of the algorithms and compares them to centralized approaches. We conclude and give an outlook on future work in Section 5. An extended version of this paper with more details can be found in [8].

## 2 Service Value Networks

In this section we introduce a formal model that is the foundation for our understanding of service value networks (SVNs). First we define a *service specification* which determines how *atomic services* are combined to form more complex *composed services*. Afterwards service providers and customers are introduced that are connected in a service network. For the fulfillment of service requests the SVNs are dynamically formed and create a value that is defined in the end of this section.

**Service Specification.** Services that cannot be decomposed in smaller sub-services are called *atomic*. In contrast a *composed* service is equivalent to its *component* services that can in turn again be atomic or composed. Let  $S$  be the set of all services. The set of ordered pairs  $E_S$  represents the component relationship meaning that  $s_1 \in S$  is a component service of  $s_2 \in S$ , iff  $(s_1, s_2) \in E_S$ . The directed graph  $G_S = (S, E_S)$  is called *service specification*. We require  $G_S$  to be acyclic. Furthermore we define the two disjoint sets of *atomic* services  $S_a = \{s \in S \mid \nexists s' \in S : (s', s) \in E_S\}$  and *composed* services  $S_c = S \setminus S_a$ . We define the complexity of a service  $s$  as the number of basic services that are required to create an equivalent service.

**Service Provider.** Each service provider  $p$  is able to execute the services denoted by his capability set  $\phi_p \subseteq S$ . For the execution of a service  $s \in \phi_p$  the provider is charged with internal costs, determined by his cost function  $c_p : S \rightarrow \mathbb{R}$ . As a provider aims

at making a profit he charges a price that adds a margin to his costs, given by the margin function  $m_p : \mathbb{R} \rightarrow \mathbb{R}$ . The charged price for  $s \in S$  is  $f_p(s) = m_p(c_p(s))$ . The provider is able to subcontract services to a set of cooperating service providers, denoted as  $V_p$ . The costs for  $p$  when subcontracting  $s$  are given by  $c_p(s) = \min_{p' \in V_p} f_{p'}(s)$ .

A provider  $p$  maintains a function  $d_p : S \rightarrow \{\text{false}, \text{true}\}$  for all services that he can either offer himself or in cooperation. The function returns true for services that should be decomposed and false for services that are executed or delegated as a whole.

**Customer.** Besides service providers we consider the set of service customers  $W$  that consume services but do not provide any services nor forward requests. A customer  $w \in W$  has a valuation for the services he requests, given by his utility function  $u_w : S \rightarrow \mathbb{R}$ . Service requests are sent to  $w$ 's partner network  $V_w \subseteq P$ .

**Service Value Network.** The relationships between customers, providers and among providers are represented by the directed graph  $G = (P \cup W, E)$  where an edge  $(x, y) \in E$  denotes that  $x$  sends requests to  $y$ . The edges are given by the partner networks as  $E = \{(x, y) \in (P \cup W) \times P \mid y \in V_x\}$ . The graph  $G$  is called *service network*. The customer that requests a service and the providers that are involved in the fulfillment of the request form a *service value network*. As this process is invoked for every service request it is a *dynamic formation*.

Let  $P' \subset P \cup W$  be the set of participants in a service value network fulfilling the request for service  $s$  by customer  $w$ . Service delegations are represented as tuples  $(p_1, p_2, s) \in P' \times P' \times S$ , meaning that  $p_1$  delegates service  $s$  to  $p_2$ . Internal executions are also treated as service delegations with  $p_1 = p_2$ . Let  $E'$  be the set of all service delegations then the directed graph  $G' = (P', E')$  denotes the service value network. For the reader's convenience we define the following three sets for a service value network: (I) The customer's request  $R_{G'} = \{(w, p, s) \in E' \mid w \in W\}$ , (II) the internal executions  $I_{G'} = \{(p_1, p_2, s) \in E' \mid p_1 = p_2\}$ , and (III) the set of service cooperations  $D_{G'} = E' \setminus (R_{G'} \cup I_{G'})$ .

**Welfare in Service Value Networks.** The welfare co-generated in a service value network is given by the sum of received values for all participants. For a service value network  $G' = (P', E')$  which serves customer  $w$  with service  $s$ , we define the welfare  $wf_{G'} = u_w(s) - \sum_{(p,p,s') \in I_{G'}} c_p(s')$ . This shows that the welfare can be maximized if the internal costs that occur during the execution of a service are minimized. For each service delegation  $(p_1, p_2, s') \in D_{G'}$  the payment of  $f_{p_2}(s')$  increases  $p_2$ 's value by the same amount as it decreases  $p_1$ 's value. Therefore payments do not influence the welfare, except in the case where the total price is above the customer's valuation meaning that no transaction will take place and thus no value at all is generated.

### 3 Network Formation and Service Delivery Algorithms

The algorithms can be divided into two groups. The first initializes and maintains the data structures a provider keeps to determine the best executions strategies for each service. Based on this data the second group generates concrete offers upon service requests.



Every new provider  $p$  that joins the service value network assigns all services  $s \in S$  to himself with costs of  $\infty$ . Then he updates the services in his capability set to have a price given by his internal costs plus his margin ( $\forall s \in \phi_p : f_p(s) := m_p(c_p(s))$ ). Afterwards the providers notifies his partner network about his capabilities.

A provider  $p$  that receives a notification about the capability of  $p'$  to deliver  $s$  at price  $f_{p'}$ , first checks if the new price is better than his current costs. If this is the case he updates the preferred provider for the service and sets the costs to the received price. He also updates the price  $f_p(s)$  he charges for the service to be the new costs plus his margin. Then he updates his cost structure.

This function is done by first notifying the partner network about the new capability respectively the new price. Then it is checked if the updated service is part of composed services. In that case for each composed service the sum of the prices for the components is compared to the current total price. If a decomposition is cheaper this is saved in the provider's data structure and the cost structure is updated recursively for the composed services.

When a provider  $p$  is requested a concrete offer for a service  $s$  then  $p$  simply adds his margin on top of his costs associated with  $s$ . The associated costs are calculated with the following three cases. If  $s$  is marked for decomposed execution, it costs the sum of the recursively calculated costs of its components. If  $p$  is the preferred provider for  $s$ , the internal costs are taken. Else an offer from the provider that  $p$  has assigned for  $s$  is requested.

**Correctness and Scalability.** Initially a provider assigns all services to himself and memorizes as costs either his internal costs or infinity for services that he is not capable of doing. In the succeeding phase the provider broadcasts his capabilities. We can assume that each provider adds a margin that leads to increasing prices ( $\forall x \in \mathbb{R}, p \in P : m_p(x) > x$ ). Therefore at some point providers will dismiss further notifications based on their price. The notification phase is very similar to the *Routing Information Protocol (RIP)* [9]. With RIP routers on the internet exchange information about which networks they can reach. The costs are measured as the number of intermediary routers that are used by a router to reach the network. The different network speeds are equivalent to the margins charged by providers. RIP has proven to be correct by operating reliably the internet and other networks. However it has some scalability issues and is therefore replaced more and more by link-state based routing protocols. These protocols are not applicable to our problem domain, as they require that providers can gather complete topological information about the service network, including the margins of other providers. This is not a realistic option. The scalability issues will not be a problem in the near future as the service networks are supposed to be much smaller than the internet.

Routers operating with RIP resend their routing information every 30 seconds and delete routes that are not confirmed by such resendings. In this way the protocol deals with the removal of links or routers. This can also be applied to our algorithms in order to react to price increases or false advertisements. False advertisements are notifications of providers that they can deliver a service for a given price but always return higher prices in the offer phase. Other providers can detect such a behavior and remove the provider from their partner network.

In the offer making phase a provider knows exactly if he should decompose a service and to which other providers services are delegated. This efficiency for the frequent service offers is bought with increased costs for the propagation of changes in the service value network.

## 4 Network Simulation

We ran simulations of service value networks in order to compare them to an approach with a central registry. The following questions are analyzed with the simulation results:

1. What is the performance of the notification algorithm?
2. How many service requests can be fulfilled?
3. How large is the price increase for customers?
4. What is the impact on the welfare?

For the experiment we first create a service specification consisting of  $n_s$  services. Each of the  $n_p$  providers is capable of doing a randomly selected service  $s$  with internal costs randomly selected between  $0.2 \cdot \text{complexity}(s)$  and  $0.8 \cdot \text{complexity}(s)$ . It is insured that every service has at least one provider. A provider  $p$  is assigned the margin function  $m_p(x) = (1 + M_p) \cdot x$ , where  $M_p$  is randomly chosen for every provider between 10% and 20%. The partner network is created by establishing  $0.05 \cdot n_p^2$  random partnerships. The customer's valuation of a service  $s$  is given by  $\text{complexity}(s)$ .

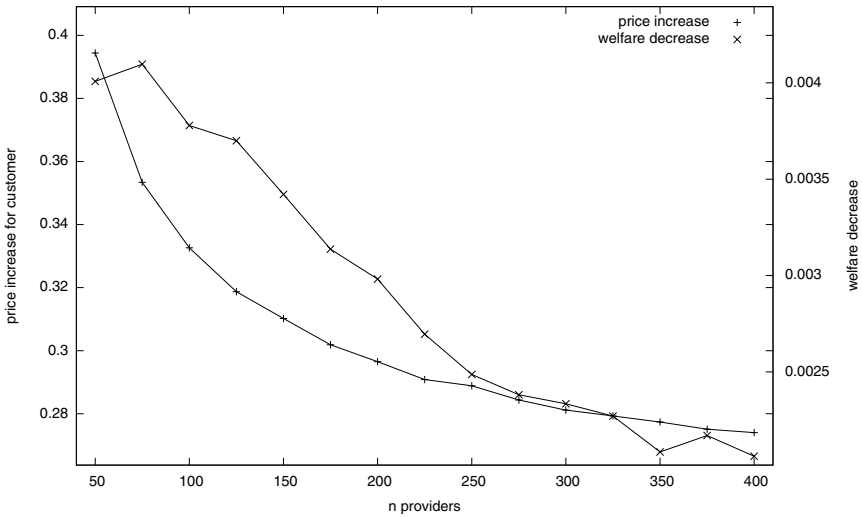
For the evaluation of our approach we compare it to the case where a central service registry exists. This can be modelled as a provider that has partnerships with all other providers and charges no margin. The registry is able to serve a customer all services with optimal price. In some comparisons we assume that the registry also knows about the internal costs of individual providers and can therefore provide service executions with optimal welfare.

We ran the experiment with  $n_s = 50$  services and varied the number of providers from 50 to 400 in steps of 25. We repeated this 100 times for each  $n_p$  and based on the average values, we came to the following results.

**Performance of Algorithm (Question 1).** The number of notifications that are sent between providers until all service pricing information is exchanged grows with speed of  $O(n^3)$ . As the number of providers also grows, this means an average number of  $O(n^2)$  notifications per provider. Therefore the algorithm can be considered efficient.

**Decentral Execution Ratio (Question 2).** The decentral execution ratio is the probability that a request to a random provider for a random service results in a price that is lower than the valuation of the service. This ratio converges fast to the optimal value of 100% and is even for small numbers of providers ( $n_p \leq 100$ ) in an acceptable range between 80% and 90%.

**Price Increase for Customer (Question 3).** We see in Figure 1 that the price increase for a customer requesting a random service from a random provider gets smaller with increasing size of the service network. In our simulation there is an inherent reason why



**Fig. 1.** Price increase for customer and welfare decrease for varying  $n_p$  from 50 to 400

a zero price increase is not possible: every provider can execute himself only one service and therefore has to delegate all other services, which means for almost all requests at least two providers are involved that both charge a margin. In the central case we assumed that the registry does not charge money and can therefore offer the best available price. The operation of such a central registry however is associated with costs that have to be reimbursed either by charging a margin or receiving payments from the service providers which will increase their costs. Therefore we conclude that with increasing service network size the prices get more competitive compared to a central scenario.

**Welfare Decrease (Question 4).** The participants in a service network are self-interested. The proposed algorithms aim at keeping prices low in order to stay competitive while ensuring that a given margin is earned. Our main concern is how the welfare of a decentral formed service value network is compared to a centrally planned cooperation. We observed in the simulation that the average welfare decrease over all services and providers behaves similar to the price increase (see Figure 1). However the decrease is always very small (below 0.6%) even for small networks.

We showed that the algorithm is efficient and delivers results that are competitive to a central approach with a registry that operates for free, which is optimal but unrealistic. Although customers have to pay slightly higher prices, practically all requests can be fulfilled without a significant welfare decrease. We also observed that a larger service network is better both for the customer and the overall welfare.

## 5 Conclusion

In this paper we considered the problem of decentralized service value network formation. We provided an algorithm that distributes a service request over a network of

self-interested, non-cooperative service providers and thereby creates an efficient service value network. The algorithm is novel compared to existing approaches as peers in the network do not have to provide any information about their business network to their customers. Thereby, new business models for service intermediaries are enabled, whose only business asset is a strong partner network. We showed by means of a simulation that the algorithm is tractable for reasonable sized scenarios as the number of required notifications in a network with  $n$  providers is  $O(n^3)$ . In addition, the results show that the algorithm performs quite well in terms of welfare decrease and price increase compared to a central scenario. In fact, the total loss in welfare is below 0.6%.

There are several directions in which we plan to extend this work. First, we plan to replace the current uniform distribution used to create the service network with a power-law distribution which seems to be a more realistic assumption for social networks as well as Web environments [10]. We plan an analytical and experimental evaluation how this change impacts the performance of our algorithms. Second, we plan to extend the algorithms for quality of service aspects. Modeling the trade-off between quality and price requires the introduction of multi-attribute price and value functions. Third, we plan to assess whether introducing a market mechanism such as a path auction might further increase the efficiency of the service allocation.

**Acknowledgement.** This work was partially funded by the German Research Foundation (DFG) in scope of the Graduate School Information Management and Market Engineering.

## References

1. Papazoglou, M.: Web Services: Principles and Technologies. Prentice Hall, Englewood Cliffs (2007)
2. Tapscott, D., Lowy, A., Ticoll, D.: Digital Capital: Harnessing the Power of Business Webs. Harvard Business School Press (2000)
3. Hagel III, J.: Spider versus Spider. *The McKinsey Quarterly* (1), 4–5 (1996)
4. Steiner, F.: Formation and Early Growth of Business Webs: Modular Product Systems in Network Markets. Physica-Verlag, Heidelberg (2004)
5. Schmidt, C., Parashar, M.: A peer-to-peer approach to web service discovery. *World Wide Web Journal* 7(2) (2004)
6. Vu, L.H., Hauswirth, M., Aberer, K.: Towards P2P-based Semantic Web Service Discovery with QoS Support. In: Workshop on Business Processes and Services (BPS), pp. 18–31 (2006)
7. de Weerd, M., Zhang, Y., Klos, T.: Distributed task allocation in social networks. In: AA-MAS 2007, pp. 1–8 (2007)
8. Speiser, S., Blau, B., Lamparter, S., Tai, S.: Formation of service value networks for decentralized service provisioning. Technical report, KSRI, Universität Karlsruhe, TH (2008)
9. Hedrick, C.: Routing Information Protocol. RFC 1058 (Historic) (June 1988) Updated by RFCs 1388, 1723
10. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM 1999, pp. 251–262 (1999)

# Towards Automated WSDL-Based Testing of Web Services\*

Cesare Bartolini<sup>1</sup>, Antonia Bertolino<sup>1</sup>, Eda Marchetti<sup>1</sup>, and Andrea Polini<sup>1,2</sup>

<sup>1</sup> Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"  
Consiglio Nazionale delle Ricerche  
Via Moruzzi 1 - 56124 Pisa, Italy

<sup>2</sup> Dipartimento di Matematica ed Informatica  
University of Camerino  
Via Madonna delle Carceri, 9 - 62032 Camerino, Italy  
{cesare.bartolini, antonia.bertolino, eda.marchetti,  
andrea.polini}@isti.cnr.it

**Abstract.** With the emergence of service-oriented computing, proper approaches are needed to validate a Web Service (WS) behaviour. In the last years several tools automating WS testing have been released. However, generally the selection of which and how many test cases should be run, and the instantiation of the input data into each test case, is still left to the human tester.

In this paper we introduce a proposal to automate WSDL-based testing, which combines the coverage of WS operations with data-driven test case generation. We sketch the general architecture of a test environment that basically integrates two existing tools: soapUI, which is a popular tool for WS testing, and TAXI, which is a tool we have previously developed for the automated derivation of XML instances from a XML Schema.

The test suite generation can be driven by basic coverage criteria and by the application of some heuristics, aimed in particular at systematically combining the generated instance elements in different ways, and at opportunely varying the cardinalities and the data values used for the generated instances.

## 1 Introduction

Service-oriented Architecture (SOA) is the emerging paradigm for the development of distributed applications that are easy to integrate and flexible to fast changes of the environment and of user needs.

The escalation of Web Service (WS) technology is now evident to everyone. All major IT vendors, such as IBM, Tibco, Software AG, Oracle, just to cite the top competitors, have made huge investments into SOA in the last years.

Moreover service providers from virtually any domain, banks, governments, hospitals, academies, travel agencies, and so on, are progressively shifting towards the on-line service-market.

---

\* The authors wish to thank Antonino Sabetta for his help in defining the test cases. This work was supported by the TAS<sup>3</sup> Project (EU FP7 CP n. 216287).

The net effect of this IT technology trend is that unavoidably business and social welfare are more and more depending on the proper functioning of services delivered over the Net. WS trustworthiness is a modern buzzword to qualify those characteristics that allow a client to put justified reliance on a provided service, against accidental or intentional faults. Because of their pervasive distribution, WSs must offer very strict guarantees in this regard, even for services that are not dealing with safety-critical or money-critical applications.

For this reason, it is imperative that WSs are thoroughly tested before deployment. Essentially, a WS collects a set of functions, whose invocation syntax is defined in the associated WSDL document. The adoption of open standard specifications for the WS interface has been instrumental to achieve interoperability and is at the basis of several available testing tools. The WSDL formalized description of service operations and of their input and output parameters can be in fact taken as a reference for black box testing at the service interface. In the last years a wealth of WSDL-based WS test tools has been developed [5,8]. In general such tools can automatically derive skeletons of WS test cases and provide support for their execution and result analysis. Nevertheless they do not provide support for input data selection, for which they still rely on the human tester's intervention.

To us, it is somewhat surprising that till today WS test automation is not pushed further than this, since in principle the XML-based syntax of WSDL documents could support fully automated WS test generation by means of traditional syntax-based testing approaches. In this direction, we have defined a framework for "turn-key" generation of WS test suites<sup>1</sup>, in which we combine coverage of WS operations (as provided by soapUI) with data-driven test case generation.

In this paper we illustrate the feasibility of the idea by means of a proof-of-concept implementation that integrates soapUI and TAXI. The latter is a tool we have previously developed [9] for the automated derivation of XML instances from a XML Schema. The idea, in comparison with soapUI and other existing WS test tools, is to derive from the WSDL interface of a WS, in a completely automated way, a test suite that thoroughly exercises the WS operations by systematically varying the possible input message structures and values.

The paper is structured as follows. In the next section we present the envisaged approach to fully automated WSDL-based testing; in Sec. 3 we then illustrate some feedbacks from our preliminary hands-on experience. Related work is briefly surveyed in Sec. 4 and conclusions are drawn in Sec. 5.

## 2 Approach

Our methodology aims at generating a set of SOAP messages sufficient to cover the whole interface provided by a WSDL file. Specifically, the tasks which must be carried out are:

---

<sup>1</sup> To be precise the test suites and test cases we derive only refer to input messages and data; the test oracle has still to be defined by the tester.

1. **WSDL Analysis:** A parser reads the WSDL specification and extracts information on operations, messages and data structures.
2. **SOAP Envelope Derivation:** Some tool is used to generate a skeleton of the SOAP message.
3. **Message Parts Definition:** For each data structure in the WSDL specification, different message instances are generated.
4. **Envelopes Composition:** The bogus data in the envelope skeletons are replaced with the actual derived instances.
5. **Messages Sending and Results Analysis:** The tester, or a batch script, sends the envelopes to the WS under test and collects the outputs for future inspection.

This methodology is eligible for combining different components to perform some of the above mentioned activities; in particular, we have selected soapUI and TAXI as two of them. The proposed architecture is sketched below.

The soapUI tool is responsible of the SOAP envelope skeleton derivation. TAXI is in charge of the actual message definition, and to do this it must extract the XML Schema data from the WSDL file (a modified version of the software or a preproduction script can be used for this purpose). The XML instances derived by TAXI and the envelope skeletons generated by soapUI can be assembled and sent to the WS. The results of the WS invocation are presented to the tester, or checked against provided expected output annotations (as done by soapUI). The whole process can be automated via a wrapping tool and the incorporation of suitable test strategies. We envisage that the generation of the SOAP messages can be carried out with various coverage criteria such as Operation Coverage, Message Coverage and so on, producing different degrees of detail.

## 2.1 soapUI

soapUI [5] is a tool developed by Eviware Software AB, available both in free and improved commercial versions. It assists programmers in developing SOAP-based web services. In particular, within the proposed methodology it allows the developer to generate stubs of SOAP calls for the operations declared in a WSDL file. Additionally, it is possible to use soapUI to send SOAP messages to the web service and display the outputs; this can be used for preliminary testing purposes.

Alternatively, for the purposes of this research it is possible to use any other tool capable of generating SOAP envelopes from WSDL files, such as Altova XMLSpy [1].

## 2.2 TAXI

TAXI (Testing by Automatically generated XML Instances) [3,4,9] is a tool able to generate XML instances compliant with a given XML Schema. It has been conceived so as to cover all interesting combinations of the schema by adopting a systematic black-box criterion. For this reason, TAXI applies the

well-known Category Partition (CP) technique [6] to the XML Schema. CP provides a stepwise intuitive approach to identify the relevant input parameters and environment conditions and combine their significant values into an effective test suite.

TAXI activity starts with the analysis of an input XML Schema. The implementation of CP requires the analysis of the XML Schema and the extraction of the useful information.

**choice** elements are processed by generating instances with every possible child.

Multiple *choice* elements produce a combinatorial number of instances. This ensures that the set of sub-schemas represents all possible structures derivable from *choice*.

**Element occurrences** are analyzed, and the constraints are determined, from the XML Schema definition. Boundary values for *minOccurs* and *maxOccurs* are defined.

**all** elements result in a random sequence of the *all* children elements for generating the instance. This new sequence is then used during the values assignment to each element.

Exploiting the information collected so far and the structure of the (sub)schema, TAXI derives a set of intermediate instances by combining the occurrence values assigned to each element.

The final instances are derived from the intermediate ones by assigning values to the various elements. Two approaches can be adopted: values can be picked from an associated database or generated randomly if no value is associated to an element in the database. Since the number of instances with different structures could be huge, in the current implementation TAXI only selects one value per element for each instance.

### 3 Preliminary Evaluation

To measure the feasibility and strength of the proposed approach, our methodology has been trialed for testing a WS which queries a publications database.

Using the WSDL available description we derived systematically a test suite along the steps presented in Sec. 2, and we compared it against a manually generated one, mimicking a human tester using the soapUI tool.

Both test suites consisted exclusively of XSD-compliant messages, and included both data actually taken from the publications database, and fictitious names or keywords. The two test suites have been used for testing the web services and the results have been collected. The first obtained feedbacks made clear that the manual test suite completely ignored certain classes of problem of the tested WS, while they evidenced a good performance of our approach in finding more problems.

In particular this experiment gave us the opportunity to detect bugs which had not popped out before. These errors were related to some parameters which were not passed to the search function.



The in-use version of the web service software had been thoroughly tested and is “bug-free enough” for ordinary use, while the version used for this experiment contains several improvements which still have to be integrated into the in-use application. Several manual tests had been previously run against the new features, but they were not sufficient to highlight the errors we found with a proof-of-concept of our methodology.

In conclusion, the experiments showed out that our systematic automated approach can provide a test suite which is more effective than the one which is created manually. In particular, having a test suite which covers such a wide range of variability in the structure and the values of the data would require a huge effort if done manually, even starting from a basis of automatically generated skeletons such as those provided by soapUI. Even though we have not yet performed a formal benchmark evaluation, the effort and time required appear drastically reduced using this methodology.

## 4 Related Work

There is today a list of good tools that can be used and have been adapted to test web services. Just to mention a few interesting ones, there are soapUI [5], PushToTest [8], SOATest [7]. Typically such tools are extremely effective in supporting the various testing activities and in increasing the productivity of testers. Nevertheless they mainly focus on management and execution of test cases and none of them tries to automatically provide test design and generation. Such a step is still mainly on the shoulder of testers and is strongly related to their ability. In particular none of the tools we analyzed take advantage of the availability of service models expressed in computer readable format suitable for automatic manipulation. In particular we refer here to the availability of XML-based description of service operation data models.

To the best of our knowledge, the only work which addresses issues similar to ours is [2], which also proposes XML-based test data generation and test operation generation. However, the work only outlines the possible perspectives of WSDL-based testing, but does not provide a tool, nor does it rely on standard test approaches. In our approach, instead, we intend to offer a “turn-key” tool which, by exploiting the existing TAXI tool, focuses on a systematic generation of test cases based on the Category Partition algorithm.

Finally, automatic generation of instances from XML Schemas its nowadays a feature of some, even commercial, tools [11,10]. Therefore our approach could be pursued even using other XML instance generation tools.

## 5 Conclusions and Future Work

Testing of Web Services is a challenging activity. Many characteristics (runtime discovery, multi-organization integration) of this new paradigm and its related technologies certainly contribute to make testing much more difficult. Nevertheless there are other characteristics that could be fruitfully exploited

for testing purposes. Among these, the representation of data in a computer readable format (typically XML-based) facilitates the automatic derivation of data instances to be used for testing invocations.

Starting from this consideration we presented a methodology to automatically derive test messages from WSDL descriptions. Such messages include data representing possible values that a real implementation of the service should be able to handle. We proposed that the generated data instances are encapsulated in correct SOAP envelopes that can be used to invoke a service implementation. Furthermore, by use of our tool TAXI, we proposed to exploit the characteristics of an XML Schema-based data description to automatically apply well known testing methods such as Category Partition and boundary value selection. This would result in the derivation of a test suite of messages that are representative of the space of possible messages.

The described methodology is still undergoing development and validation. Main tasks for the future include: To perform focussed and extensive evaluations in order to identify fault categories that are easily discovered and better define the usage scope; to extend the embedded TAXI functionalities so as to also generate non-compliant test cases that can support robustness testing of the invoked service; to complete the approach implementation and make it available as a free tool to the community for download and experimentation.

## References

1. Altova. XML Spy, [http://www.altova.com/products/xmlspy/xml\\_editor.html](http://www.altova.com/products/xmlspy/xml_editor.html)
2. Bai, X., Dong, W., Tsai, W.-T., Chen, Y.: WSDL-based automatic test case generation for web services testing. In: Proc. of IEEE Int. Work. SOSE, Washington, DC, USA, pp. 215–220. IEEE Computer Society, Los Alamitos (2005)
3. Bertolino, A., Gao, J., Marchetti, E., Polini, A.: Systematic generation of XML instances to test complex software applications. In: Guelfi, N., Buchs, D. (eds.) RISE 2006. LNCS, vol. 4401, pp. 114–129. Springer, Heidelberg (2007)
4. Bertolino, A., Gao, J., Marchetti, E., Polini, A.: Automatic test data generation for XML Schema based partition testing. In: Proc. Int. Work. on Automation of Software Test (ICSE 2007 companion), Minneapolis, Minnesota, USA (May 2007)
5. Eviware. soapUI; the Web Services Testing tool (accessed May 30, 2008), <http://www.soapui.org/>
6. Ostrand, T.J., Balcer, M.J.: The category-partition method for specifying and generating functional tests. Commun. ACM 31(6), 676–686 (1988)
7. Parasoft. SOATest (accessed June 3, 2008), <http://www.parasoft.com/jsp/products/home.jsp?product=SOAP>
8. PushToTest. PushToTest TestMaker (accessed June 3, 2008) <http://www.pushtotest.com/Docs/downloads/features.html>
9. TAXI. Testing by automatically generated XML instances (2007), [http://labse.isti.cnr.it/index.php?option=com\\_content&task=view&id=94&Itemid=49](http://labse.isti.cnr.it/index.php?option=com_content&task=view&id=94&Itemid=49)
10. Toxgene. Toxgene (2005), <http://www.cs.toronto.edu/tox/toxgene/>

# Automated Service Composition with Adaptive Planning\*

Sandrine Beauche<sup>1</sup> and Pascal Poizat<sup>1,2</sup>

<sup>1</sup> INRIA/ARLES project-team, France

{sandrine.beauche,pascal.poizat}@inria.fr

<sup>2</sup> IBISC FRE 3910 CNRS – Université d'Évry Val d'Essonne, France

**Abstract.** Service-Oriented Computing is a cornerstone for the realization of user needs through the automatic composition of services from service descriptions and user tasks, i.e., high-level descriptions of the user needs. Yet, automatic service composition processes commonly assume that service descriptions and user tasks share the same abstraction level, and that services have been pre-designed to integrate. To release these strong assumptions and to augment the possibilities of composition, we add adaptation features into the service composition process using semantic descriptions and adaptive extensions to graph planning.

**Keywords:** Services, Task-Oriented Computing, Composition, Software Adaptation, Planning, Workflow Languages, Tools.

## 1 Introduction

Task-Oriented Computing (TOC) envisions a user-friendly world where *user tasks* would be achieved by the automatic assembly of resources available in the environment. Service-Oriented Computing (SOC) is a cornerstone towards the realization of this vision, through the abstraction of heterogeneous resources as services. Yet, services being elements of composition developed by different third-parties, their reuse and assembly naturally raises composition mismatch issues [1]. Moreover, the TOC vision yields a higher description level for the composition requirements, *i.e.*, the user task(s), as the user only has an abstract vision of her/his needs which are usually not described at the service level.

To illustrate these issues, we use a running example [2], inspired by [3], which exposes a set of available services described with a conversation, a capability, inputs and outputs (Figs. 1 and 2). Conversations describe *how* to use services, while capabilities are semantic annotations that enable automatic reasoning for discovery and composition. In our work, conversations are described with a generic workflow language, YAWL, for which transformations from/to BPEL have been defined [4]. The Amazon service can be used to look for an eBook and provides a capability called BookSearch with a conversation (sequence) over

---

\* This work is supported by the project “PERvasive Service cOmposition” (PERSO) of the French National Agency for Research, ANR-07-JCJC-0155-01.

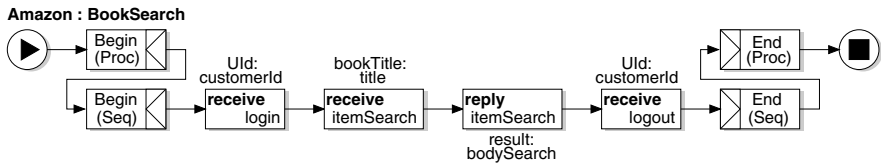


Fig. 1. Amazon service conversation (YAWL)

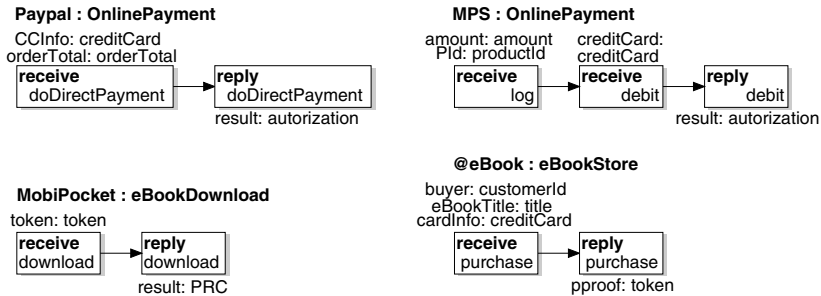


Fig. 2. Service conversations (communication part, YAWL)

three operations: `login` and `logout`, with a customer identifier (`customerId`) as input (in message part `UId`), and `itemSearch` with a book title (`title`) as input (in message part `bookTitle`) and a structured information on the search result (`bodySearch`) as output (in message part `result`). Additionally, `Paypal` and `MPS` can be used for payment, while `@eBook` can be used to search and pay at once. Finally, `MobiPocket` can be used to download an eBook in PRC format.

Still, the user knows neither the service capabilities, nor the data that should be exchanged between them to achieve service composition. The user only has a high-level view of her/his needs (user task): a capability, the inputs (s)he is ready to provide and the outputs (s)he expects. In the example, (s)he requires an `eBookRetrieve` capability, to provide `title`, `customerId`, and `creditCard` information, and finally get an eBook in PRC format. There is clearly a (vertical) mismatch between the user's needs and the service descriptions.

Additionally, the services have been developed by different third-parties. One may expect to compose them, while from the input/output perspective they could not be chained as-is. For example, `Amazon` should be composed with `Paypal` or `MPS` but part of the input data they require (respectively `orderTotal` and `amount+productId`) does not correspond to what one gets from a call to `Amazon` (`bodySearch`). This illustrates a (horizontal) mismatch.

These two dimensions of interoperability, namely *horizontal* (communication protocol and data flow between services) and *vertical matching* (correspondence between an abstract user task and concrete service capabilities) should be supported in the composition process.

The rest of this paper is organized as follow. Section 2 motivates the use of planning and adaptation, and discusses related work. Then, in Section 3, we present the principles of our approach for which more details can be found in [2], and we end with conclusions.

## 2 Discussion and Related Work

On the one hand, *planning*, is increasingly applied in SOC due to its support for automatic service composition from underspecified requirements [5]. Chaining-based planning composes services from provided and expected data, while hierarchical planning supports the decomposition of abstract requirements into concrete sets of tasks. Still, planning is not able to solve horizontal mismatch. On the other hand, *software adaptation* [1], is used to augment the possibility for component reusability and assembly, thanks to the automatic generation of software pieces, called adaptors, solving mismatch out in a non intrusive way. In this article we propose to combine planning and adaptation techniques.

Automatic composition is an important issue in SOC and numerous works have addressed this over the last years [6–13]. Various criteria could be used to differentiate these works, yet, due to our motivations, we will focus on issues related to user task requirements, vertical, and horizontal adaptation.

While both data input/output and capability requirements should be supported to ensure composition is correct wrt. the user needs, only [12, 13] do, while [7–11] support data only and [6] supports capabilities only. As far as adaptation is concerned, [9–12] support a form of horizontal (data) adaptation, using semantics associated to data; and [7] a form of vertical (capability abstraction) adaptation, due to its hierarchical planning inheritance. In our proposal, we combine the two techniques to achieve both adaptation kinds.

Few works explicitly add adaptation features to SOC [4, 14]. They adopt a different and complementary view wrt. ours since their objective is not to integrate adaptation within the service composition process in order to increase the composition possibilities, but rather to tackle protocol adaptation between clients and services, *e.g.*, to react to service replacement. Indeed, the most advanced software adaptation works [15, 16, 1] solve protocol mismatch between a fixed set of components, but tackle neither the discovery of the required components nor the composition towards user needs.

More information on planning and related work can be found in [2].

## 3 Adaptive Planning Composition

The basis of our work is the extension of the GraphHTN hierarchical planning technique [17] with horizontal adaptation features, and its application for service composition. Comprehensive information about the extension is given in [2].

We rely on two structures to support adaptation. Horizontal adaptation is supported by relations in an ontology of data types, in a structure we call *Data Semantic Structure* (DSS). It associates a set of concepts with a composition

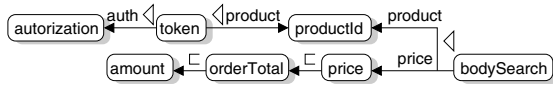


Fig. 3. DSS example

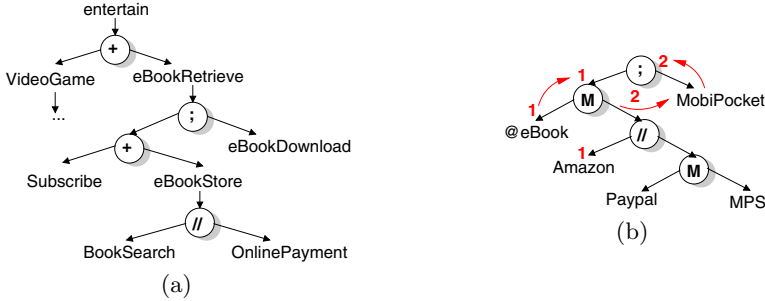


Fig. 4. CSS and l-CSS example (marking is used in graphplan building wrt. Fig. 5)

relation ( $\triangleleft$ ) – supporting (de)composition of data – and a simulation relation ( $\sqsubset$ ) – supporting data replacement. Using the DSS of our example (Fig. 3), we see that a `token` could be decomposed into an `authorization` and a `productId` (or the other way round) and that a `price` could replace an `orderTotal` as input for a service. Vertical adaptation is supported by a hierarchical (tree) structure describing relations between capabilities, that we call *Capability Semantic Structure* (CSS). It expresses (i) decomposition relations between abstract capabilities and more concrete ones, and (ii) ordering constraints between capabilities. The CSS nodes are either capabilities or control structures: sequence (`;`), choice (`+`) and parallel (`//`). In our example (Fig 4(a)), `eBookStore` is a capability which can be performed directly by a service, or that can be decomposed as the parallel execution of `BookSearch` and `OnlinePayment` capabilities.

Given a user task, a set of services, and both a DSS and a CSS, we proceed as follows. The CSS is first used to select, on the basis of their capabilities, services that could be used in the composition. Accordingly, the CSS is labelled with these services (l-CSS). A graph planning structure, named graphplan, is then computed. It chains services capabilities based on input/output dependencies and l-CSS constraints. Finally the graphplan is analyzed to retrieve all service compositions corresponding to the user task (which can be none).

**Service Discovery and l-CSS Computation.** The CSS is first restricted to the subtree with the user task capability as root. An abstract capability node is replaced by a *method node* (M) which denotes a choice: it can be either instantiated directly by some service *or* its definition (*i.e.*, its subtree) can. In Figure 4(a), `eBookStore` may be either obtained by calling `@eBook` or by composing in parallel `Amazon` (capability `BookSearch`) and `Paypal` or `MPS` (capability `OnlinePayment`). A capability node is replaced by the service that supports it (or

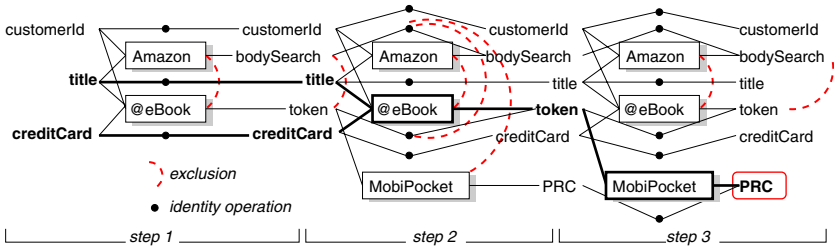


Fig. 5. Adaptive graphplan building (no data adaptation)

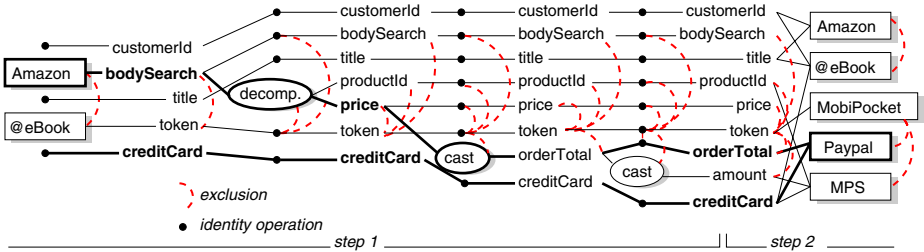
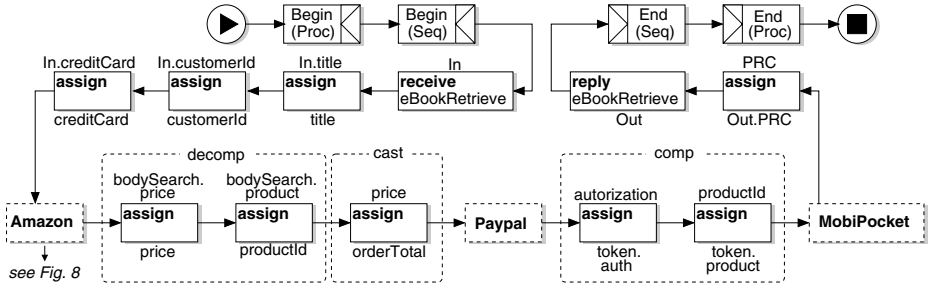


Fig. 6. Adaptive graphplan building (with data adaptation, principle)

siblings under a M node if several services apply, as for OnlinePayment). Finally, branches without service instances are discarded and control nodes with only one child are simplified. The l-CSS for our example is presented in Figure 4(b).

**Graphplan Building with Vertical Adaptation.** The graphplan is a structure with alternating fact (data) and action (service calls) layers. Dependencies between data and services are represented with arcs. The initial data layer corresponds to the user-provided inputs. The graphplan is then built (Fig. 5) chaining services (i) if their input data is available and (ii) following the orderings imposed by the l-CSS. Once a service is selected, it is tagged in the l-CSS (Fig. 4(b)) and its outputs are added to the next data layer. Identity operations are used to keep data from one data layer to the next one. As an example, the chaining of @eBook at step 1 enables the chaining of MobiPocket at step 2 (see Figs. 4(b) and 5). This would yield a correct composition, still, that should not contain Amazon that has been chained at step 1. To deal with such cases, exclusion relations are used to prevent services with exclusive capabilities in the l-CSS to appear in the same solution. Exclusions are propagated all along the graphplan. Since our objective is to generate all possible compositions, we stop the building when the maximum solution length, calculated with the l-CSS (here, 3), is reached.

**Adding Horizontal Adaptation to the Picture.** Let us now suppose we are after step 1 of Figure 5 and continue in Figure 6. According to the l-CSS



**Fig. 7.** A composition for user task (eBookRetrieve, {title, customerId, creditCard}, {PRC}) (YAWL)

(Fig. 4(b)), Paypal should be applicable. Yet, it is not, as it requires the unavailable orderTotal data. However, looking at the DSS (Fig. 3), we see that this can be obtained from price which in turn can be obtained using decomposition of bodySearch, which is available. The idea for horizontal adaptation is to add such data transformations in the graphplan building process. Supported transformations are the DSS ones:  $\text{decomp}(d, D)$  if  $D = \{d_i \mid d \triangleleft d_i\}$  (decomposition),  $\text{comp}(D, d)$  if  $D = \{d_i \mid d \triangleleft d_i\}$  (composition), and  $\text{cast}(d_1, d_2)$  if  $d_1 \sqsubset d_2$  (cast). Interestingly, one can have a task vision of these, *e.g.*, task cast above has precondition  $d_1$  and postcondition  $d_2$ . Data adaptation planning steps are performed at the end of the basic planning steps and are directed toward the set of data missing for applicable services (here, {orderTotal} for Paypal and {amount, productId} for MPS).

**Plan Extraction and Orchestration Generation.** Plan extraction is achieved backtracking the graphplan from the user task output data. The l-CSS is used for filtering at extraction time and to ensure that extracted plans respect the CSS constraints. Three plans are generated for our example:

- @eBook;MobiPocket (in bold in Fig. 5),
- Amazon;decomp(bodySearch, {productId, price});cast(price, orderTotal);Paypal; comp({authorization, productId}, token);MobiPocket (in bold in Fig. 6), and
- Amazon;decomp(bodySearch, {productId, price});cast(price, orderTotal); cast(orderTotal, amount);MPS;comp({authorization, productId}, token);MobiPocket.

Plans are then transformed into YAWL orchestrators, as demonstrated for the second plan in Figure 7. Orchestrators have a single operation, named according to the user task capability. Variables are used for semantic data types (*e.g.*, title) and for messages (*e.g.*, AmazonloginIn, or AloginIn in Fig. 8). Plan control structures are expressed using sequence and flow. Conversations of selected services are integrated reversing them, a receive/reply couple being replaced by an invoke. Finally, assignments are used to encode cast, comp, and decomp tasks.



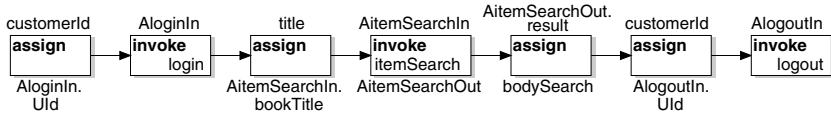


Fig. 8. Amazon conversation integration (YAWL)

## 4 Conclusion

In this paper we have proposed a technique that integrates adaptation features in the service composition process. We support both horizontal and vertical adaptation, which has been achieved combining semantic descriptions and hierarchical planning. We are also able to generate different composition solutions to the user task requirements, while ensuring they are correct from both data and semantics points of view. Our technique is fully automated thanks to *GraphAdaptor*, a prototype tool which takes as input a set of description files for user task, service and semantic structures, and outputs a YAWL file for each possible service composition. The main perspective of this work is the extension of our service model with conversations over capabilities and security features.

## References

1. Canal, C., Poizat, P., Salaün, G.: Model-based Adaptation of Behavioural Mismatching Components. *IEEE Transactions on Software Engineering* 34(4), 546–563 (2008)
2. Beauche, S., Poizat, P.: Automated Service Composition with Adaptive Planning (long version). In: Poizat, P. Web page
3. Marconi, A., Pistore, M., Poccianti, P., Traverso, P.: Automated Web Service Composition at Work: the Amazon/MPS Case Study. In: Proc. of ICWS 2007 (2007)
4. Brogi, A., Popescu, R.: Automated Generation of BPEL Adapters. In: Dan, A., Lamersdorf, W. (eds.) *ICSOC 2006*. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
5. Peer, J.: Web Service Composition as AI Planning – a Survey. Technical report, University of St.Gallen (March 2005)
6. Berardi, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In: Proc. of *ICSOC 2004*, pp. 105–114. ACM, New York (2004)
7. Klush, M., Gerber, A., Schmidt, M.: Semantic Web Service Composition Planning with OWLS-Xplan. In: Proc. of the *AAAI Fall Symposium on Agents and the Semantic Web* (2005)
8. Brogi, A., Popescu, R.: Towards Semi-automated Workflow-based Aggregation of Web Services. In: Benatallah, B., Casati, F., Traverso, P. (eds.) *ICSOC 2005*. LNCS, vol. 3826, pp. 214–227. Springer, Heidelberg (2005)
9. Constantinescu, I., Binder, W., Faltings, B.: Service Composition with Directories. In: Löwe, W., Südholt, M. (eds.) *SC 2006*. LNCS, vol. 4089, pp. 163–177. Springer, Heidelberg (2006)

10. Liu, Z., Ranganathan, A., Riabov, A.: Modeling Web Services using Semantic Graph Transformation to Aid Automatic Composition. In: Proc. of ICWS 2007 (2007)
11. Benigni, F., Brogi, A., Corfini, S.: Discovering Service Compositions that Feature a Desired Behaviour. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSSOC 2007. LNCS, vol. 4749, pp. 56–68. Springer, Heidelberg (2007)
12. Ben Mokhtar, S., Georgantas, N., Issarny, V.: COCOA: CONversation-based Service Composition in Pervasive Computing Environments with QoS Support. *Journal of Systems and Software* 80(12), 1941–1955 (2007)
13. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated Synthesis of Composite BPEL4WS Web Services. In: Proc. of ICWS 2006 (2006)
14. Motahari-Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-Automated Adaptation of Service Interactions. In: Proc. of WWW 2007, pp. 993–1002. ACM, New York (2007)
15. Inverardi, P., Tivoli, M.: Deadlock Free Software Architectures for COM/DCOM Applications. *Journal of Systems and Software* 65(3), 173–183 (2003)
16. Bracciali, A., Brogi, A., Canal, C.: A Formal Approach to Component Adaptation. *Journal of Systems and Software* 74(1), 45–54 (2005)
17. Lotem, A., Nau, D.S., Hendler, J.A.: Using Planning Graphs for Solving HTN Planning Problems. In: Proc. of AAAI/IAAI 1999 (1999)

# A Planning-Based Approach for the Automated Configuration of the Enterprise Service Bus

Zhen Liu, Anand Ranganathan, and Anton Riabov

IBM T.J. Watson Research Center  
{zhenl, arangana, riabov}@us.ibm.com

**Abstract.** The Enterprise Service Bus facilitates communication between service requesters and service providers. It supports the deployment of “message flows” from a service requester to one or more service providers. These message flows incorporate different functions such as routing, transformation, mediation, security and logging. In this paper, we propose an AI Planning-based approach for the automated construction of message flows between requesters and providers based on high-level goals specified by the enterprise architect or administrator. This automated construction of flows can be used either in the design phase where a developer or architect is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changed requirements. The planning model is based on tags, where goals, components, and links in the message flow are described using sets of tags. We describe the planning model and a case study that demonstrates the power of our approach in constructing flows in response to high-level requirements.

## 1 Introduction

The Enterprise Service Bus (or ESB) is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world. It facilitates mediated interactions between service endpoints. The Enterprise Service Bus supports event-based interactions as well as message exchange for service request handling.

One of the key challenges in the ESB lies in the construction of valid “message flows” between the service requesters and the service providers that perform the required set of mediation operations. Examples of such mediation operations include the routing of the messages to different end-points (e.g. for load balancing), transforming the messages to overcome differences in data schemas or semantics, different kinds of mediations such as splitting or combining messages, verifying security credentials, logging the messages, looking up the service reference using a registry, etc. Typically, a developer or an architect has to design each flow manually taking into account the specific requirements for that flow. This can be quite tedious and difficult when there are multiple options for each operation (such as different ways of transforming or logging the messages). The manual

approach to building the flows is also more prone to errors. Another source of difficulty is that the message flows may have to be manually reconstructed when there is a change in the data schema at any of the end-points, or when the requirements in terms of mediation operations change.

In this paper, we propose a methodology for the automated construction of message flows between the requesters and providers. This involves having a reusable set of components and sub-flows that perform different kinds of mediation operations. These components and sub-flows can be combined together and parameterized in different ways depending on high-level mediation requirements for the final flow. The automated composition allows the user to specify the set of high-level operations and have a message flow be generated automatically.

The key technology behind the automated composition of the message flows is an efficient AI Planning-based approach that constructs the flows given high-level goals specified by an enterprise architect or administrator. Our AI Planner [1] uses a tag-based model of the links in the message flow, the different components and of the goals. In this tag-based model, the inputs and outputs of different components are associated with semantic metadata in the form of a set of tags (or keywords) that are drawn from a taxonomy of tags. This taxonomy can, in fact, be a folksonomy that is built through the collaborative efforts of different developers and architects. Similarly, the high-level goals for the composition can also be described as a set of tags. An example high-level goal may consist of the following tags: `RequestLogging`, `ServiceProxy`, `ServiceX`, `RegistryY`, `AccessControl`. These tags together represent a requirement for a flow from a requester to a `ServiceX` that goes through a proxy which looks up the reference (or address) of the service in a `RegistryY`. The flow should also log all request messages and verify that that the requester has access to the service. Our planner can take this goal, along with tag-based descriptions of different components, to compose a flow that performs these different mediation operations. In some cases, it may come up with multiple flows, and then it uses a provided metric to rank the different plans (such as the resource utilization of the plan).

The automated construction of flows can be used either in the design phase where a developer is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changing requirements. In this paper, we describe a case study with a set of components that can be composed into different flow. We also provide performance results for our planner in this domain.

## 2 Message Flows in the Enterprise Service Bus

In our work, we model message flows between service providers and requesters as directed acyclic graphs (DAGs), where the vertices represent different components and the edges represent dataflow links. Each component performs some kind of mediation operation. Request messages flow from the requesters to the providers, and response messages flow in the reverse direction. Note that the

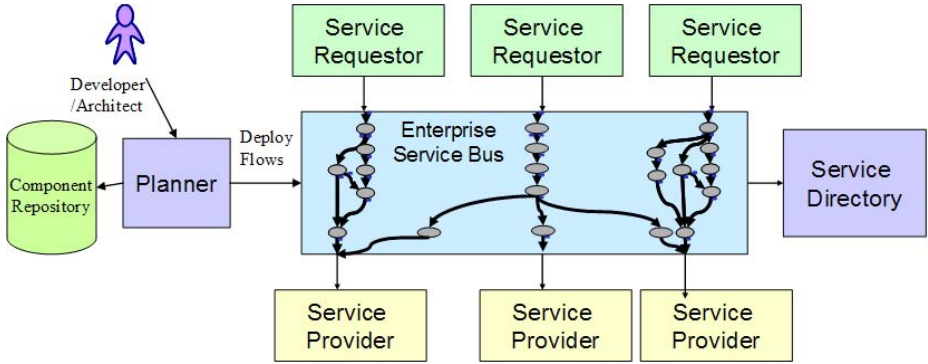


Fig. 1. Architecture for Automatic Composition

request and the response message flows can have very different structures, with different components performing different mediation operations.

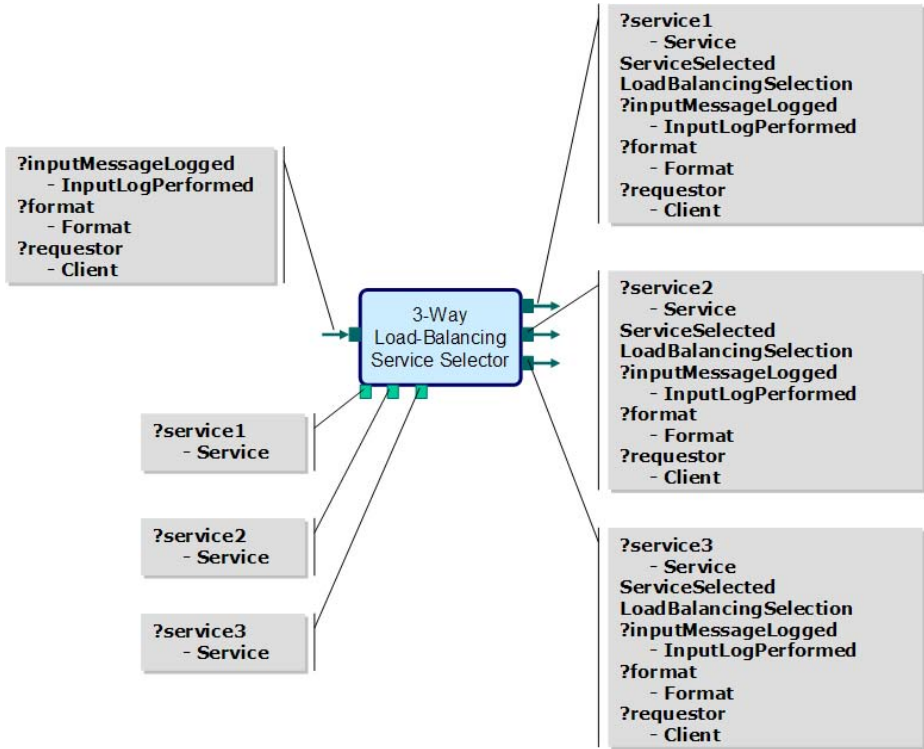
Fomally, a flow is a graph  $G(V, E)$  where  $G$  is a DAG (Directed Acyclic Graph). Each vertex  $v_i \in V$  is a component. Each edge  $(u, v)$  represents a logical flow of messages from  $u$  to  $v$ . If there is an edge  $(u, v)$ , then it means that an output message produced by  $u$  is sent as an input message to  $v$ . For now, we assume that a vertex has only one incoming edge to it.

Figure 1 shows the architectural elements supporting automatic composition. The planner automatically composes the message flows given high-level goals in the form of sets of tags. These goals may be provided by developers or architects when they are composing new flows between endpoints in the ESB. The planner may also be invoked during runtime, when the goals (or requirements) of existing flows change, and these existing flows need to be replaced by new flows that obey the new requirements.

The planner obtains tag-based descriptions of different components from a component repository. Note that this component repository consists only of mediation components that can be deployed in the ESB. This repository is different from a Service Directory that stores references to service end-points. The Service Directory may be used by components in the ESB to look up service providers that satisfy certain properties.

### 3 Tag-Based Model of Components and Goals

Let  $T = \{t_1, t_2, \dots, t_k\}$  be the set of tags in our system. A tag hierarchy,  $H$ , is a directed acyclic graph (DAG) where the vertices are the tags, and the edges represent “sub-tag” relationships. It is defined as  $H = (T, S)$ , where  $T$  is the set of tags and  $S \subseteq T \times T$  is the set of sub-tag relationships. If a tag  $t_1 \in T$  is a sub-tag of  $t_2 \in T$ , denoted  $t_1 \prec t_2$ , then all resources annotated by  $t_1$  can also be annotated by  $t_2$ . An example of a sub-tag relationship is `LoadBalancingSelection`  $\prec$  `ServiceSelection`. For convenience, we assume that  $\forall t \in T, t \prec t$ .



**Fig. 2.** Tag-Based description of a 3-way Load-balancing Service Selector component. The component has one input port where it receives messages. It forwards this message onto one of 3 output ports, each of which is connected to a possibly different service provider. The identities of these 3 service providers is determined by the values of the parameters (`?service1`, `?service2`, `?service3`). All variables in the input, parameters and outputs are associated with their type.

In our approach, tags are used to describe each data link in a message flow is associated with a set of tags. The tags describe the semantics of the messages that flow in the link, as well the actual syntax (using tags that correspond to names of types).

### 3.1 Component Model

Our model uses the tags in a folksonomy to associate format and semantic information with the input message requirements, the configuration parameters and the output messages of components. Our model also includes the use of variables to describe how the semantic properties of the data are propagated from the input and configuration parameters to the output message. This helps in capturing the notion of semantic propagation, i.e. the semantic description of the output of a component depends on the semantics of the input.

Figure 2 provides an example description of the `SelectService` component that selects one of 3 service providers based on load-balancing requirements. This component has one input port, 3 parameters and 3 output ports.

The folksonomy-based description of the `SelectService` includes a variable called `?inputMessageLogged`, which is defined to be of type `InputLogPerformed`. The variable `?inputMessageLogged` may be bound to any sub-tag of `InputLogPerformed` (such as `InputMessageLogged` and `InputMessageNotLogged`). This is an example of how our model captures semantic propagation; the output packet is annotated by the same input information about whether the input message was logged or not.

### 3.2 Composition

The tag-based model allows determining whether a data link, produced by some sub-flow, or a parameter value, can be given as input to another component. The syntax and semantics of a data link,  $a$ , can be described by a set of tags,  $d(a)$ . An input message constraint,  $I$  is defined as a set of tags and variables. We define that  $d(a)$  *matches* an input constraint,  $I$  (denoted by  $d(a) \sqsubseteq I_o$ ), iff

1. For each tag in  $I$ , there exists a sub-tag that appears in  $d(a)$ .
2. For each variable in  $I$ , there exists a tag in  $d(a)$  to which the variable can be bound. Note that variables can be bound to any sub-tag of their types.

After a match is found for each input to a component, the tag-description of the output message of the component is then formed by replacing all the variables in the output description by the tags to which they were bound in the input side. The use of variables allows us to describe how the semantic properties of the data are propagated from the input to the output packet.

### 3.3 Goals and Planning

A goal is also described as a set of tags. The goal is satisfied by a flow that produces a message flow with a data link that *matches* the goal tags.

In order to compose flows given an end-user goal as a set of tags and the descriptions of components, we use a planner based on the SPPL formalism. SPPL [1] is a variant of PDDL (Planning Domain Definition Language) and is specialized for describing stream-based planning tasks (a stream can be considered to be a special kind of a data link). At a high level, the planner works by checking if a set of links available in the current state can be used to construct an input to a component, and if so, it generates a new data link corresponding to the output. It performs this process recursively and keeps generating new links until it produces one that matches the goal pattern, or until no new unique links can be produced.

## 4 Case-Study

We developed a prototype implementation of our automatic composition approach running on IBM's Websphere Message Broker. For this purpose, we created tag-based descriptions of 60 different mediation components that were

deployed on the bus. These components could be composed into different kinds of message flows that followed different mediation patterns. Examples of the patterns included a service proxy pattern (where a proxy component performed a service endpoint lookup in a registry for request messages), a service selector pattern (where a service selector component routed request messages to different implementations of the same service interface) and a service normalizer pattern (where a service selector component routed request messages to different services with different interfaces and transformation components changed the format appropriately). Each of these basic patterns could be enhanced with additional functionalities such as logging of input messages, access control, logging of transformed messages, service lookups in different registries, etc. In our descriptions of the different components, we associated different tags with the different patterns, the different enhanced functionalities as well as different ways of configuring the basic patterns (such as using different service registries or different service selection criteria like load-balancing or content-based routing).

We have a tag-cloud based interface where tags corresponding to composable flows are displayed and selectable by the end-user as part of his goal. The planner may often come up with multiple flows for the same goal (especially if the goal contains few tags and is hence under-constrained). In this case, the lowest cost message flow is displayed to the end-user, although the user can view the alternative flows in the interface. In our setup, each component is associated with a cost, and the cost of a message flow is the sum of the costs of the constituent components. By default, all the components have the same cost; this results in the shortest message flows being shown to the user.

## 5 Related Work and Conclusion

The most closely related works are in the area of web service composition. Many different kinds of web service models have been proposed in prior work, and these models have been used for discovery and automatic composition. Some of these approaches use ontologies and associated standards such as OWL-S to describe components used in composition [2,3,4]. The key difference in our approach is that message flows in the ESB are generally data processing flows, where the the models of the components must be able to express message mediation and transformation operations, but need not model the state of the component itself. For describing the messages themselves, we use a much simpler tag-based model that reduces the knowledge engineering work required upfront for composition (compared to the more complex models suggested in prior work).

In conclusion, we have described propose an AI Planning-based approach for the automated management and configuration of the ESB. In this approach, message flows between requesters and providers are constructed automatically from high-level goals specified by an enterprise architect or administrator. This automated construction of flows can be used either in the design phase where an architect or developer is designing the message flows, or it can be used during runtime for the automated reconfiguration or adaptation of the flows in response to changed requirements.



## References

1. Riabov, A., Liu, Z.: Planning for stream processing systems. In: AAAI (2005)
2. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: WWW (2002)
3. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
4. Heflin, J., Munoz-Avila, H.: LCW-based agent planning for the semantic web. In: Ontologies and the Semantic Web, AAAI Workshop (2002)

# Verifying Interaction Protocol Compliance of Service Orchestrations<sup>\*</sup>

Andreas Schroeder and Philip Mayer

Ludwig-Maximilians-Universität München, Germany  
{schroeda,mayer}@pst.ifi.lmu.de

**Abstract.** An important aspect of service-oriented computing is the ability to invoke services without knowledge of the actual implementation. This requires at least a description of the service interface; better yet is a specification of the complete interaction protocol. This applies to atomic services as well as service compositions. In both cases, however, guaranteeing that a service complies with the promised interaction protocol is crucial for deadlock-free communication. In this paper, we present an analysis method and tool for verifying compliance of service orchestrations with service interaction protocols given as UML models. Our method is part of a larger suite of methods and tools for model driven development of service oriented architectures covering code generation for the Web service stack and other service platforms: MDD4SOA.

## 1 Introduction

A core aspect of Service-Oriented Computing (SOC) is enabling (semi-) automatic discovery and composition of services. To achieve this aim, services are accompanied with a description of their interface, consisting of a description of accepted and sent messages and its interaction protocol detailing which interactions with the service are legal at a certain point in a conversation. Interaction protocols greatly help in assembling services and enable more thorough mechanical checkings of whether services fit together. However, for this checking to be helpful, it is crucial that service orchestrations conform to their interaction protocols; otherwise, services may be proven incompatible although fitting, or even worse, services may be considered compatible when they are not. Therefore, tools for checking the conformance of service implementation (in our case, the orchestration) and service description (i.e., the interaction protocol) are urgently needed.

In this paper, we present an approach for checking the conformance of service orchestrations given as UML activity diagrams and interaction protocols given as UML protocol state machines. We use the UML as it constitutes the lingua franca of software engineers, and furthermore allows to follow a model driven approach: to create platform independent software models that can be transformed into platform specific realizations.

The remainder of this paper is structured as follows. First, we give a brief overview over our UML profile for services in Sec. 2. In Sec. 3, we detail the basic ideas of

---

<sup>\*</sup> This work has been partially supported by the EC project SENSORIA, IST-2005-016004.

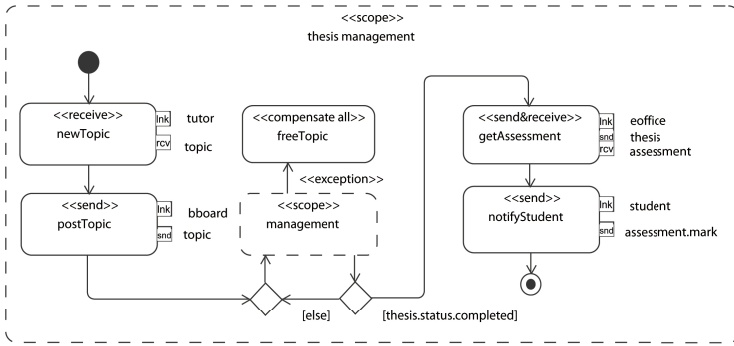


Fig. 1. Thesis Management: Service Orchestration

our refinement analysis, and describe an accompanying tool. Finally, we present related work in Sec. 4 and conclude in Sec. 5.

The work presented in this paper has been developed in the EU project SENSORIA, which aims at developing a novel comprehensive approach to the engineering of service-oriented software systems. More about SENSORIA may be found in [14].

## 2 Modelling Services with UML4SOA

Our UML profile UML4SOA [9] enables convenient modelling of service-oriented systems on a platform independent level. So far, UML4SOA included stereotypes for modelling (a) service orchestrations only, and was extended for the purpose of conformance checking to address (b) service architecture modelling with UML composite structure diagrams, and (c) service interface descriptions using class diagrams (for the static part) and protocol state machines (for the dynamic part).

**Service Orchestration.** UML4SOA includes a profile for modelling service orchestrations with UML activity diagrams. The profile employs native UML2 elements where possible, e.g. for modelling branches, loops, and parallel control flow, and extends the UML2 with service-specific concepts like service calls as well as event- and compensation handling. For details on the extension for service orchestration, see [9].

Fig. 1 shows the orchestration of the student thesis management example taken from [9]. It models the process of handling a student thesis from the initial topic provided by a tutor, through managing the ongoing thesis writing process, to the final assessment and notification.

**Service Composition Structure.** The service composition structure describes the participants in a service oriented architecture and how services may be used by participants. For this, we use UML2 composite structure diagrams. To specify the services of a SOA, UML4SOA follows the UML2 component-based design approach: <<service>>-stereotyped Ports represent services, and <<service provider>>-stereotyped Components represent service providers, that is to say, software entities that expose and offer services to other software entities.

**Service Interfaces.** While the service composition structure gives an overview over the SOA infrastructure, it does not describe the interfaces of the involved services. For this task, we use UML2 class diagrams to model service providers, their services, provided operations, and required callbacks. Going beyond static service descriptions, we use UML2 protocol state machines to describe behavioural aspects of service interfaces. In this way, the intended usage can be specified more precisely than by relying on operation names and message types as done e.g. in WSDL.

UML protocol state machines (PrSM) are used to describe valid interaction sequences of a port with finite state machines consisting of protocol states and protocol transitions between them. Transitions of PrSMs may contain a precondition, event, and postcondition. In UML4SOA, we use events to model incoming and outgoing messages. Pre- and postconditions are not considered yet, but may be in the future.

Fig. 2(a) shows the protocol offered to a thesis tutor by the thesis management service provider. It specifies that the provider first requires to `<<receive>>` a `newTopic` message, and only allowing status queries (performed by `<<receive>>` `getStatus` and `<<send>>/<<reply>>` `getStatus` from the service) from a tutor once the topic was picked up by a student, which is invisible to the tutor and hence `<<internal>>`. In addition to the correct protocol, we introduced a `<<receive>>` transition from and to `topicPosted` with the reception of a `postTopic` message, hence requiring that the topic for a thesis may be posted arbitrarily often; obviously, the orchestration does not satisfy this requirement, which is discovered by the conformance verification tool, as discussed in Sec. 3.

### 3 Analysing Service Orchestrations

Once a service orchestration and its service interface is modelled, the question arises whether the orchestration conforms to the defined protocol – which is part of the published service description. In more formal terms, the question is whether the orchestration constitutes a valid refinement of the protocol.

For this, we implemented an approach that (1) allows to select service providers and services for checking, (2) analyses the associated service orchestrations and interaction protocols for syntactic consistency, (3) allows to select failure configurations, (4) generates modal input/output labelled transition systems (modal I/O LTS, see [8] for the complete formal framework) from both service orchestrations and interaction protocols, (5) performs the conformance analysis and (6) outputs the results graphically.

**Selecting service providers and services.** The MDD4SOA conformance checker accepts UML XMI files and allows users to select, from the model, the services and service providers they would like to validate.

**Checking Syntactic Consistency.** Since not all UML activity diagrams may be translated into a modal I/O LTS, the MDD4SOA analysis tool checks the syntactic consistency of a given service orchestration before performing any refinement analysis, and informs the service engineer about syntactic inconsistencies found in the model. We cover general consistency constraints such as forbidding non-executable activities and MDD4SOA specific constraints restricting the branching and forking structure.

**Selecting Failure Configurations.** Before performing the conformance analysis, the tool allows the user to choose possible erroneous service interactions, i.e. service interactions that may fail due to unavailability of the communication partner or transmission errors. By this, the user is able to adjust the generated LTS and hence analyse the service orchestration conformance with varying fault assumptions. This feature emerged from practice, as conformance to reasonable service protocols is only achievable if some service interactions are assumed as being reliable.

**Generating Modal I/O LTS.** In order to reach an answer to the question of protocol conformance, a common semantic basis for service orchestrations and interaction protocols must be found. We chose as this common basis modal I/O LTS. Since the simple UML protocol state machines we use in our approach can be easily translated to modal I/O LTS, one half of the transformation is very straightforward.

Service orchestration models on the other hand use a subset of UML activity diagram constructs as well as the stereotypes introduced by UML4SOA. We defined a translation of service orchestration models to modal I/O LTS by following the widespread interpretation of activity diagrams as Petri nets in the sense that a state is a set of markers, and markers can be roughly interpreted as Petri net tokens (cf. Fig. 2(b) for an extract of the LTS generated for the thesis management orchestration).

The modal I/O LTS that is constructed from a service orchestration  $S$  roughly consists of states representing possible execution states of  $S$ , and transitions from one execution state to its successor for each service interaction or event that may occur. An execution state of  $S$  consists of the orchestration  $S$  with markers of different kinds (e.g. “program counter” markers, “active” markers on currently processed scopes, or “completed” markers attached to activity nodes).

**Performing Conformance Analysis.** Now that both service orchestration and interaction protocol are represented as modal I/O LTS, we can verify whether a refinement exists using observational modal refinement as defined in [8].

Observational modal refinement is computed in the MDD4SOA conformance analysis tool by – starting from the pair of initial orchestration and protocol states – enumerating all alternative successor pairs using and-or trees. If a pair of states is found during process which violates the observational modal refinement conditions in one step (by not providing a required action or performing an illegal action), this fact is memorized within the and-or tree, and propagated to its predecessor nodes in a final step. All propagated paths that lead (together with others or alone) to a falsification of the initial pair represent execution traces that must be reported to the user, as they constitute violation traces of the refinement between interaction protocol and service orchestration.

**Graphical Output.** Considering the thesis management example, Fig. 2(c) shows how an error trace is displayed: the service orchestration does not support the reception of `postTopic` after it was once received. The analysis result states that the service orchestration fails to support the reception of a `postTopic` message once it completed the first `postTopic` receive activity.

The MDD4SOA conformance analysis tool is an open source Eclipse plugin available at [mdd4soa.eu](http://mdd4soa.eu). The MDD4SOA tool suite also features code generation tools that allow the generation of e.g. BPEL/WSDL code from UML4SOA models.

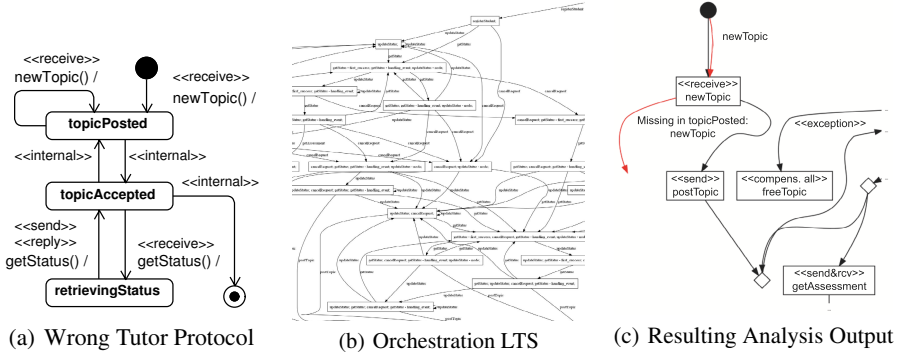


Fig. 2. Analysing with UML4SOA

### 4 Related Work

Several proposals exists for modelling service-oriented systems defining own UML profiles for SOA or proposing a proprietary approach. For a comparison of these approaches to our UML4SOA profile, see [9].

Several other approaches to activity diagram semantics exist, of which a comprehensive overview can be found in [12]. Other tools for refinement analysis based on bisimulation exist, e.g. Ticc [11] and Chic [4], where the latter is also applicable to Web services [3]. Both tools however define their own textual notations for modelling.

Other approaches to analyzing services often focus on BPEL [5,7,6], XLANG [15] or own notations (based on petri nets, e.g. [13], finite state machines, e.g. [2], or process-calculi, e.g. [11]), entailing platform lock-in or a steeper learning curve for software engineers compared to our UML-based approach, respectively. Furthermore, most approaches focus the analysis of single services (e.g. verifying internal consistency of single BPEL processes) or on composability of services (e.g. checking whether a composition of BPEL processes is deadlock-free or satisfies other properties). However, conformance of services to behavioural descriptions will constitute a vital part in service engineering and deployment [11,10]. This may be related to the fact that the Web service stack does not yet offer means for behavioural description of Web services.

A particular close approach to ours is [15], which generates behavioural descriptions of orchestrations instead of checking their conformance. This is an interesting approach that however does not support top-down service engineering (first creating the description, then the realization) as a conformance verification approach does.

### 5 Conclusion

In this paper, we have addressed the problem of checking compliance of service orchestrations with their interactions protocols. We employ modal observational refinement based on a modal I/O LTS semantics of service orchestrations and service protocols, which allows to display informative error traces to the service engineer. Our work is

built on an extension for UML2, the UML4SOA profile, which includes stereotypes for specifying both static and dynamic aspects of service compositions.

As conformance between service orchestration and interaction protocol is of crucial importance for (semi-) automatic service composition, we believe that the provided ability to mechanically verify and confirm that an orchestration follows its promised interaction protocol significantly eases achieving error-free communication between services. With the UML4SOA profile and the MDD4SOA tool suite, our approach offers straightforward modelling, checking, and generation of SOA artefacts.

## References

1. Adler, B.T., de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Raman, V., Roy, P.: Ticc: A Tool for Interface Compatibility and Composition. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 59–62. Springer, Heidelberg (2006)
2. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of E-Services that Export Their Behavior. In: Orłowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
3. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web service interfaces. In: 14th Int. Conf. on World Wide Web, pp. 148–159. ACM, New York (2005)
4. Beyer, D., Chatterjee, K., Henzinger, T.A., Mang, F.Y.C.: Chic: Checker for Interface Compatibility, [www.eecs.berkeley.edu/~arindam/chic](http://www.eecs.berkeley.edu/~arindam/chic)
5. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Compatibility Verification for Web Service Choreography. In: 3rd Int. Conf. on Web Services, pp. 738–741. IEEE, Los Alamitos (2004)
6. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: 3rd Int. Conf. on Web Services, pp. 621–630. IEEE, Los Alamitos (2004)
7. Kovács, M., Gönczy, L.: Simulation and Formal Analysis of Workflow Models. In: 5th Int. Workshop. on Graph Transformation and Visual Modeling Techniques. Electronic Notes in Theoretical Computer Science, pp. 215–224. Elsevier, Amsterdam (2006)
8. Larsen, K., Nyman, U., Wasowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
9. Mayer, P., Schroeder, A., Koch, N.: UML4SOA: Model-Driven Service Orchestration. In: 12th Int. Enterprise Computing Conf. IEEE, Los Alamitos (2008)
10. Meredith, L.G., Bjorg, S.: Contracts and Types. *Comm. ACM* 46(10), 41–47 (2003)
11. Salan, G., Bordeaux, L., Schaerf, M.: Describing and Reasoning on Web Services Using Process Algebra. In: 3rd Int. Conf. on Web Services, pp. 43–50. IEEE, Los Alamitos (2004)
12. Störrle, H.: Structured Nodes in UML 2.0 Activities. *Nord. J. of Comput.* 11(3), 279–302 (2004)
13. van der Aalst, W., Weske, M.: The P2P approach to interorganizational workflows. In: Ditttrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–155. Springer, Heidelberg (2001)
14. Wirsing, M., Clark, A., Gilmore, S., Hölzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-based development of service-oriented systems. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)
15. Wombacher, A., Mahleko, B.: Finding trading partners to establish ad-hoc business processes. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 339–355. Springer, Heidelberg (2002)

# Specify Once Test Everywhere: Analyzing Invariants to Augment Service Descriptions for Automated Test Generation

Amit Paradkar\* and Avik Sinha

IBM T J Watson Research Center, 19 Skyline Drive, Hawthorne, NY, USA 10532  
{paradkar, aviksinha}@us.ibm.com

**Abstract.** We present a technique which enables a novel *specify once, test everywhere* paradigm by exploiting invariants in a reference ontology. In our approach, each service operation is described in an IOPE paradigm: Input, Output, Precondition and Effect. Our approach augments the service description by creating additional service fault specifications to describe the exceptional behaviors which may arise as a result of invariant violations. We describe our invariant analysis technique and present experimental results which justifies the underlying intuition.

**Keywords:** Invariants analysis, Service Functional Testing, Automated Test Generation.

## 1 Introduction

Service oriented architectures (SOA), and Semantic Web Services are specified using standards such as OWL-S and WSDL [1] and consists of service descriptions in terms of Inputs, Outputs, Preconditions, and Effects (IOPEs). Complete specification of semantic service contracts entails specification of exceptional behavior as well. Furthermore, the same fault may be returned by more than one services. Oftentimes, these faults originate from an invariant on the system state. Identification of preconditions for fault messages which originate from system state invariants may be an onerous task for a service specifier. Furthermore, testing of such web services poses challenges because of the need to ensure that services which may violate such invariants have a correct behavior.

In the past, we reported a technique for Automated test generation for semantic web services described using IOPE [2]. However, this technique does not exploit the information provided in the invariants in the reference ontology. In this paper, we introduce a novel approach which exploits the invariants in a reference ontology to augment a service description with additional `fault` messages. In particular, the specific contributions of this paper are:

1. A novel technique to perform analysis of invariants on the reference ontology and the IOPE model of services that manipulate the instances of classes of the reference ontology to derive appropriate additional alternate faults for relevant services.

---

\* Contact author.



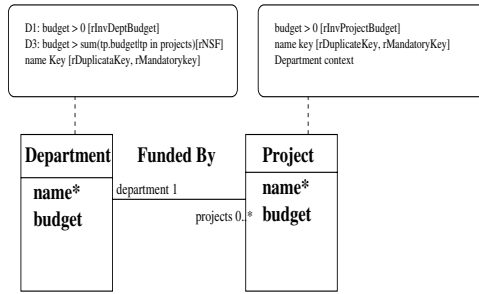


Fig. 1. Reference Ontology for DPSpec

- Results of an experiment using several service descriptions which demonstrate the benefits of using such approach during two phases of service development lifecycle: service description and service testing.

The rest of this paper is organized as follows: Section 2 describes our invariant analysis technique. Section 4 reviews the work related to ours and finally Section 5 summarizes the work and provides directions for future work.

## 2 Analysis of Invariants

Figure 1 shows a graphical representation of the reference ontology for a simple web service, called DPSpec, which manages Departments and Projects within a department.

It consists of 2 classes: `Department` and `Project` along with the properties as shown. Classes `Department`, and `Project` have object property `FundedBy` with cardinality constraints as shown. The reference model also has several invariants described using SWRL [3]. For example, invariant labeled D1 states that the `budget` attribute of the `Department` class has to be positive. Another interesting invariant D3 states that the `budget` of a `Department` should be sufficient to fund the cumulative budgets of its `Projects`.

Figure 2 describes some of the operations in DPSpec as specified by the modeler (before the augmented alternate flows derived during invariant analysis). For example, operation `Create Department` takes two inputs: `dName` and `dBudget` and returns as output: `rc`. The pre-condition for the operation is `TRUE` and its effect is of creating a `Department` instance and initializing its attributes to the supplied values. Each operation has a default successful behavior and potentially several faults each representing an exceptional behavior. The operation `Modify Department` has a normal flow guarded by the condition that ensures that a department object which satisfies the search criteria exists; along with appropriate state updates. An alternate flow is executed if no department object satisfying the search criteria is found and results in a `WSDL: fault`. Operation `Move Project` allows to move a project from an existing department to another one.

Information present in the invariants on a reference ontology may not be consistent with the information provided in the service operations model. Assuming that the

<pre> Operation <b>Create Department</b> <b>in</b> String dName <b>in</b> Double dBudget <b>FLOW</b> rOK <b>IF</b>   TRUE <b>effects</b> {d:=Create(Department)            d.name := pName            d.budget := pBudget} </pre>	<pre> Operation <b>Move Project</b> <b>in</b> String pD1Name <b>in</b> String pD2Name <b>in</b> String pPName <b>FLOW</b> rOK <b>IF</b>   ∃ d1, d2:Department,   p:Project•{d1.name =pD1Name}∧   {d2.name = pD2Name}∧   {p ∈ d1.projects ∧    p.name = pPName} <b>effects</b>   {CreateLink(Fundedby, d2,p)    DeleteLink(Fundedby, d1,p)} </pre>
<pre> Operation <b>Modify Department</b> <b>in</b> String pName <b>in</b> String pNewName <b>in</b> Double pBudget <b>FLOW</b> rOK <b>IF</b>   ∃ d:Department•d.name = pName <b>effects</b> {d.name := pNewName           d.budget := pBudget} <b>FLOW</b> rDeptNotFound <b>IF</b>   ∀ d:Department• d.name ≠ pName <b>fault</b> {message:=No such department} </pre>	<pre> Operation <b>Modify Project</b> ... </pre>

**Fig.2.** Operations for Web Service DPSpec

information in the invariants is correct, the inconsistencies could be resolved by either adding more flows (with new guard conditions) to the service operations model or by adding new effect statements to the operation model. In this paper, we focus on the first class of repair actions since these affect the test generation process. The second class of repair action is an indication of a genuine model error that needs to be fixed by the modeler.

## 2.1 Relating Operations and Invariants

To facilitate the necessary analysis, we need to establish the relationship among the invariants on reference ontology and the service operations since both these artifacts refer to the entities in the same reference ontology. Our invariant analysis technique accomplishes this task by first computing a *tripartite* graph, called *SOROINGraph*, in which the set of operations, the set of Reference Ontology entities (Classes and Properties), and the set of Invariants each form a partition. An operation node has an edge to a reference ontology entity if the operation manipulates (either *creates*, *updates* or *deletes*) the entity. Such an edge is labeled with 1) the nature of the manipulation (Create/Update/Delete), 2) the set of attributes of the concerned entity being initialized/modified, and 3) the *navigation path* - the chain of class accesses - used to reach the instance being manipulated. An invariant has an edge to a reference ontology entity if the invariant refers to it. Such an edge is labeled with the referred attributes and the navigation path used to reach each attribute.

Our analysis exploits the *SOROINGraph* to identify the set of potential operations affected by an invariant. This is done by traversing the edge from an invariant to the

reference ontology entities referred in it and then following the edges from the reference ontology entities to the operations that modify those entities. Thus, invariant D1 refers to class `Department` and attribute `budget` which in turn are modified by operations `Create Department` and `Modify Department`.

## 2.2 Deriving Flow Conditions

The next step is to create a predicate which reflects the conditions under which the invariant will be violated. We will illustrate the approach through several examples.

Our approach starts with the simplest class of invariants: those which do not involve navigation or aggregate functions. For example, the invariant D1 has the following form:  $\forall d1 : Department \bullet d1.budget \geq 0$ . Service operation `Create Department` has a effect statement  $d.budget = pBudget$ . Our approach exploits the `SORINGraph` to recognize that invariant D1 affects `Create Department`, and substitutes the variable `d` for the quantified variable `d1` in D1. Furthermore, the assignment of `pBudget` in the effect statement is accounted for by rewriting  $d.budget$  with `pBudget`. The predicate that leads to violation of the invariant D1 is given by negating the resulting predicate (thus converting the universal quantification into an existential one). The resulting predicate is:  $\exists d : Department \bullet pBudget \geq 0$ . The expression within the scope of the quantifier is independent of the quantified variable `d`, and further simplification results in expression  $pBudget \geq 0$ , which is used as a guard condition for an augmented service operation flow named `rInvDeptBudget` for `Create Department`.

The *key* invariants on class `C` are addressed by identifying the service operations which either create an instance of `C` (and hence initialize the key attribute) or modify an instance of `C` by modifying its key attribute. For each such operation, two alternate flows are identified: one which checks for the attribute value being duplicate (for uniqueness) and another which checks for the attribute value being null (for mandatory). Thus, for `Create Department`, two alternate flows with faults - `rKeyExists` and `rKeyNull` - are created with guard conditions:  $\exists d : Department \bullet d.name = pName$  and  $pName = NULL$  respectively.

The result of applying the rewrite rule for an invariant with aggregate operation `SUM` (invariant D3 in `DPSpec`), and which affects a service operation with `Create Link` effect (`Move Project`) is given below. The invariant D3 has the following form:  $\forall d : Department \bullet d.budget \geq \text{sum}(tp.budget \mid [tp : Project \in d.projects])$ . The `Create Link` effect of `Move Project` is given by `Create Link (Funded By, d2, p)`. We recognize that the association `Funded By` is accessed in the invariant by the navigation  $d.projects$ , thus we substitute all instances of the quantifier variable `d` in D3 with the instance variable `d2` in the effect. Furthermore, the effect of the `Create Link` with `Project p` is accounted for by adding the  $p.budget$  to the `SUM` expression. The resulting predicate is given by:  $\exists d2 . budget < \text{sum}(tp.budget \mid [tp : Project \in d2.projects]) + p.budget$ .

Unfortunately, considering all the predicates (and thus the alternate flows) obtained during this process may lead to infeasible alternate flows. For example, consider the predicate obtained as a result of applying the `SUM` invariant and `Remove Link` rule

to service operation `Move Project: d1.budget < sum (tp.budget | [tp : Project ∈ d1.projects ]) - p.budget`. Given that `d1.budget ≥ sum (tp.budget | [tp : Project ∈ d1.projects ])` holds (because the invariant was satisfied before the `Remove Link`), and that `p.budget > 0` (due to the invariant on the `budget` attribute of `Project` class), the predicate for the computed alternate flow above cannot be satisfied. We use constraint solvers to statically remove such infeasible alternate flows.

### 3 Experiments

In order to assess the effectiveness and efficiency gained in model description through the invariants analysis we ran an experiment. For the experiment we obtained web service descriptions for five applications with varying net number of operations. The IOPE description for each service, were manually derived. A reference ontology was populated based on the understanding of the systems. Invariants were modeled for each of the services following which “invariant analyses” were performed. The efficiency benefit comes from the reduction in the net size of the descriptions. Specification of the invariants eliminates specification of fault behavior in the individual service descriptions because such information can be automatically populated through invariant analysis. The reduced size of the description increases its maintainability and also brings in a reduction in cost when the service descriptions are manually administered. To measure this effect, physical and logical sizes of the descriptions were measured before and after the invariants analysis. Physical size of a description is measured by counting the total number of tokens in the description whereas its logical size is measured by counting the possible results of invoking the service. This count could be statically determined from a description as follows:

$$\text{logical size} = \sum_{\text{Operations}} \text{Number of WSDL faults in an Operation} + 1$$

The percent reduction in size of the description is used to measure the efficiency benefit(T).

$$T = \frac{S1 - S0}{S0} \times 100\%$$

$S0$  is the physical size or the logical size of the description and  $S1$  is the size of the enhanced model.

The effectiveness in test generation comes from the fact that the test generation processes are guided by coverage of the model. Thus if, for instance, one is using “boundary value testing” as his test generation process, the coverage criteria is determined by determining the boundaries of the variables of interest. In black box testing of web services, such variables are determined from their descriptions. Thus if the description does not explicitly refer to the variables of interest, they don’t become subject of “boundary value testing”. The effectiveness benefit(F) of invariant analysis is computed as follows:

$$F = \frac{f1 - f0}{f0} \times 100\%$$

**Table 1.** Experiment Measurements

Service	No. of Oprns	No. of Invrnts	$T_P$	$T_L$	$F_{BVT}$	$F_{ECT}$
ATM	3	1	83.33%	28.57%	33.33%	0.00%
Dept Project	4	1	64.71%	33.33%	37.50%	0.00%
Hotel Reservation	4	2	110.34%	50.00%	100.00%	100.00%
Purchase Order	7	2	108.11%	35.71%	28.57%	10.71%
Library	8	2	143.10%	110.00%	31.25%	31.25%

$f_0$  is number of faults targeted by a test suite when it is generated using the actual model and likewise,  $f_1$  is number of faults targeted when the test suite is generated using the enhanced model. In order to evaluate the effectiveness of the invariant analysis on test case generation, we generated test cases for the web service description using boundary value testing(BVT) [4] and equivalence class testing(ECT) [4] using the both the original and the enhanced model. In BVT, the boundary variables are identified by examining the guards on the flows of the operations. The boundary values for such variables were determined based on the types. In ECT, the process is similar, except that the boundary values are determined based on equivalence classes.

The results of our measurements are summarized in Table 1. In Table 1, the suffix  $BVT$  denotes the results for boundary value testing and the suffix  $ECT$  denotes the results for equivalence class testing. Also, the suffix  $P$  denotes the result for physical size and the suffix  $L$  denotes the result for logical size.

As is evident from the measurements the efficiency benefits are higher for services with higher number of operations. This follows the intuition that by concentrating the information one can reduce the effort in modeling of individual operations significantly. This strengthens our hypothesis  $H_T$ .

Effectiveness benefits, on the other hand, depend on the kind of domain invariants. If the domain invariant does not affect the variables in the guard of the operation but affect its effects, then the  $F$  is maximized. This is so because both  $BVT$  and  $ECT$  generate test cases based on free variables in the flow conditions and therefore they fail to consider the variables in effect of the operation. Following invariant analysis, the flow conditions are modified to include the variables in the effect that are under the influence of some invariants. Consequently, post-enhancement  $BVT$  and  $ECT$  target higher number of faults.

## 4 Related Work

Our work of invariant analysis is in the spirit of consistency analysis of specifications. Several works in the area of consistency analysis and fixing inconsistencies have been reported in the past. We briefly review the most relevant ones here. Nentwich *et al.* [5] introduced the concept of consistency analysis for XML documents based on rules defined in a XML based notation called *xlinkit*. The *xlinkit* rules allow users to specify consistency constraints among XML schema elements. *xlinkit* also has a consistency checker component which takes as input an XML document(s) which it checks for violation of the specified consistency constraints. Nentwich *et al.* [6] extended the

consistency analysis in *xlinkit* to incorporate *fixing* of the reported inconsistencies. Egyed [2] reported an approach for fixing inconsistencies across a set of artifacts of a UML design model. The artifacts being considered are class diagrams, statechart diagrams, and sequence diagrams.

## 5 Conclusions and Future Work

This paper presented an approach which enables a novel *specify once, test everywhere* paradigm by exploiting invariants in a reference ontology. In this approach, each service operation is described in an IOPE paradigm: Input, Output, Precondition and Effect. Our technique augments the service description by creating additional service fault. The augmented service operation model is then used during subsequent test generation. We described the techniques used in our invariants analysis approach and presented experimental results which justifies the underlying intuition.

There are several future directions we would like to pursue. We would like to conduct empirical studies both in industrial and academic settings to evaluate the our approach along the dimensions of useability and effectiveness. Since our approach is a form of consistency analysis for service models, we would like to extend it to the other class of invariants which need modifications to the *effect* part of the service model.

## References

1. T.O.S. Coalition. Owl-s: Semantic markup for web services (2003)
2. Egyed, A.: Fixing inconsistencies in uml design models. In: ICSE 2007: Proceedings of the 29th international conference on Software Engineering, pp. 292–301 (2007)
3. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Groszof, B., Dean, M.: Swrl: A semantic web rule language combining owl and ruleml (2004)
4. Jorgensen, P.: Software Testing: A Craftman’s Approach. CRC Press, Inc., Boca Raton (2001)
5. Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: Xlinkit: a consistency checking and smart link generation service. ACM Trans. Interet Technol. 2(2), 151–185 (2002)
6. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency management with repair actions. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering, pp. 455–464 (2003)
7. Paradkar, A., Sinha, A., Williams, C., Johnson, R., Outterson, S., Shriver, C., Liang, C.: Automated functional conformance test generation for semantic web services. In: ICWS 2007. IEEE International Conference on Web Services, pp. 110–117 (2007)

# A Model-Driven Approach to Dynamic and Adaptive Service Brokering Using Modes

Howard Foster<sup>1</sup>, Arun Mukhija<sup>2</sup>,  
David S. Rosenblum<sup>2</sup>, and Sebastian Uchitel<sup>1</sup>

<sup>1</sup> London Software Systems, Dept. of Computing, Imperial College London,  
180 Queen's Gate, London SW7 2BZ, UK  
{hf1,su2}@doc.ic.ac.uk

<sup>2</sup> London Software Systems, Dept. of Computer Science, University College London,  
Gower Street, London WC1E 6BT, UK  
{a.mukhija,d.rosenblum}@cs.ucl.ac.uk

**Abstract.** Industry and academia are exploring ways to exploit the services paradigm to assist in the challenges of software self-management. In this paper we present a novel approach which aims to bring these two fields closer by specifying the requirements and capabilities within a UML2 model architecture style and illustrating how these model elements are used to generate specifications for dynamic runtime service brokering given different modes of a software system. The approach is implemented in a tool suite integrated into the Eclipse IDE with a prototype runtime service broker engine.

## 1 Introduction

Software architectures have typically been specified for a static configuration, where static means that the initial configuration defines a single relationship model between various components in the architecture. With the use of a service-oriented architecture (SOA) style for loosely-coupled reusable software components (typically by technology independence) there is an interest to dynamically configure relationships between these components such that the architecture configuration changes as the system requirements or environment changes (re-configuration). We introduced the notion of service modes in service-oriented computing in [2] which outlined an approach to defining, abstracting and generating deployment artifacts from component models specified using the mode style. Additionally, dynamic reconfiguration of service architectures also requires that a series of services representing the same functional (or non-functional) requirements may be dynamically composed in to the network of services. As a part of the Dino project [9], we are working on addressing a number of challenges faced by the dynamic and adaptive composition of services in open dynamic environments. Our collective aim in this project is to provide technologies, tools and runtime systems for comprehensively supporting all stages of service engineering (i.e. requirements, discovery, selection, binding, delivery, monitoring and adaptation). In this paper we describe an integrated approach to dynamic service compositions and architecture reconfigurations.

## 2 Background and Related Work

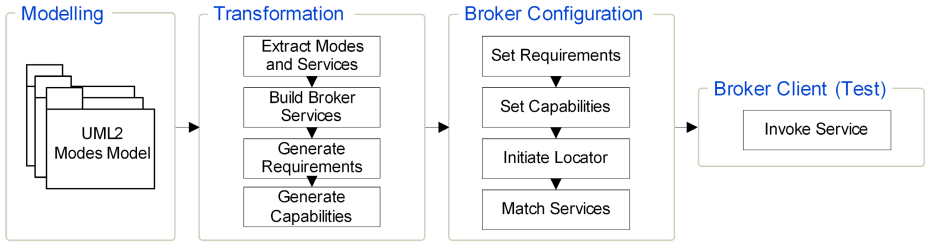
A mode, in the context of Service-Oriented Computing (SoC), abstracts a set of services that collaborate towards a common goal [5]. A mode can be used to identify which services are required in states of a system, and assist in specifying service composition requirements through component state changes. Modes can specifically be used towards addressing reconfiguration issues within a self-managed system. Self-management is typically described as a combination of self-assembly, self-healing and self-optimisation. Self-management of systems is not a new idea, with ideas from both the cybernetics and system theory worlds. However in SoC specifically, a dynamic service brokering solution needs to address issues like how to specify the Quality-of-Service (QoS) requirements and capability, and how to select the most appropriate service provider among several functionally-equivalent providers based on the QoS offered. An example service broker engine is called Dino [9]. Dino provides a runtime infrastructure consisting of a number of brokers. These brokers are responsible for, among other things, service discovery and selection on behalf of service requesters. Integrating service modelling, self-management concepts and dynamic service brokering aims at enhancing service engineering to cater for change, adaptive and extendible service solutions.

Related work is split between the modelling and brokering aspects of our work. For modelling requirements of services and SOA there has been several UML profiles proposed in [6,11,7,8]. These profiles generally provide a set of stereotypes that represent features of service artifacts, including a service specification (interface), gateway (ports) and orchestrated collaboration (behaviour specifications). What is generally missing from these existing profile approaches is the ability to identify the requirements and capabilities of services and then to elaborate on the dynamic changes anticipated for self-management. In terms of dynamic service brokering, most of the work on runtime infrastructure for service composition assumes that the global view of an abstract service composition is available centrally, such as the work by Yu et al. [10] and Zeng et al. [11]. In our approach, we do not make such assumptions, and instead allow decentralized composition of services, making use of extended modelling profiles to capture the dynamic change requirements anticipated for self-management.

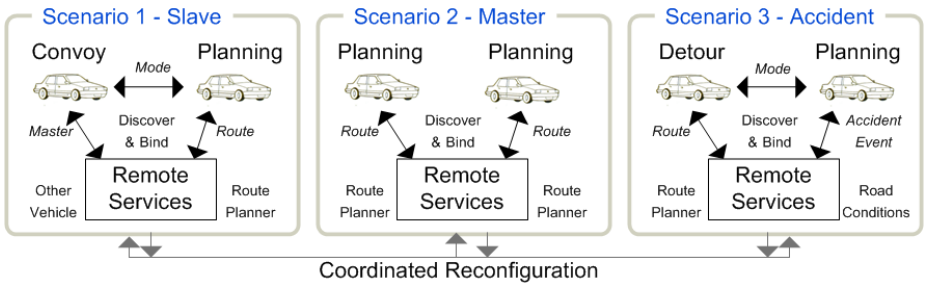
## 3 Overview of Approach and Case Study

Our approach is illustrated in Figure 1. Firstly, service engineers create a set of architecture models in UML2 using a Modes Profile to stereotype particular elements needed for service brokering requirements. These models are then used in a second step, to extract the mode related elements from the model and generate inputs to service brokers (in the form of requirements and capability documents). The inputs are passed to a service broker to prepare the required services for matching (both functionally and non-function aspects). Additionally, service provider capabilities can be registered with the broker to announce new





**Fig. 1.** Approach to Model-Driven Dynamic Service Brokering Requirements with Modes



**Fig. 2.** Mode Scenarios for Vehicle and Remote Service Collaboration

service offerings. Lastly, the broker can be invoked with service parameters to execute a given service call using a matched service linked with the broker.

To guide our work we used a case study based upon a set of requirements formed from those reported as part of a European Union project called SENSORIA [7], our role is to support the deployment and re-engineering aspects of this case study and in particular, to provide a self-management approach. In this case study are a number of scenarios relating to a Vehicle Services Platform and the interactions, events and constraints that are posed on this services architecture. One particular scenario focuses upon Driving Assistance (illustrated in Figure 2), and a navigation system which undertakes route planning and user-interface assistance to a vehicle driver. Modes are used to describe the changes between various states of the vehicle services system.

## 4 Capturing the Architecture Models and Modes

Using the modelling concepts of UML2 and Modes, we created a UML2 Modes Profile which assists Service Engineers in identifying mode collaborations, organised hierarchically using Mode Packages. Note that this profile extends and depends on the SENSORIA UML for SoC profile [7], for general service stereotypes (such as service, provider, requester etc). The stereotypes for the UML2 Modes Profile are defined in [3], however, here we provide a brief overview. In

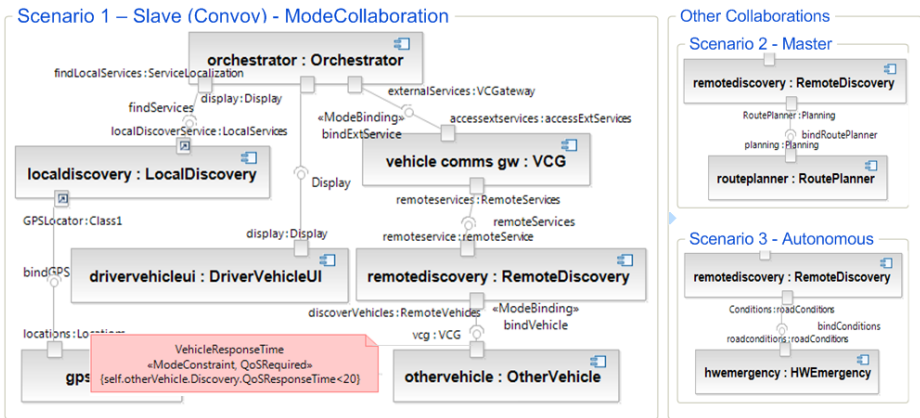
this paper we concentrate on architecture, configuration and dynamic service brokering requirements, and as such, examples of behaviour specification and transformation is left as future work.

*UML Models and ModePackage.* A UML2 model is a package representing a (hierarchical) set of elements that together describe the physical system being modeled. We define a ModePackage stereotype as an extension of this definition with a stereotype to designate that a package is being described for a system mode configuration, events and signals.

*ModeCollaboration and ModeBinding.* A ModeCollaboration extends a UML2 Collaboration containing a Composite Structure Diagram (CSD) to represent the collaborating service components relationships in this mode and one or more ModeInteraction or ModeActivity diagrams. Within each ModeCollaboration CSD, the UML2 element of Connector is stereotyped as a ModeBinding. A ModeBinding represents a required connection (or instantiation) in order to carry out the mode behaviour as described in a collaboration interaction set.

*ModeInteraction and ModeActivity.* A ModeInteraction contains a single interaction (sequence) diagram and optionally a single communication diagram. The sequence diagram is a message sequence chart describing the sequence of interactions between service components in the mode collaboration. A communication diagram provides an alternative view of the sequence logic for the mode interactions, in a sense a "bird's eye" view of the way the service components collaborate for a given configuration.

*ModeConstraint.* Constraining changes to a Modes-based architecture and service composition can be achieved in two ways. Firstly, in a ModeCollaboration specification, a ModeBinding can be constrained with a ModeConstraint, categorised by a further constraint stereotype. We extend a recommendation profile



**Fig. 3.** Composite Structure Diagram (ModeCollaboration) for Slave (Convoy) Component Configuration and alternatives for Planning and Detour Modes

of the Object Management Group (OMG) in [4]. Additionally, architectural constraints may be specified in the Object Constraint Language (OCL) or another constraint based language. The constraint language adopted becomes an implementation-dependent aspect of analysing or extracting from models in UML2. An example constraint for service ResponseTime, applied to a ModeCollaboration, is illustrated in Figure 3.

## 5 Requirements and Capabilities Specification for Dynamic Brokering

Dino provides a specification language for describing both functional and non-functional properties of service requirements and capabilities [9]. The specification language provided by Dino builds on the advances already made in the field of *semantic* specifications of services. Some of the popular efforts in this direction include, OWL-S, SA-WSDL, and WSMO. The main purpose of the Dino specification language is to provide the *mode* information for different service requirements and capabilities. For practical reasons, we have chosen to specify functional service descriptions using OWL-S in the current Dino broker implementation. This is because OWL-S is a mature standard for semantic service specifications, and a number of tools supporting this standard are available. For specifying the non-functional properties of services, none of the existing semantic service standards were found to be completely satisfactory. Therefore, we have developed our own specification language for describing non-functional properties of services, which relies on a standard QoS ontology. Examples of

```

<ReqDoc name="Driving-Assistant">
  <mode name="planning">
    <service name="GPS" functional="gps-req.owl" qos="gps-req.qos"/>
    <service name="Map" functional="map-req.owl" qos="map-req.qos"/>
  </mode>
  <mode name="convoy">
    <service name="RPS" functional="rps-req.owl" qos="rps-req.qos"/>
  </mode>
  <mode name="detour">
    <service name="HES" functional="hes-req.owl" qos="hes-req.qos"/>
  </mode>
</ReqDoc>

<CapDoc name="Driving-Assistant">
  <mode name="planning, convoy, detour">
    <service name="RPS" functional="rps-cap.owl" qos="rps-cap.qos"/>
  </mode>
</CapDoc>

```

**Fig. 4.** Requirements (ReqDoc) and Capabilities (CapDoc) of Driving Assistance Modes

non-functional properties include response time, availability, security etc. Details of the specification language for describing non-functional properties in Dino can be found in a previous paper [9]. Figure 4 shows an example requirements specification document for the Driving Assistance case study. As discussed earlier, Driving Assistance has three possible modes: planning, convoy and de-tour. Driving Assistance requires different services in all three different modes. However, the service provided by Driving Assistance in all three different modes remains the same, i.e. the route planning service, also shown in Figure 4.

A Dino broker can be hosted on a trusted third party node, or even on the same node as a service requester. Any number of Dino brokers can be deployed in an environment, as required for scalability. The brokers perform three main scenarios. Firstly, the service requester can invoke the Dino broker at runtime and forward its requirements specification document to the broker. Secondly, the Dino broker can discover candidate services and perform matchmaking based upon the requirements and thirdly, a service can be delivered either directly from the service provider, or through the Dino broker.

### 5.1 Extracting Service Requirements and Capabilities

To refine the specification for extracting service requirements and capabilities, we have three main element lists: *a set of mode packages*, *a set of mode collaborations* and *a set of mode components* (services). As described earlier in this section, the Dino broker considers two types of service requirements; 1) required services for a given mode and 2) provided services to announce capabilities that can be matched for a mode of operation. Identifying the type of a service in the mode model relies on alternative properties. The type, requester or provider, can be determined either by direct or profiled stereotype on a given component. The usage of a particular service is analysed by considering the ports and connectors of the component. Identifying the type and usage allows us to generate functional and non-functional broker inputs.

*Functional Broker Inputs.* If the service component has a provided interface, then each operation type, id and name is appended to the Dino Service representation as provided operations. For a provider of operations, building the Dino Service operations is relatively straightforward. However for a requester type service, the connector between service requester and provider instances must be referenced.

*Non-Functional Broker Inputs.* We also support extracting ModeConstraints for service bindings, or more specifically a quality of service attribute applied to assembly connectors between two or more components. A ModeConstraint is expected to be expressed in a particular format. The QoSRequired constraint consists of two key aspects. Firstly, that it has applied the QoSRequired stereotype from the QoS Profile, and secondly that it specifies an OCL statement which constraints the binding operation. We have developed a prototype tool suite to mechanically support the approach steps described in this paper as part of our service engineering tool suite, known as WS-Engineer, which is available from <http://www.doc.ic.ac.uk/ltsa/wsengineer>.

## 6 Conclusions and Future Work

We believe that the notion of Service Modes helps service engineers abstract appropriate elements, behaviour and policy from the services domain, and can facilitate the specification of appropriate control over both architectural change and service behaviour. In this paper we have presented our approach to the modelling of service-oriented computing component architectures using an abstraction of modes to represent the changes in such an architecture. Using a UML2 Modes Profile allowed us to define different capabilities and requirements for dynamic service brokering. Our future work will explore how mode configurations and their constraints are analysed for completeness and correctness. We are also seeking to implement the generation of service orchestrations and choreography from service mode behaviour specifications. This work has been partially sponsored by the EU funded project SENSORIA (IST-2005-016004). David Rosenblum holds a Wolfson Research Merit Award from the Royal Society.

## References

1. Ermagan, V., Krüger, I.H.: A uml2 profile for service modeling. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 360–374. Springer, Heidelberg (2007)
2. Foster, H., Uchitel, S., Kramer, J., Magee, J.: Towards self-management in service-oriented computing with modes. In: Workshop on Engineering Service-Oriented Applications, Vienna, Austria (2007)
3. Foster, H., Uchitel, S., Kramer, J., Magee, J.: Leveraging Modes and UML2 for Service Brokering Specifications. In: 4th Model-Driven Web Engineering Workshop (MDWE), Toulouse, France (2008)
4. Object Management Group. Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms. Proposal-AD/02-01/07 (2002)
5. Hirsch, D., Kramer, J., Magee, J., Uchitel, S.: Modes for software architectures. In: Third European Workshop on Software Architecture. Springer, Heidelberg (2006)
6. Johnston, S.: Uml 2.0 profile for software services (2005), <http://www-128.ibm.com/developerworks/rational/library/05/419soa>
7. Koch, N., Mayer, P., Heckel, R., Gonczy, L., Montangero, C.: D1.4b: Uml for service-oriented systems. Technical report (October 2007)
8. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of uml models for service-oriented software architectures. In: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Washington, DC, USA, pp. 173–182 (2005)
9. Mukhija, A., Dingwall-Smith, A., Rosenblum, D.S.: QoS-aware service composition in Dino. In: Proceedings of the 5th IEEE European Conference on Web Services (ECOWS 2007) (November 2007)
10. Yu, T., Lin, K.-J.: A broker-based framework for QoS-aware web service composition. In: Proceedings of the International Conference on e-Technology, e-Commerce and e-Service (EEE 2005) (March-April 2005)
11. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)

# Integrated Security Context Management of Web Components and Services in Federated Identity Environments

Apurva Kumar

IBM India Research Lab. 4, Block C Vasant Kunj Institutional Area,  
New Delhi, India-110070  
kapurva@in.ibm.com

**Abstract.** The problem of providing unified web security management in an environment with multiple autonomous security domains is considered. Security vendors provide separate security management solutions for cross-domain browser based and web service based interactions. This is partly due to the fact that different web standards dominate in each space. E.g. Security Assertion Markup Language (SAML) which is an important standard in cross domain single sign on (SSO) specializes in browser based access while WS-\* standards focus on security needs of web services. However, cross domain web services are often invoked in context of a secure browser session. Considering these interactions in isolation will lead to a fractured security solution. This paper proposes a solution that provides seamless transfer of security context across various types of cross-domain web interactions.

## 1 Introduction

Web is increasingly becoming the dominant channel for customer interaction with service providers. Providers compete with each other to enable the widest range of services on their websites. However, very few providers are diverse enough to be able to satisfy a wide range of services without interacting with their partners. The interaction can either be an explicit redirection to a partner or it could involve implicit invocation of web interfaces exposed by the partner. Needless to say, such interactions should be both secure as well as trusted by partnering organizations. One of the fundamental problems in designing such security solutions is the lack of a common source of identity information.

Federated identity management solutions address the problem by having authentication and attribute authorities that are trusted by all partners. Such solutions provide secure token exchange mechanisms to convey assertions about authenticated identity to multiple service providers.

At present, there are two standardization efforts in this space. Security Assertion Markup Language (SAML) [1,2] is an important and widely used federated identity management standard from OASIS Security Services Technical Committee. The single most important problem that SAML tries to solve is the Web Browser Single Sign-On (SSO) problem. A set of WS standards (WS-Security [3], WS-Trust [4],

WS-SecureConversation [5] and WS-Federation) collectively address the problem in web services domain. Since the two solutions address problems from seemingly different domains: machine to machine interactions and browser based interactions it might seem reasonable that organizations use both of them independently depending on the type of interaction.

However, very often, interactions between machines are driven by a human action of clicking a link or button on a website. If such interactions pass through organization boundaries, it is often important to propagate the security context. If the security context of the browser interaction is not passed to a web service invoked at a partner organization, it will lead to inferior solutions that compromise privacy or result in unnecessarily increased trust level between partners. Such factors will certainly limit the capacity of web as a medium for carrying out secure business transactions. In this paper we take an example of a telecom service provider to illustrate how existing solutions do not provide satisfactory solution for the problem. We propose a solution that extends the SAML browser based SSO use case to incorporate additional requirement arising out of the more complex interactions required in the trust model. The solution is based on introducing a new type of assertion called a '*Resource Request Assertion (RRA)*'.

## 2 Case Study of a Telecom Service Provider

### 2.1 Problem Description

Consider a telecom service provider (denoted as SP) having a customer portal where it allows its customers to purchase content (e.g. ring tones, wallpapers, music etc). SP does not host its own content but depends on a content provider (CP). CP supports advanced algorithms for rating of content based on subscriber usage. The agreed revenue model is that CP will charge SP based on the usage profile of the customer, charging less for a heavy (frequent) user and more for a light user. Communication between the two organizations is through two web services exposed by the CP for: *browsing/rating* service and *purchase* service.

When a customer logs on to the SP portal and wants to view a page of contents before purchase, the browsing/rating CP web service is invoked and the customer identity is passed as a parameter. The prices displayed to the customer are set by SP based on rating information received from CP for the customer. As customer continues browsing this interface is invoked multiple times by the SP portal application. The customer then chooses to purchase a content in response to which SP charges the customer and sends a request to the purchase interface of the CP web service. In response, CP provides a download URL to SP. The download URL is in CP domain.

To manage these transactions securely, the two parties approach another service provider (IDP) which provides identity management solutions. They agree on trusting IDP for authentication and as an authority for issuing, validating and exchanging tokens. The IDP sets up a customer repository which is maintained synchronized with the customer master (e.g. a CRM database) of SP. The identity provider website also provides web registration facility to customers of SP.

For CP, its content is the key and it wants to ensure that SP should not be in a position to take advantage of the charging model. For SP, its subscriber base is the key asset and it wants to ensure that its subscriber details are not misused or divulged to other parties. We now consider some solutions based on available federated identity management technologies.

In the following discussions, we assume there are four types of links on the SP website. *GUEST* links are the only ones which can be browsed without sign in. *BROWSE* links are those that require access to browse/rating web service from CP. *PURCHASE* links are those which require access purchase web service from CP. *DOWNLOAD* links are those which are redirected to CP website for downloading purchased content.

### 2.1.1 Solution Approach 1: SP Asserts Customer Identity

IDP proposes the following first solution in which it handles browser and web service interactions using a uniform approach, but independent of each other. For browser based interaction, the federated identity is that of the end user. For web service based transactions, the authenticated identity is an application ID, which identifies an SP application that invokes the web service exposed by the CP.

Figure 1 shows the steps executed in a typical user session in which user browses and then selects a content to purchase and download. These steps are described below:

*Step 1:* Customer connects to the SP website and browses.

*Step 2:* On clicking at a *BROWSE* link, the browser is redirected to the IDP website. An authentication request is encoded in the redirection URL.

*Step 3:* The IDP site throws a password challenge page to the customer.

*Step 4:* The user credentials are validated by the IDP and a token is issued. The token is digitally signed by the IDP. Also a security context is created for the user. The token is embedded in an HTML page returned to the browser (e.g. as a hidden form control).

*Step 5:* An auto-submit script causes the token to be HTTP POSTed to a URL in SP domain which is a consumer of assertions provided by the IDP.

*Step 6:* The token is validated and a new security context is created at SP for the customer and the customer is logged in. The originally requested URL is retrieved and forwarded to the SP application. In processing the *BROWSE* request, the SP application needs to invoke the CP browsing/rating web service. Since all customer requests are routed through the same application, the application is already signed in with the IDP and shares a security context with the web service (e.g. through a WS-SecureConversation [5] secure context token, SCT). The CP web service is invoked and the customer identity is passed as a parameter.

*Step 7:* CP confirms the security context is valid and then processes the request. The customer identity is used to provide rating based on customer usage. SP uses the result of the web service, maps rating points to prices and displays the catalogue/content to the customer. Steps 6-7 might be repeated multiple times, till the user decides to buy an item.



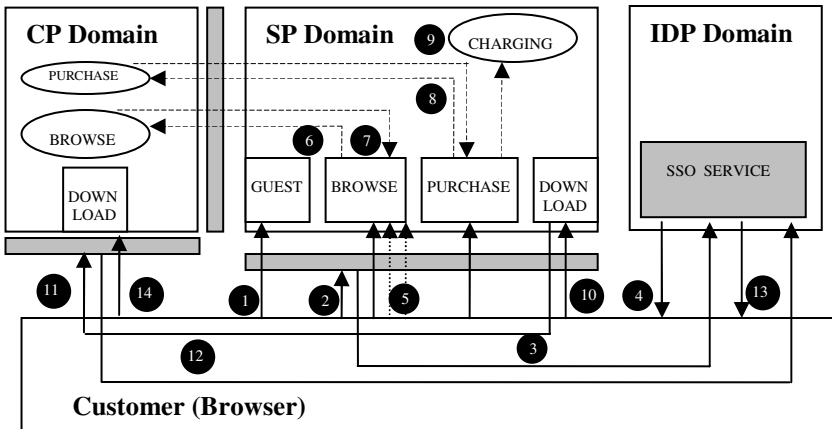
*Step 8:* The customer selects a content item and clicks on PURCHASE link. The SP web application initiates a charging request for the customer (e.g. by calling a web service in its own domain). After successful charging, SP invokes the purchase web service of the CP. The security context (e.g. based on an SCT) used in Step 6 is used.

*Step 9:* The CP returns the download URL in its own domain, which is displayed as a link on the SP website to the customer.

*Step 10:* The customer follows the link and is redirected to the CP website.

*Steps 11-14:* The sign on steps 1-4 are repeated for CP. However, this time authentication of customer is not required, since the browser already has a security context with the IDP. Once the token is verified by CP, it allows the user to access the content.

**Summary.** In an SAML based solution, steps 1-5 correspond to SAML Browser SSO profile [2]. Steps 6-7 correspond to accessing the browsing/rating web service. Step 8-9 corresponds to accessing the purchase web service. Step 10-14 correspond to downloading the content after the SAML browser SSO profile is repeated with the CP.



**Fig. 1.** Case Study: Sequence of events in solution approach 1

**Analysis of Trust Model.** The customer identity used by IDP can be a pseudonym rather than an identifier relevant to the business, thus the approach does not risk privacy of SP customer data. However, the trust model does not work quite so well for the CP. In Steps 6 and 8, it has to trust SP assertion about the identity of the customer. The revenue model is based on both the volume of each content item purchased as well as the purchaser. In this model, for the same sequence of contents downloaded the revenue for CP will be more if the content is accessed by light users as opposed to frequent users. If CP has to trust identity supplied by SP, it is possible for SP to replace light users by heavy users while asserting identity, thus bringing down the cost to be paid to CP.

## 2.2 Solution Approach 2: CP Controls Content Delivery

To address the above problem, IDP proposes an alternative approach in which CP controls delivery of content by sending it directly to the customer. This solution goes through the following flow:

*Steps 1-8:* Same as approach 1.

*Step 9:* CP returns the download URL (which is in its own domain) to SP. It also associates the URL with the pseudonym in the request.

*Step 10:* Customer accesses the DOWNLOAD link on SP website and is redirected to the CP website.

*Steps 11-14:* The sign on steps 1-4 are repeated for CP. Same as approach 1.

*Step 15:* CP sends an attribute request to the IDP with the pseudonym corresponding to the accessed URL for retrieving mobile number of the customer.

*Step 16:* The user is asked to confirm the mobile number for which the content was requested. Once user confirms, the content is delivered directly to the mobile device.

**Analysis of Trust Model.** This approach meets requirements for the CP since it is able to ensure that the content is delivered to a mobile number of the user asserted by the SP. However, this approach requires that CP has access to mobile numbers of SP subscribers. Since CP already has the usage profile for the customers, this knowledge allows CP to target subscribers of SPs network through other channels.

## 2.3 Need for Resource Request Assertion

Both the solutions described though feasible do not satisfy all the requirements of collaborating parties. The basic issue is that when a web service is called in the context of a browser SSO session, there is no secure means of passing the identity information from browser to the service. The problem arises because the access from SP to CP is direct without involvement of the IDP or the browser.

As a trusted party, it would have been ideal if IDP had certified that the browse/purchase URL for which the web service has been invoked was accessed by the customer. We call a statement binding an authenticated subject with a resource as a *Resource Request Assertion (RRA)*. However, this type of assertion is not included in the assertions types available in major federated identity management standards. E.g. SAML supports authentication, authorization and attribute assertions.

We now outline strategy for solving the integrated browser SSO and web service security problem. First, we should use extensibility of XML based federated identity standards to define a new assertion type: resource request assertion. Next, we should incorporate request and response messages for the new token in the browser based SSO flow. Finally, we use this token to propagate browser security context to the web service. In the following section we propose a solution based on this strategy.

### 3 Proposed Approach for Integrated Web Security Context Management

In the proposed approach (Figure 2), we use a Resource Request Assertion (RRA).

*Step 1-7:* Identical to approach 1. The BROWSE links are accessed as before.

*Step 8:* Customer selects a content item and clicks on PURCHASE link.

*Step 9:* The browser is redirected to the IDP website with a request for an RRA token. The requested link can be passed in the name field of the subject element in an SAML exchange.

*Step 10:* IDP verifies that a login context for the user exists. It then retrieves the requested URL and presents a page to the customer to confirm that he has requested access to the URL.

*Step 11:* User confirms the access request. IDP issues a signed RRA token which binds the URL with the authenticated subject (user). The token also contains the time of the request as well as validity period. It embeds the token in an HTML FORM control and returns the form to the browser.

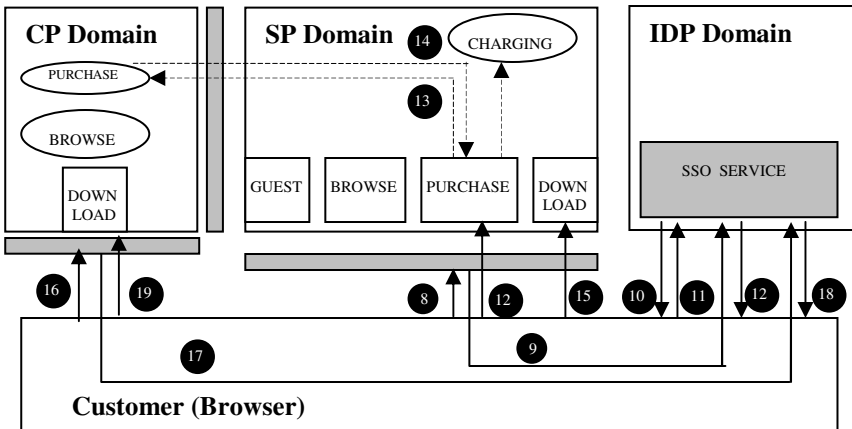
*Step 12:* An auto-submit script is executed on the browser which POSTs the form to the assertion consumer service of SP.

*Step 13:* The assertion consumer service forwards the request to the SP application after inserting the RRA token as an HTTP header. During processing of the request, the SP application needs to invoke the CP purchase web service to get the content URL. As in solution 1, we assume that the web application is already logged in to use the web service through IDP. The SP web application initiates a charging request for the customer (e.g. by calling a web service in its own domain). Finally the purchase interface of web service is invoked. The RRA token in the HTTP header of the request is used to obtain a new security context based on the existing one. Details are omitted due to space limitation, but this is done using WS-Trust token exchange facility.

*Step 14:* CP associates the new security context with the user (customer) identifier in the RRA token. It confirms that the user identifier passed in the purchase request matches with that in the token. *This is the key step instrumental in solving the trust problem of solution approach 1.* After this verification the request is allowed to proceed as earlier. Finally the download URL of the content is returned as a result of the call.

*Step 15-19:* Identical to Steps 10-14 in solution approach 1. The user clicks the download URL to be redirected to the SP DOWNLOAD page. The normal SAML browser SSO profile is executed between CP and IDP. Finally, the user is allowed to download the content.

**Summary.** In an SAML based solution, Steps 1-5 correspond to SAML Browser SSO profile for signing on SP website. Steps 6-7 correspond to accessing the browsing/rating web service. Step 8-12 corresponds to issue of an RRA token. This exchange is very similar to the browser SSO exchange. Steps 13-14 correspond to invoking the purchase web service at the CP. Steps 15-19 correspond to downloading the content after the SAML browser SSO profile is repeated with the CP.



**Fig. 2.** Case Study: Sequence of events in proposed solution for integrating Web Security Context Management

## 4 Conclusion

We demonstrate by means of a realistic case study, the problems that can arise in security solutions which ignore links between browser based and machine to machine communications. We propose a new type of assertion, called the Resource Request Assertion (RRA), which binds a subject with one or more requested resources (e.g. an HTTP URL). We use RRA as the means of controlling security context of a web service interaction on the basis of the browser security context. We use SAML as the browser SSO protocol and WS-SecureConversation and WS-Trust for web services security context management to provide a concrete framework for implementation of the solution. The RRA concept is powerful and it should be possible to use it to define other trust models not necessarily restricted to the federated identity domain.

## References

1. Cantor, S., et al.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
2. Hughes, J., et al.: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
3. Nadalin, A., et al.: Web Services Security: SOAP Message Security 1.0, WS-Security 2004 (2004), <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
4. Anderson, S., et al.: Web Services Trust Language (WS-Trust) (February 2005), <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-trust/ws-trust.pdf>
5. Anderson, S., et al.: Web Services Secure Conversation Language (WS-SecureConversation) (February 2005), <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secon/ws-secureconversation.pdf>

# Predicting and Learning Executability of Composite Web Services

Masahiro Tanaka and Toru Ishida

Department of Social Informatics, Kyoto University  
Kyoto 606-8501 Japan

mtanaka@ai.soc.i.kyoto-u.ac.jp, ishida@i.kyoto-u.ac.jp

**Abstract.** Configuring a composite Web service by setting endpoints reduces the cost of development, but raises the probability of a request message triggering runtime execution failures. Previous works on validation of composite Web services are not useful because the application developer cannot modify atomic/composite services and the specifications needed for validation are not always available. Therefore, in this paper, we address two issues: predicting the executability of composite Web services for each request message, and acquiring input specifications to improve the prediction. To resolve these issues, we model atomic/composite services in a formal specification. Moreover, we apply constraint acquisition algorithm to acquire input specifications of atomic Web services. We conduct an experiment in which the proposed method is applied to a composite Web service in practical use. The result shows that our method can detect almost all messages that will trigger execution failure at a rather early stage of specification acquisition.

## 1 Introduction

Various organizations have released Web services and standardized the interfaces of Web services. This makes it possible to develop composite Web services in the following way. The designer of a composite Web service provides his/her composite Web service in WS-BPEL or OWL-S through the combination of abstract atomic Web services, for which only the interfaces, and not the endpoints, are defined. We refer to such a composite Web service as an abstract composite Web service. The application developer simply sets endpoints for the abstract atomic Web services forming the abstract composite service; this identifies the concrete atomic Web services that will be actually invoked. We refer to such an implemented composite Web service as a concrete composite Web service.

However, a concrete composite Web service developed in the above way may suffer runtime failure. WSDL definitions for the abstract atomic Web service define only types of values of request messages, but do not define their valid range. Thus the execution of a concrete composite Web service which contains atomic Web services may fail for some request messages if it is configured by setting endpoints for the abstract atomic Web services. When a request message triggers execution failure of any atomic Web service in the composite Web service, the cost of executing all prior atomic Web services that is wasted.

Previous works have proposed methods with verification techniques such as petri net [1] and model checking [2,3] in order to prevent execution failure. However, these previous works are not useful because providers of concrete atomic Web services, designers of abstract composite Web services, and application developers reside in different organizations. Even if an application developer verifies a concrete composite Web and determines that it might suffer runtime failure, he/she can modify neither the concrete atomic Web services nor the abstract composite Web service. Moreover, the application developer cannot always perform the verification because the specifications of concrete Web services required for the verification are often unavailable.

Therefore, in this paper, we address the following issues:

- To ensure that a composite Web service is executable, we need to predict the executability of the composite Web service for each request message based on as much specifications as is known.
- To improve the accuracy of executability prediction, we need to acquire the specifications of concrete atomic Web services based on a success/failure of execution for each request message.

To predict executability, we model atomic and composite Web services in a formal specification. Moreover, we apply a constraint acquisition algorithm [4] to acquiring the input specifications of concrete atomic Web services.

## 2 Formal Specification to Model Web Services

To predict the executability of a composite Web service, we need specifications about input/output relation (Request message to response message mapping) and input specification (Request message validity) for each constituent atomic Web services.

To allow the above specifications to be checked, we model an atomic service as a module in the formal algebraic specification CafeOBJ [5], that consists of the following two operations.

**domain-service-name** This operation represents the input specification. This takes a request message to the Web service and returns true or false. True means executable and false means not executable.

**execute-service-name** This operation represents the input/output relation. This takes a request message to an atomic Web service and returns the response message of the atomic Web service.

We note, however, the input specification and input/output relations are not always completely known. Moreover, in general, it is impossible to describe the input/output relations completely. This is why we describe constraints on values or types of elements of request/response messages as far as are known.

Figure 1 shows the specifications of a machine translator Web service. First the definitions of data types and messages are imported (line 2). The response

```

1: mod TRANSLATOR {
2:   pr (LANGUAGE + TRANSLATOR-REQUEST + TRANSLATOR-RESPONSE)
3:
4:   op domain-translator : TranslatorRequest -> Bool
5:   op execute-translator : TranslatorRequest
                               -> TranslatorResponse
6:
7:   var e : TranslatorRequest
8:
9:   -- Source language must be English or Japanese
10:  eq domain-translator(e) =
11:    1*(e) == english or 1*(e) == japanese .
12:  -- Language of result is specified by target language
13:  eq get-language(execute-translator(e)) = 2*(e) .
14: }

```

**Fig. 1.** Specification of a machine translator service

message named `TranslatorResponse` is a string which is the result of the translation into the target language. Next, the two operations which represent input specification and input/output specification are declared (lines 4-5). Finally, axioms are described as equations following `eq` (lines 10-11,13). In this example, the first axiom states that the first value of the request message (source language) must be English or Japanese. The second axiom states that the Web service translates a given string into the language that is specified by the second value of the request message (target language). “`n*`” is an operation on N-tuple which extracts the *n*th value.

Predicting the executability of a composite Web service requires the specifications of the composite Web service. Our approach is to create the specifications of a composite Web service by combining the specifications of its constituent atomic Web services.

In OWL-S or WS-BPEL, a composite Web service has nested structures. A control construct block contains atomic Web services or other control construct blocks. We follow this and recursively define the specifications of control construct blocks. To allow this, we consider a control construct block as a Web service and define it using the request/response message, the input specification, and the input/output relation. The block that contains all other blocks and atomic Web services corresponds to the composite Web service. We define two operations to represent the input specification and the input/output relation in the specification of each control construct block. Dataflows and constraints based on features of control constructs are represented as axioms in the specification.

### 3 Acquiring Input Specifications

Complete specifications of Web services for the prediction are not always known, especially in the case of Web services. Therefore, we propose a method that acquires the input specifications of atomic Web services to improve prediction accuracy.

In our model described in the previous section, input specifications of a Web service are represented as a logical formula. Thus we adopt the constraint

acquisition algorithm [4] to acquire the input specifications because the result of the acquisition can be represented as a logical formula in the formal specification.

In our method, a request message to an atomic Web service and the success/failure of the execution of the message are given to the constraint acquisition algorithm as a training example. The acquisition result can be easily transformed into descriptions in the formal specification by defining logical formulas in the formal specification that correspond to predefined predicates.

We explain below how to model the input specifications to apply the constraint acquisition algorithm. First we define a request message to a service which has  $k$  elements as  $I = \{x_1, \dots, x_k\}$ . Next we define predicates which represent input constraints. For the sake of simplicity, we assume that the predicates have one or two variables. We refer to a unary constraint on an input value  $x_i$  to  $i$ th parameter as  $b_i$ . We also refer to a binary constraint on input values  $x_i, x_j$  to  $i$ th and  $j$ th parameters as  $b_{(i,j)}$ .

$b_i$  and  $b_{(i,j)}$  can be defined as  $b_i : x_i \in \text{class}$  or  $x_i \notin \text{class}$  and  $b_{i,j} : \{x_i, x_j\} \in \text{class1} \times \text{class2}$  or  $\{x_i, x_j\} \notin \text{class1} \times \text{class2}$  respectively. *class*, *class1* and *class2* represent any class.  $x_i \in \text{class}$  indicates that  $x_i$  is an instance of *class*.  $\text{class1} \times \text{class2}$  represents a Cartesian product set of *class1* and *class2*. Constraint library  $B_s$  contains  $b_i$  and  $b_{(i,j)}$  for all known classes or pairs of classes.

The constraint acquisition algorithm works based on the above formalization. When a positive example is given, the constraint acquisition algorithm adds to formula  $K$  ( $K = \text{true}$  in the initial state) the conjunction of negation of all constraints in the constraint library that the example does not satisfy. When a negative example given, it adds to  $K$  the disjunction of all constraints in the constraint library that the example does not satisfy. The acquisition result is the conjunction of literals that should be set to true in order to satisfy  $K$ .

In general, the set of literals that satisfy  $K$  is not unique. Thus we consider the possible sets of literals that satisfy  $K$  as a set of hypotheses,  $H$ , and define a partial order  $\prec$  between hypotheses in  $H$  as follows:  $h_i \prec h_j \equiv (\forall x \in X)[h_i(x) = \text{true} \rightarrow h_j(x) = \text{true}]$ .  $X$  is a set of possible values of a request message.  $h_i(x) = \text{true}$  means that the atomic Web service is executable for request message  $x \in X$  under the hypothesis  $h_i$ . Our method performs prediction by reducing operation *domain-service-name* under all  $h_g$  defined as follows:  $\{h_g \in H | h_g \not\prec (\forall h \in H)\}$ . Our method cancels execution only if the results of reducing under all  $h_g$  are **false**. This is because our prediction of executability involves detecting request messages that would cause service execution to certainly fail.

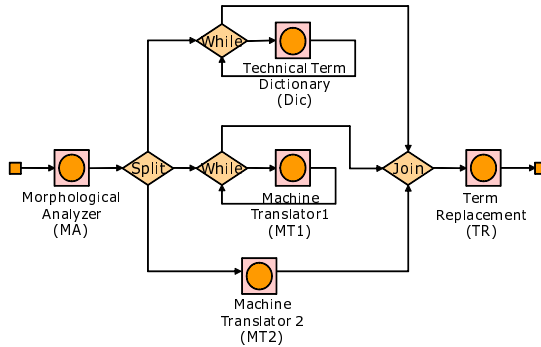
## 4 Experiment

We conducted an experiment to show how much our method can improve the efficiency of executing a composite Web service. We applied our method to a composite Web service shown in Fig. 2, which is for translation used in Language Grid [6]

The details of the process of the composite Web service are as follows:

1. Split the string given as a request message into words using Morphological Analyzer (MA)





**Fig. 2.** Composite service for translation in a special domain

2. Concurrently execute the followings:
  - (a) Translate all the words by Technical Term Dictionary (Dic)
  - (b) Translate all the words by Machine Translator 1 (MT1)
  - (c) Translate the string given as a request message by Machine Translator 2 (MT2)
3. Term Replacement (TR) replaces words in the string translated by MT2 with corresponding words translated by Dic

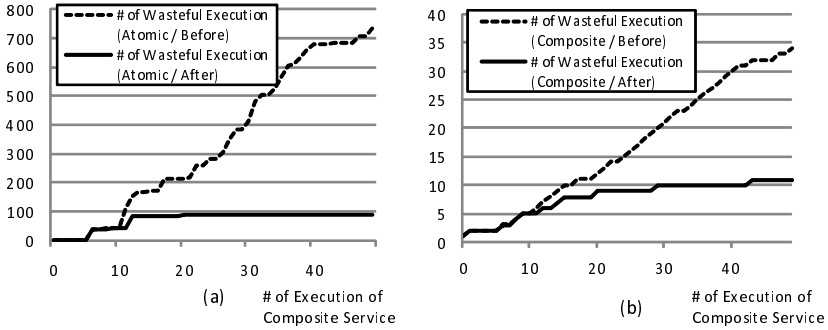
Suppose MA, MT1 and MT2 have the following input specifications.

- MA : Fail if any language other than Japanese or English is specified for the language of the given string.
- MT1, MT2 : Fail if the specified source language is different from the actual language of the given string or the given string is longer than 100 characters.

These input specifications lead to the possible failure of execution of the composite Web service as shown below.

- When the source language is neither Japanese nor English, MA fails and the cost of MA execution goes counted as waste.
- When the given string is longer than 100 characters, MT2 fails and the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic is counted as waste.
- When the given string has words in multiple languages (e.g. Japanese sentences often contain English words.), MT1 fails because the actual language of some of the given words differ from the source language specified. In this case, the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic are counted as waste as in the previous case.

We applied our method under the conditions described above. We assume that all input specifications of the atomic Web services are unknown in the initial state. Moreover, we defined predicates for the constraint acquisition algorithm. The predicates involve the input specifications of MA, MT1 and MT2. They



**Fig. 3.** Number of wasteful execution of atomic/composite services

represent classes for the source language, the target language, and the actual language of the given string (three classes for each) as described in Section 3. In this experiment, we also defined predicates to represent given string length (two classes: short and long). We generated request messages of all combinations of the classes for each element and executed the composite Web service by giving the messages in random order.

We counted failure of the execution of the composite Web service due to failure of any of the atomic Web services as one of the wasteful executions of the composite Web service. Similarly, we counted the sum of the executions of atomic Web services until one of the atomic Web service failed as the number of wasteful executions of atomic Web services. Figure 3(a)(b) compares the numbers of wasteful executions of the atomic/composite Web services shown in Fig. 2 before and after applying our method, respectively.

The figures show that the rate of increase in the number of wasteful executions saturates as the number of execution increases and more input specifications are acquired. In particular, Fig. 3(a) shows that our method works well in our example because it prevents some atomic Web services from being iteratively executed after the failure of some atomic Web service. This is very effective in reducing the cost of executing atomic Web services.

## 5 Conclusion

In this paper, we proposed a method for predicting the executability of composite Web services in order to reduce the cost of wasteful executions of atomic Web services. The major contributions of our method are as follows:

- We showed a model of Web services in a formal specification and applied it to predict the executability of a composite Web service for each request message by using a theorem prover.
- We applied the constraint acquisition algorithm in order to acquire input specifications of atomic Web services and showed that it improves the prediction of executability.

We conducted an experiment in which our method was applied to a composite Web service in practical use. The results showed that our method could detect almost all request messages that would cause execution failure. Compared to previous works, we assume that the application developer cannot modify concrete atomic Web services or abstract composite Web services because the stakeholders are in different organizations. This paper is the first work that focuses on the point and tries to reduce the wasteful execution of Web services by predicting the executability of each request message.

## Acknowledgments

This work was partially supported by Grant-in-Aid for JSPS Fellows and Global COE Program “Informatics Education and Research Center for Knowledge-Circulating Society”.

## References

1. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: The 11th International Conference on World Wide Web (WWW 2002), pp. 77–88 (2002)
2. Ankolekar, A., Paolucci, M., Sycara, K.: Towards a formal verification of owl-s process models. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 37–51. Springer, Heidelberg (2005)
3. Fu, X., Bultan, T., Su, J.: Analysis of interacting bpel web services. In: The 13th conference on World Wide Web (WWW 2004), pp. 621–630 (2004)
4. Bessière, C., Coletta, R., O’Sullivan, B., Paulin, M.: Query-driven constraint acquisition. In: The 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 50–55 (2007)
5. Futatsugi, K., Nakagawa, A.: An overview of cafe specification environment—an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In: The 1st International Conference on Formal Engineering Methods, pp. 170–181 (1997)
6. Ishida, T.: Language grid: An infrastructure for intercultural collaboration. In: IEEE/IPSJ Symposium on Applications and the Internet (SAINT 2006), pp. 96–100 (2006)

# Authorization Policy Based Business Collaboration Reliability Verification

Haiyang Sun<sup>1</sup>, Xin Wang<sup>2</sup>, Jian Yang<sup>1</sup>, and Yanchun Zhang<sup>2</sup>

<sup>1</sup> Department of Computing, Macquarie University,  
Sydney, NSW2109, Australia  
{hsun, jian}@ics.mq.edu.au

<sup>2</sup> School of Computer Science and Mathematics, Victoria University,  
Melbourne, Victoria, Australia  
xin@csm.vu.edu.au, Yanchun.zhang@vu.edu.au

**Abstract.** Collaborative business can become unreliable in terms of authorization policy conflicts, for example, when (1) incorrect role assignment or modification occurs in a service within one organization or (2) messages transferred from one organization are accessed by unqualified roles in other collaborating business partners. Therefore reliability verification based on access policies is critical for business collaboration. In this paper, a role authorization model, Role-Net, is developed based on Hierarchical Colored Petri Nets (HCPNs) to specify and manage role authorization in business collaboration and to verify collaboration reliability according to partners' authorization policies.

## 1 Introduction

Emerging web service and business process technologies have provided technological support for business collaboration across organization boundaries [1]. However, security concerns have become one of the main barriers that prevent its widespread adoption [2]. Models and methods are therefore required to manage secured business collaboration.

Role Based Access Control (RBAC) [3] is a popular security paradigm where users are assigned with roles in order to gain certain permissions to access messages or perform tasks. Hence, RBAC is normally used to define authorization policy for managing tasks in an organization [4]. However, in collaborative business environment, organization not only requires the correct role assignment to access messages for its own services, but also the right role to access the messages it passes to its collaborating partners. But business collaboration is peer based and services are autonomous. Authorization policies defined for individual organizations normally can not be seen by others. Therefore, in order to guarantee that the messages transferred among organizations can be accessed by the qualified roles in business collaboration, each organization need to send its collaborators the required role information together with messages to be accessed at collaborators' service. Based on this assumption, a message transferred and processed between services can be associated with two types of roles: one is the

access role of the current service, the other is the required role for the next service. The authorization policies in various organizations can then be coordinated to enforce access control in business collaboration.

Even with the above assumed setting, business collaboration can still become unreliable in terms of authorization policy conflicts occurred within or across organizations. For example, (1) within one organization, any message shall be associated with a required role before it can be processed in a service, and an actual role assigned to process the message in the service. If this 'required role' is not consistent with the role assignment for this coming message in the service, we can conclude that the role assignment is incorrect and authorization policy has conflict. Let us look at another example. (2) Business partners are peers with their own authorization policies that are agnostic to each other. Therefore, without central control, it is difficult to guarantee that the message is accessed by the qualified roles in business partners' service. Therefore, in this paper, we propose a Role-Net model which is developed based on Hierarchical Colored Petri Nets (HCPNs). Role-Net provides a verification mechanism to detect the authorization policy conflicts within or across organizations.

The rest of paper is organized as follows. In section 2, we introduce the structure of Role-Net, followed by presenting the execution policy. Related work is discussed in section 4 while conclusion is presented in section 5.

## 2 Structure of Role-Net

Role-Net is a role-authorization oriented, petri-net based model to simulate business collaboration for each participating organization. A *Role* is modeled as a *RO-Token* in Role-Net. Its movement among consecutive *transitions* thereby models the role assignment at specific services, which consequently generates a role flow. However, before a *Place*, the RO-Token is called **Operational Role** which represents the role who accesses and modifies the message at previous service; while after a *place* the RO-Token represents **Required Role** which is used to describe the set of roles required in the next service. AO-Token is another type of token operated in Role-Net, whose movement represents message flow. (The petri net terminology, e.g., *Place* and *Transition*, will be explained in following sections).

Role-Net is separated into two layers to model the inter-organizational role authorization in business collaboration. (1) The upper layer of Role-Net is used to describe the role-based authorization policy within local organization only. (2) The lower layer of one organization's Role-Net models the authorization policy of the services of the collaborators' with which the organization is interacting. In other words, if the service in local organization requires business interaction with its collaborators, the local organization's projection on collaborators' Role-Net will be modeled as the lower layer of local organization's Role-Net. We present the formal definition of Role-Net's two layers as follows:

**Definition 1.** *The upper layer of organization  $G_i$ 's Role-Net is a tuple  $\rho_{G_i}^{upper} = (P_{G_i}^{upper}, T_{G_i}^{upper}, F_{G_i}^{upper}, I_{G_i}^{upper}, \Delta_{G_i}^{upper}, \Theta_{G_i}^{upper}, \Omega_{G_i}^{upper})$ , where:*

- $P_{G_i}^{upper}$  is a set of places in upper layer of  $G_i$ 's Role-Net which graphically are represented as circles in Fig. 1 and model the state of collaboration.
- $T_{G_i}^{upper}$  is a set of transitions graphically represented as dark bars in upper layer of Role-Net in Fig. 1. Transitions are used to model services and implement corresponding functions.  $P_{G_i}^{upper} \cap T_{G_i}^{upper} = NULL$ .
- $F_{G_i}^{upper} = (p^u \times V \times t^u) \cup (t^u \times V \times p^u)$  is the flow relation between places and transitions representing the execution order of services in business collaboration, where  $p^u \in P_{G_i}^{upper}$ ,  $t^u \in T_{G_i}^{upper}$ , and  $V$  is the sets of variables  $V = \{x, y, \dots\}$  to represent the tokens.
- $\Gamma_{G_i}^{upper}(p^u, a, r) \rightarrow \text{Boolean}$  is a correlation function to evaluate the relationship of RO-Token and AO-Token at specific place, where  $a \in \text{AO-Token}$ ,  $r \in \text{RO-Token}$ , and  $p^u \in P_{G_i}^{upper}$ .  $\Gamma_{G_i}^{upper}$  guarantees that the AO-Token can only be moved with assigned RO-Tokens at specific places.
- $\Delta_{G_i}^{upper}(p^u, a, r) \rightarrow r^\varepsilon$  is a function to change RO-Token from representing role (operational role  $r$ ) that accessed the AO-Token at previous transition to indicating the roles (required roles  $r^\varepsilon$ ) which are needed by the next transition, according to role authorization policies, where  $p^u \in P_{G_i}^{upper}$ ,  $a \in \text{AO-Token}$ ,  $r, r^\varepsilon \in \text{RO-Token}$ .
- $\Theta_{G_i}^{upper}(t^u, \varphi, r^\varepsilon) \rightarrow \text{Boolean}$  is comparison function, where  $t^u \in T_{G_i}^{upper}$ ,  $r^\varepsilon \in \text{RO-Token}$ , and  $\varphi$  is a threshold variable representing the role element selected from the set  $\gamma$ .  $\gamma$  is the set of roles that are permitted to access and modify the AO-Token in the transition at upper layer of  $G_i$ 's Role-Net, named as available role set. The TRUE result of  $\Theta_{G_i}^{upper}$  function reflects the existence of qualified roles for specific transition  $t$ .
- $\Omega_{G_i}^{upper}(t_\beta^u, a) \rightarrow L$  is refinement function on transition  $t_\beta^u$  to connect Role-Net's lower layer, where  $t_\beta^u \in T_{G_i}^{upper}$  represents transition including link between two layers of Role-Net.  $L = \{g(x), \{e(x), \rho_{G_i}^{lower}, r(x)\}\} x \in V$ .  $g(x)$  is a function to evaluate the token and decide which  $\rho_{G_i}^{lower}$  shall be initiated in other collaborators.  $e(x)$  and  $r(x)$  are the guard functions of relevant lower layers to evaluate whether or not the  $\rho_{G_i}^{lower}$  is available to initiate and exit.

**Definition 2.** The lower layer of organization  $G_i$ 's Role-Net is a tuple  $\rho_{G_i}^{lower} = (P_{G_i}^{lower}, T_{G_i}^{lower}, F_{G_i}^{lower}, \Gamma_{G_i}^{lower}, \Psi_{G_i}^{lower})$ , where:

- $P_{G_i}^{lower}, T_{G_i}^{lower}, F_{G_i}^{lower}, \Gamma_{G_i}^{lower}$  are as same as relevant elements in upper layer of Role-Net.
- $\Psi_{G_i}^{lower}(t^l, a, r^\varepsilon) \rightarrow (b, R^\varepsilon)$  is a switch function to transfer the value of AO-Token and RO-Token from input of the transition in lower layer to output of the transition, where  $t^l \in T_{G_i}^{lower}$ ,  $a, b \in \text{AO-Token}$  ( $a \neq b$ ),  $r^\varepsilon, R^\varepsilon \in \text{RO-Token}$  ( $r^\varepsilon$  is required roles transferred from Role-Net's upper layer to lower layer.  $R^\varepsilon$  is the operational role in lower layer and is returned from Role-Net's lower layer to upper layer. When  $R^\varepsilon$  arrives at the upper layer, it is an input to  $\Theta_{G_i}^{upper}$  to detect the qualified role set). Any modification on AO-Token and RO-Token is unknown to organization  $G_i$ ' since it only observes the behavior of its collaborators through this lower layer of Role-Net. Hence, the switch function is only used to transfer the value of AO-Token and RO-Token after they have been modified according to collaborator's polices.

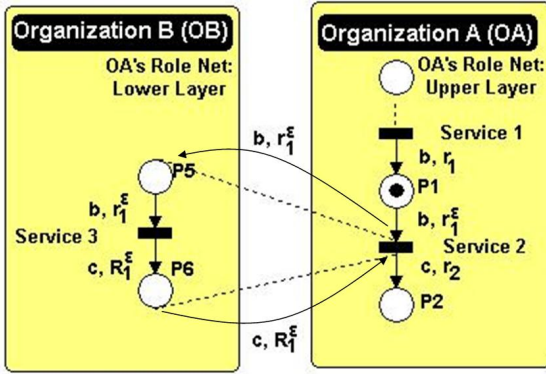


Fig. 1. Role-Net of Organization A

In Fig. 1, we illustrate a Role-Net of organization A (OA) which is separated into two layers. The upper layer models application process at OA side while lower layer simulates the projection of OA on organization B (OB)’s Role-Net. They are linked by Refinement Function at the transition which is represented as service 2.  $b$  and  $c$  in the figure are AO-Tokens which indicate the message transferred in the business collaboration.  $r_1, r_2, r_1^\epsilon$ , and  $R_1^\epsilon$  are RO-Tokens while  $r_1, r_2$  and  $R_1^\epsilon$  represent operational roles for each service and  $r_1^\epsilon$  is required role.

### 3 Execution Policy of Role-Net

There are two types of tokens that are operated within a Role-Net: the Application-Oriented Token (AO-Token) and the Role-Oriented Token (RO-Token) whose movements correspond to the *message flow* and *role flow*. The AO-Token will move together with the relevant RO-Token to correlate the message flow and role flow, which can guarantee that the desired message can only be accessed by the specific roles at the designated service. The execution policies of Role-Net are described as follows:

– **Token at Place**

- (1) Each RO-token is correlated to a specific AO-token. The *Correlation function*  $\Gamma_{G_i}$  in upper layer and lower layer of Role-Net will check the correlation of these two types of tokens at each place. If a RO-Token and an AO-Token are received separately, the Place will abandon the token as an unexpected role or message respectively.
- (2) (i) Before Places in upper layer, the RO-Token  $r$  represents the *Operational role* that has accessed the correlated message at previous Transition. After Places in upper layer, the RO-Token  $r^\epsilon$  will represent the *Required role* which will be required by the next transition. The Function  $\Delta_{G_i}^{upper}$  will deal with the transfer of RO-Token at each Place in upper layer.
- (ii) The place in lower layer is used to receive the AO-Token and RO-Token from upper layer, and return the two correlated tokens together to upper layer after they are processed by the services of the collaborating partners.

- **Token at Transition in Upper Layer of Organization  $G_i$ 's Role-Net.**

(1) If the link between upper layer and lower layer of Role-Net exists, AO-Token and RO-Token  $r^\varepsilon$  representing Required Roles will move together to the lower layer of Role-Net as cross-organizational message transfer. The refinement function  $\Omega_{G_i}^{upper}$  is used to identify the lower layer of organization  $G_i$ 's Role-net  $\rho_{G_i}^{lower}$  (the lower layer of local organization's Role-Net represents the local organization's view on its collaborator's Role-Net). When the modified AO-Token and RO-Token return from the lower layer, the transition in upper layer then invokes the *Comparison Function*  $\Theta_{G_i}^{upper}$  to identify the qualified roles. (2)  $\Theta_{G_i}^{upper}$  function is implemented to detect the qualified roles when Required Role  $r^\varepsilon$  arrives at transition in upper layer with AO-Token (no link between lower layer and upper layer in this transition) or returned RO-Token  $R^\varepsilon$  arrives at the transition with AO-Token from lower layer (link between lower layer and upper layer exists in this transition). (i) Each transition in upper layer of Organization  $G_i$ 's Role-Net has a set of available roles  $\gamma$  which are qualified to access message in this transition. However, depending on the properties of message and role authorization policies, all or part of them may not be authorized to process message at runtime. Therefore, a threshold  $\varphi$  is dynamically decided by choosing roles from  $\gamma$  at each transition. (If the transition in upper layer of Role-Net has link to lower layer, then  $\varphi$  is selected from  $R^\varepsilon$  and is input in function  $\Theta_{G_i}^{upper}$  to verify whether the AO-Token is modified by the Required Role  $r^\varepsilon$  at collaborator's Role-Net). (ii) If  $r^\varepsilon$ 's element equals to the threshold  $\varphi$ , the role in threshold will be moved to the set  $\varrho$  as qualified role to access the message in this *transition* and the threshold will be degraded for the next role in  $\gamma$ . The comparison will continue until all role elements in Required Roles  $r^\varepsilon$  and  $\gamma$  (or  $R^\varepsilon$ ) have been dealt with. (iii) Finally, if  $\varrho$  is not empty, then the role elements in this set will be authorized the permission to access the messages in this service. The RO-Token will thus represent the role that actually accesses the message and is moved with AO-Token together to the next places. If  $\varrho$  is empty, then there is no qualified role to deal with this message at this service. The process will be suspended due to the authorization policy conflicts. Therefore, by comparing  $\varphi$  with each role element in *Required Roles*  $r^\varepsilon$ , we can verify authorization policy based business collaboration reliability.
- **Token at Transition in Lower Layer of organization  $G_i$ 's Role-Net.**

$\Psi_{G_i}^{lower}$  in the transition of lower layer of organization  $G_i$ 's Role-Net is used to transfer the value of AO-Token and RO-Token. The switch function  $\Psi_{G_i}^{lower}$  can not identify how the value of AO-Token and RO-Token are changed in the transition, since local organization  $G_i$  is agnostic to its collaborator's internal process, including which role is assigned to process the message. Hence, these modifications on AO-Token and RO-Token are implemented according to collaborator's own authorization policies, and  $\Psi_{G_i}^{lower}$  in lower layer of local organization  $G_i$ 's Role-Net can only identify and exchange the result of modification on tokens.



## 4 Related Work

Research has been done in the area of role authorization in business collaboration. The authors in [5] focused on extending the specification WS-BPEL [6] with role authorization constraints in business collaboration. Liu and Chen [7] developed another extended RBAC model, WS-RBAC. Three new elements were introduced into the original RBAC model, namely enterprise, business process and web services. Knorr in [8] has proposed a role based access control method through Petri Net workflows. Role authorization rights were granted according to the state of the workflow. However, they are still insufficient in: (1) describing role authorization in business collaboration with regard to the organization's peer nature; (2) detecting role authorization errors and verifying business collaboration reliability in terms of role authorization.

## 5 Conclusion

Business collaboration can become unreliable in terms of authorization policy conflicts. Current approaches can not provide model to simulate role authorization in business collaboration, nor verification mechanism to enforce collaboration reliability in terms of authorization policy. In this paper, we provide a role authorization model (Role-Net) to verify authorization policy based business collaboration reliability.

## References

- [1] Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing: Introduction. *Communications of the ACM* 46-10, 24–28 (2003)
- [2] Wang, X., Zhang, Y., Shi, H., Yang, J.: BPEL4RBAC: An Authorisation Specification for WS-BPEL. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) *WISE 2008*. LNCS, vol. 5175, pp. 381–395. Springer, Heidelberg (2008)
- [3] Ferraiolo, D., Cugini, J., Kuhn, R.: Role Based Access Control: Features and Motivations. In: *Proceedings of Annual Computer Security Applications Conference*. IEEE Computer Society Press, Los Alamitos (1995)
- [4] Wang, H., Cao, J., Zhang, Y.: A flexible payment scheme and its role-based access control. *IEEE Transactions on Knowledge and Data Engineering* 17(3), 425–436 (2005)
- [5] Bertino, E., Crampton, J., Paci, F.: Access Control and Authorization Constraints for WS-BPEL. In: *Proceedings of ICWS* (2006)
- [6] OASIS Web Services Business Process Execution Language (WS-BPEL) Technical Committee. Web services business process execution language version 2.0, ws-bpel (2007), <http://docs.oasis-open.org/wsbpel/2.0/cs01/wsbpel-v2.0-cs01.html>
- [7] Liu, P., Chen, Z.: An Access Control Model for Web Services in Business Process. In: *Proceedings of WI* (2004)
- [8] Knorr, K.: Dynamic Access Control through Petri Net Workflows. In: *Proceedings of ACSAC* (2000)

# VGC: Generating Valid Global Communication Models of Composite Services Using Temporal Reasoning

Nalaka Gooneratne, Zahir Tari, and James Harland

School of Computer Science and Information Technology  
RMIT University,

Melbourne 3001, Australia

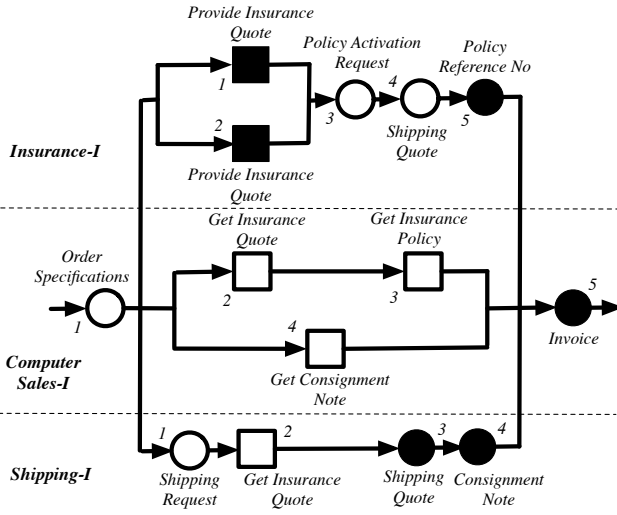
{nalaka.gooneratne,zahir.tari,james.harland}@rmit.edu.au

**Abstract.** As the range of services available on the Web increase, new value added services can be created by composing existing ones. It is then vital to ensure that compositions of web services are free from errors such as deadlocks and synchronisation conflicts. Current techniques are lacking in this regard because they either (i) do not consider all the different types of temporal relationships that exist between interactions, or (ii) do not support all types of interactions (i.e. only send and receive, not service and invoke). In this paper we introduce an approach that overcomes these problems. First, a communication model is generated by composing interactions of constituent services. Then, the temporal relationships between all the interactions of the communication model are found using a reasoning mechanism. While doing so, these relationships are compared against those specified in descriptions of interaction protocols, to detect any deadlocks or synchronisation conflicts.

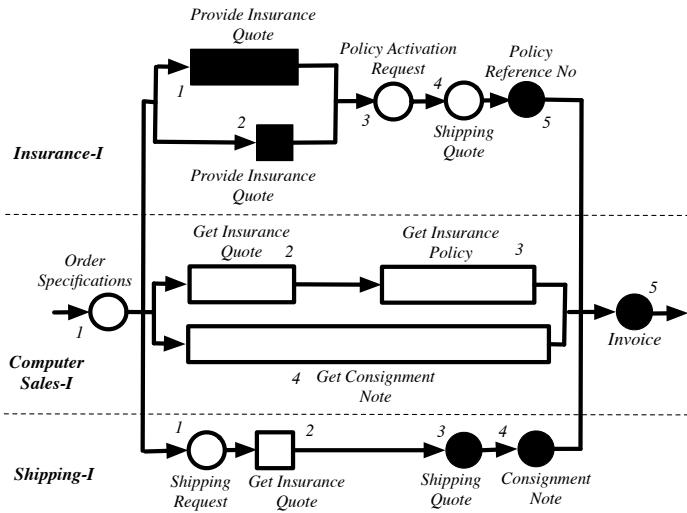
## 1 Introduction

An interaction protocol (of a web service) describes all the interactions as well as the temporal relationships (i.e. ordering constraints) between interactions. These interactions are of four types [4]: *send*, *receive*, *invoke* and *service*. These can be divided into two classes: (i) interactions that take place at time points (send and receive) and (ii) interactions that take place during time intervals (invoke and service). Existing techniques that generate valid global communication models of composite services have limitations, as they either do not consider the temporal aspects of interactions [3] (e.g. - time taken and different types of temporal relationships) or they only consider specific interactions like send and receive [4]. Modeling either an invoke or a service interaction with a send and a receive can be problematic when the same parameter is dispatched or accepted more than once [4]. Phantom deadlocks can be detected and (real) deadlocks may be missed when temporal aspects of interactions are ignored.

The model depicted in Figure 1a can be formed by composing the interaction protocols of three services (i.e. *ComputerSales-I*, *Shipping-I* and *Insurance-I*), when



(a) Temporal aspects of Interactions not considered



(b) Temporal aspects of Interactions considered

Fig. 1. Global Communication Models

the temporal aspects of interactions are not considered. A solid circle represents a “send” (interaction), an empty circle a “receive”, and a solid square a “service” and an empty square an “invoke”. On the other hand, the model depicted in Figure 1b can be generated if the temporal aspects of interactions are considered. This model is free of deadlocks and synchronisation conflicts because both the order in which the interactions are performed as well as the ordering constraints in the interaction protocols of the constituent services do not conflict.

This paper proposes a technique called VGC that addresses the stated issues. VGC generates valid global communication models by composing interactions protocols. First, conversations are formed by composing the interactions of constituent services. Then, sets of conversations that consist of all the interactions of constituent services are formed to ensure that a derived global communication model is free of unspecified receptions or transmissions [4]. Such sets of conversations are referred to as Complete Conversation Sets (CCS). Next, the temporal relationships between the interactions of a CCS are found using an Interval Time Logic (ITL)-based transitive temporal reasoning mechanism [1]. While doing so, deadlocks and synchronisation conflicts that could occur are detected by identifying inconsistencies in the temporal relationships. An inconsistency occurs if the temporal relationships between any two interactions (of a global communication model) conflict with those of an interaction protocol (of a constituent service). Finally, a concise specification of a global communication model is obtained by grouping the temporal relationships between interactions.

## 2 The Proposed VGC Approach

VGC makes the following assumptions: (i) the constituent services of a composite service are located using a service discovery technique [2] and (ii) the interaction protocols of services are specified using WS- $\pi$ -calculus [3]. WS- $\pi$ -calculus supports all four types of interactions and accurately models the temporal relationships between them. Given a set interaction protocols (say  $\{p_x, \dots, p_y\}$ ) of constituent services (say  $s_x, \dots, s_y$ ) of a composite service (say  $cs$ ), the proposed technique generates a specification of a valid global communication model  $gc$  of  $cs$ . VGC is divided into four steps.

**STEP 1.** First, all the conversations that could take place between the constituent services are determined and stored in a set of Conversation Lists. A conversation indicates how two or more interactions of constituent services should be composed and ensures that parameters can be exchanged between composed interactions.

**Definition 1 (Conversation).** *Given two distinct services  $s_a$  and  $s_b$  with interactions  $i_a$  (of  $s_a$ ),  $i_b$  and  $i'_b$  (of  $s_b$ ), we have:*

1.  $i_a$  and  $i_b$  form a conversation if  $i_a$  can be composed with  $i_b$
2.  $i_a$ ,  $i_b$  and  $i'_b$  form a conversation if  $i_a$  can be composed with  $i_b$  and  $i'_b$

Next, we introduce the concept of an *execution path*, which defines a sequence of interactions that can be used to achieve a functionality provided by a web service. Each web service could consist of multiple execution paths.

**Definition 2 (Execution Path).** *An execution path  $EP$  of a WS- $\pi$  interaction protocol  $P$  is a sub-process of  $P$ , where  $EP$  contains the first and the final interactions of  $P$ , and it does not contain any choice construct.*

As the service definitions may contain choice constructs, it is important to ensure that interactions are not taken from different execution paths. This is because interactions from different paths cannot be performed together in a single execution of a service (as they correspond to different choices made during execution). For this reason we need to work with combinations of execution paths, rather than the direct definitions of each service.

**Definition 3 (Combination of Execution Paths(CEP)).** *Let  $cs$  be a composite service formed with the constituent services  $\{s_1, \dots, s_n\}$ , where each  $s_i$  contains a set of execution paths  $E_i$ . A CEP is a tuple  $[e_1, \dots, e_n]$ , where each execution path  $e_i \in E_i$ .*

Next, a Conversation List is generated for each CEP. Such a list stores all the conversations that can take place between interactions of the execution paths in a CEP.

**STEP 2.** Sets of conversations that form Complete Conversation Sets (CCSs) are located from each Conversation List. A CCS is a set of conversations containing all the interactions of a CEP. Let  $f()$  be a function that returns the set of interactions included in an execution path or a conversation (which we will call an *interaction* function). Given a CEP  $P$  and a set of conversations  $C$ , where  $P = \{e_1, \dots, e_n\}$  and  $C = \{c_1, \dots, c_m\}$ ,  $C$  is a CCS of  $P$  if

$$\begin{aligned} f(e_1) \cup \dots \cup f(e_n) &\equiv f(c_1) \cup \dots \cup f(c_m) \text{ and} \\ 1 \leq i, j \leq m, f(c_i) \cap f(c_j) &= \emptyset \end{aligned} \quad (1)$$

A CCS ensures that a global communication model does not have any misses or overlaps. A *miss* occurs in a global communication model if this doesn't contain all the interactions of a CEP. An *overlap* occurs if either multiple interactions accept a single dispatched parameter or parameters dispatched by multiple interactions are accepted by a single interaction. Deadlocks and unspecified receptions occur in global communication models if there are overlaps or misses [4]. We model a Conversation List  $L$  as a graph  $G = \langle V, E \rangle$ , where  $V$  is a set of conversations and  $E$  is a set of edges that connects conversations which have common interactions. Then, sets of vertices that model CCSs are derived from such graphs.

**Definition 4 (Set of Vertices Modeling a CCS).** *Given a graph  $G$  generated for a Conversations List  $L$ , the corresponding CEP  $P$  and an interaction function  $f$ , a set of vertices  $V' = \{v_1, \dots, v_n\}$  of a sub-graph  $G' = \langle V', E' \rangle$  of  $G$  model a CCS if  $V' \subseteq V$ ,  $E' = \emptyset$  and  $f(v_1) \cup \dots \cup f(v_n) = f(P)$ .*

**STEP 3.** In the third step, those CCSs forming global communication models with errors (i.e. deadlocks, synchronisation) are located. This is performed by comparing the temporal relationships between the interactions of a global communication model against those specified in interaction protocols of constituent services. The relationships between interactions are found using the reasoning mechanism described in [1], and these are stored in a Relations List.

**STEP 4.** A specification of a global communication model is derived from a CCS. Concise and accurate specifications are required when executing composite services. These specifications are derived from the details in Relations Lists.

## Detecting Deadlocks and Synchronisation Conflicts

This section describes a novel technique that checks whether a valid global communication can be derived from a CCS. A conversation between interactions  $i_a$  and  $i_b$ , or  $i_a$ ,  $i_b$  and  $i_c$  dictates that the temporal relationship(s) between them should be  $= (i_a, i_b)$ <sup>1</sup> for the former case, and  $si(i_a, i_b)$  and  $fi(i_a, i_c)$  for the latter. Our approach takes these temporal relationships that exist between interactions because of conversations and the relationships defined in the descriptions of constituent services, and reasons about the relationships between all the interactions of a CEP using the transitive temporal reasoning mechanism described in [1].

Let us consider three time interval-based interactions  $i_a$ ,  $i_b$  and  $i_c$  with known temporal relationships between  $i_a$  and  $i_b$ , and  $i_b$  and  $i_c$ . The technique in [1] determines the transitive relationship between  $i_a$  and  $i_c$ . For example, if  $s(i_a, i_b) \cup m(i_b, i_c)$  holds, then the transitive relationship between  $i_a$  and  $i_c$  would be either  $oi$ ,  $d$ ,  $f$ ,  $o$  or  $s$ .

A deadlock or a synchronisation conflict occurs in a global communication model if a relationship derived by this mechanism conflicts with another. Let us consider the following CCS to illustrate the use of this approach:  $[cs_2, in_2], [cs_3, in_3, i_5], [cs_3, sh_1, sh_4], [sh_2, in_1][sh_3, in_4]$ . Figure 2 depicts a global communication model derived from this CCS. This model contains a synchronisation conflict because the reasoning mechanism determines that the relationship between  $sh_1$  and  $cs_4$  should be  $b(sh_1, cs_4)$ <sup>2</sup> and the conversation between  $cs_4$ ,  $sh_1$  and  $sh_4$  dictates that  $s(sh_1, cs_4)$ .

Given a CEP  $P$  and a CCS  $C$  derived from  $P$ , VGC derives the temporal relationships between each pair of interactions (say  $i$  and  $i'$ ) based on those that exist between the conversations in  $C$ , where  $i \in p_i$ ,  $p_i \in P$ ,  $i' \in I$  and  $I = \{P \setminus p_i\}$ . These relationships are derived using the proposed transitive temporal reasoning mechanism. While doing so, VGC checks if each derived temporal relationship conflicts with those specified in each  $p_i$  of  $P$ . If they conflict, a global communication model free of deadlocks or synchronisation conflicts cannot be derived from the given CCS.

VGC uses a Relations List to record the temporal relationships and later checks if they conflict. At the start, this list contains specifications of temporal relationships between the interactions of constituent services as well as those that implicitly exist between the interactions that are included in conversations of a CCS. Each relation in this list is specified as an Interval Time Logic (ITL)

<sup>1</sup> The following notations are used to specify the different types of in [1]:  $o$  - overlaps,  $oi$  - overlapped-by,  $s$  - starts,  $si$  - started-by,  $d$  - during,  $di$  - contains,  $f$  - finishes,  $fi$  - finished-by,  $m$  - meets,  $mi$  - met-by,  $b$  - before,  $bi$  - after and  $=$  - equals.

<sup>2</sup>  $b(sh_1, cs_4)$  according to the conversations between  $cs_2$  and  $in_2$  ( $= (cs_2, in_2)$ ) and  $in_1$  and  $sh_2$  ( $= (in_1, sh_2)$ ) and interaction protocols of *Shipping-I* and *ComputerSales-I* ( $b(sh_1, sh_2)$  and  $s(cs_2, cs_4)$ ).

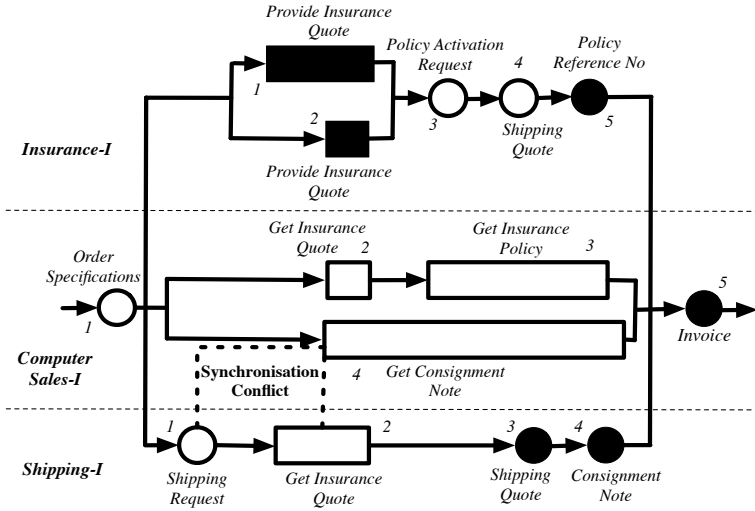


Fig. 2. A Global Communication Model with a Synchronisation Conflict

axiom modelling a relationship between two interactions. VGC then determines all the two-hop transitive temporal relationships that are extracted from the execution paths. Given two interactions, say  $i_a$  and  $i_b$ , a **two-hop transitive temporal relationship** exists between  $i_a$  and  $i_b$  if there is an interaction (say  $i_c$ ), where the relationships between  $i_a$  and  $i_c$ , and  $i_c$  and  $i_b$  are known. Once each of these two-hop transitive temporal relationships are determined, they are compared against those in the Relations List. If the relationships do not conflict, those derived are included in the list. Otherwise, the algorithm will not be able to compute a global communication model with free errors (i.e. - deadlocks or synchronisation conflicts).

### 3 Conclusion

This paper gave details of VGC - an approach to construct valid global communication models for composite services. Existing approaches have limitations, as they either do not consider the temporal aspects of interactions or only consider send and receive interactions. VGC requires interaction protocols to be described with WS- $\pi$ -calculus. This provides an advantage over existing solutions for service description because WS- $\pi$ -calculus accurately models the temporal relationships between interactions. Specifications of global communication models are derived using a four step process. First, conversations are formed by composing interactions of constituent services. Two interactions are composable if the parameters and the channels used to perform them match and they are of compatible types. In the second step, combinations of conversations that form global communication models are located. Then, deadlocks and synchronisation

conflicts are detected by reasoning about the temporal relationships between interactions (of a global communication model). A detailed version of this paper which includes a case study that analyses the correctness and soundness of the proposed approach using a sample scenario is available. Our future work will focus on the analysis of the devised approaches using real-world scenarios.

## Acknowledgments

We would like to thank the ARC (Australian Research Council) for the support given towards this work, under the Linkage Project no. LP0667600 titled “An Integrated Infrastructure for Dynamic and Large Scale Supply Chain”.

## References

1. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* 26(11), 832–843 (1983)
2. Gooneratne, N., Tari, Z.: Matching Independent Global Constraints for Composite Web Services. In: *Proceedings of the 17th International World Wide Web Conference*, pp. 765–774 (2008)
3. Gooneratne, N., Tari, Z., Harland, J.: Verification of Web Service Descriptions using Graph-based Traversal Algorithms. In: *Proceedings of the 22nd Annual Symposium on Applied Computing*, pp. 1385–1392 (2007)
4. Woodman, S., Palmer, D., Shrivastava, S., Wheater, S.: Notations for the Specification and Verification of Composite Web Services. In: *Proceedings of the 8th International Enterprise Distributed Object Computing Conference*, pp. 35–46 (2004)



# A Framework for Advanced Modularization and Data Flow in Workflow Systems

Niels Joncheere, Dirk Deridder, Ragnhild Van Der Straeten,  
and Viviane Jonckers

System and Software Engineering Lab (SSEL)  
Vrije Universiteit Brussel  
Pleinlaan 2, 1050 Brussels, Belgium  
{njonchee, dderidde, rvdstrae, vejoncke}@vub.ac.be

**Abstract.** Workflows have become a popular technique for describing processes in many different application domains, including Computer Aided Engineering (CAE). State-of-the-art workflow languages lack the necessary modularization techniques and data flow capabilities to express processes in a way that facilitates their design, evolution and reuse. In this paper, we aim to tackle this problem by presenting a conceptual framework for advanced modularization and data flow in workflow systems, which is independent of specific modeling approaches and technologies.

## 1 Introduction

Workflows have long been a popular technique for describing processes in a number of application domains, such as business process management and web service orchestration. More recently, they have started to be applied in scientific computing and Computer Aided Engineering (CAE). Workflow languages for each of these application domains have been developed.

As processes become more complex, mechanisms are needed to manage this complexity in order to facilitate the processes' design, evolution and reuse. Traditionally, workflow languages tackle this problem by allowing to decompose workflows into separate modules such as sub-workflows. However, these modules are often strongly tied to the workflow in which they are used, resulting in limited reusability. In addition, most approaches do not support modularizing concerns that crosscut a workflow (such as authentication, transaction management, and logging), and approaches that do [1,2,3] are limited in their expressiveness.

In scientific computing and CAE, the volume, complexity, and heterogeneity of data in processes is much larger than in other application domains. This means that workflows in these domains contain much more data flow: they deal with data transfer from persistent storage to the resources that will process the data, partition data in preparation of parallel processing, and handle transformation of data when different processing steps require different data formats. This *data perspective* [4] is insufficiently supported by current workflow approaches.

The goal of our approach is to improve *separation of concerns* [5] in workflow languages by tackling both the lack of modularization of the main concern and the lack of modularization of *crosscutting concerns* [6] using a single, general workflow construct. More specifically, we revalue the sub-workflow construct as a powerful means for workflow modularization. Based on our collaboration with industrial partners in Computer Aided Engineering, we also aim to provide better support for the data perspective.

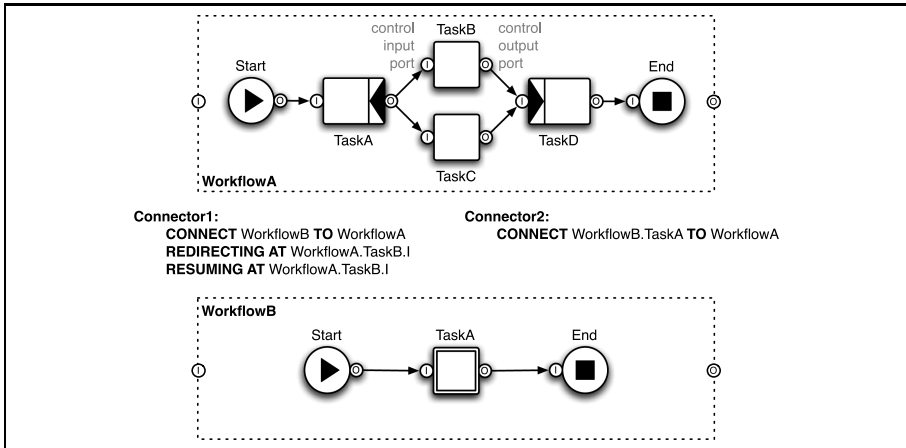
This paper presents our conceptual framework for advanced modularization and data flow in workflow systems. Section 2 describes our modularization mechanism, and Section 3 describes our data flow mechanism. Section 4 presents related work, and Section 5 states our conclusions.

## 2 Modularization Mechanism

Traditionally, sub-workflows are used to decompose the main concern of a workflow into smaller modules, thus facilitating evolution and reuse of these modules. The main workflow contains a *composite task* that specifies — at design time — which sub-workflow should be invoked. When the workflow is enacted, the sub-workflow will be executed. Although this mechanism is a good means for managing workflow complexity, it is not always present in popular workflow languages such as BPEL [7].

While our approach supports this basic mechanism, it improves on it by allowing sub-workflows to be attached to main workflows in a way which inverts the flow of control: it allows specifying at which points in a main workflow a sub-workflow should commence and cease execution, without explicitly specifying this in the main workflow. Thus, we facilitate adding concerns that were not considered when the main workflow was designed, and facilitate comprehending, maintaining, reusing, and removing concerns that are specified using such sub-workflows. For example, this mechanism can be used to invoke an authentication sub-workflow before each invocation of a certain service, even though the workflow that invokes this service was not designed with authentication in mind. This inversion of control is similar to traditional aspect-oriented techniques [8], but unlike aspects, sub-workflows are not separate language constructs introduced solely for the sake of encapsulating crosscutting concerns. Such a *symmetrical* [9,10] approach reduces the number of language constructs, and is expected to facilitate the adoption of our approach in industrial environments.

In order to allow specifying such symmetrical workflow compositions, we introduce the notion of **control ports**. Control ports are the entry and exit points of a task's control flow; each task has exactly one *control input port* and one *control output port*. A workflow's control flow perspective is specified by connecting the control output port of the workflow's start event to the control input port of a task, and connecting the control output ports of all tasks to the control input ports of other tasks or end events. Just like tasks, workflows have exactly one control input and output port; these are the entry and exit points of the workflows' control flow. We visualize control ports by extending the YAWL [11]



**Fig. 1.** Using a connector for symmetrical workflow composition (left) and for connecting a task to a workflow (right)

notation with small circles at the sides of tasks and workflows. Control input ports are marked with the letter I, while control output ports are marked with the letter O. The upper part of Figure 1 shows a workflow called *WorkflowA*, which has a control input port (at its left side) and a control output port (at its right side). Additionally, all elements of the workflow, such as *TaskB*, have control ports as well, which are connected in order to specify the workflow's control flow perspective.

Analogous to the way they are employed in component based software engineering [12] and aspect-oriented programming [10], we introduce **connectors** to connect workflows to each other. In Figure 1, **Connector1** specifies that *WorkflowB* needs to be connected to *WorkflowA*. The connector also specifies that, when *WorkflowA* is enacted, its control flow should be *redirected* to *WorkflowB* when it reaches the control input port of *TaskB*. If control flow should be *split*, one should use the **SPLITTING** keyword instead of the **REDIRECTING** keyword. Additionally, the connector specifies that, when the execution of *WorkflowB* has finished, control flow should resume at the control input port of *TaskB*. The net result of this connector is that *WorkflowB* will be executed *before* *TaskA*.

In our example, control flow is redirected and resumed at the same control port in *WorkflowA*. However, this need not be the case. For example, if the connector would specify that *WorkflowB* should resume at the control output port of *TaskB*, the net result of the connector would be that *WorkflowB* is executed *instead of* *TaskB*. In an initial phase, our system will support all the workflows' control ports as resuming points in order to maximize connector expressivity. However, some resuming points may yield undesirable workflow behavior (such as infinite loops). Therefore, future work will be directed at producing safe connector patterns.

The traditional composite task construct is still available in our approach, but we do not require the composite task to be hard-wired to a concrete workflow

at design time: a connector can be used to wire the composite task to a concrete workflow. This reduces the coupling between a workflow that contains a composite task and the concrete workflows that will implement this composite task. In Figure 11, `Connector2` specifies that `TaskA` in `WorkflowB` should be connected to `WorkflowA`. When `WorkflowB`'s control flow reaches `TaskA`, `WorkflowA` will be executed, and when its execution is finished, the control flow will continue with the remainder of `WorkflowB`. In fact, such a connection can be made even if `TaskA` is a regular task instead of a composite task. In that case, `WorkflowA` would be executed instead of `TaskA`.

Connectors are specified separate from the workflows they connect. This reduces the coupling between sub-workflows and main workflows, and improves the reusability of sub-workflows by making them independent of the context to which they might be applied. This also means that a workflow can assume the role of sub-workflow in one composition, while assuming the role of main workflow in another. Figure 11 illustrates this: in `Connector1`, `WorkflowA` assumes the role of main workflow, and in `Connector2`, `WorkflowB` assumes the role of main workflow. Of course, it would not make sense to have both connectors in the same workflow composition, as this would yield an infinite loop.

### 3 Data Perspective

Most state-of-the-art workflow languages are not designed with the data perspective in mind. Data is typically accessible by groups of tasks using some basic scoping mechanism, or is simply passed along with the control flow. The data and control flow perspectives are tightly integrated, and their independence is concealed. The concepts modeled in these perspectives, as well as their independence get blurred and distorted. Existing workflow research [13] has recognized the need to uphold this independence. Therefore, we improve on existing languages by specifying data flow and control flow separately.

A first concept we introduce to this end is **data ports**. Each task can have an arbitrary number of *data input ports* and *data output ports*, which represent the input and output parameters of a task, respectively. Each of a workflow's ports has a unique name, and specifies the types of data transfer that it supports, which are either pass-by-value, pass-by-reference, or streaming. Depending on the application domain, a data port can specify the type of its data (for example using XML Schema). Analogous to tasks, workflows have an arbitrary number of data input and output ports as well.

Secondly, we introduce a first-class **data flow** construct, which is visualized as a special kind of arrow that connects the data output port of one task to the data input port of another. The basic case of the construct specifies *data transfer* between two tasks. The data flow specifies the type of transfer that will actually be used (pass-by-value, pass-by-reference, or streaming — which should be compatible with the connected data ports), and depending on this type, specifies where data should be stored intermediately. For example, the data flow can specify that a data output port's data should be passed by reference to a

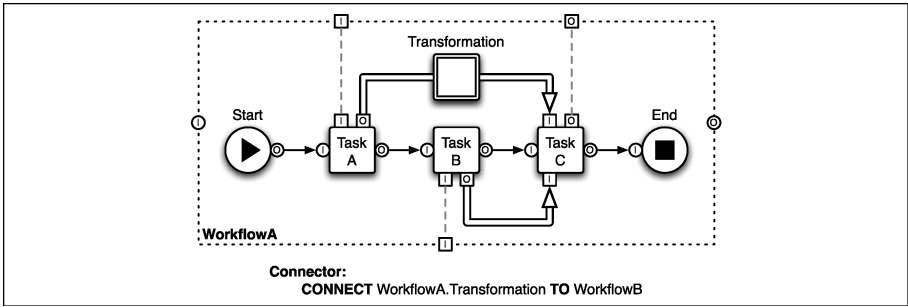


Fig. 2. Data transfer and transformation

certain data input port, while storing the actual data in a certain database. A more advanced case of the data flow construct specifies *data transformation*: instead of simply transferring data, the user can specify the way in which data should be transformed during its transfer. Using these two cases of our first-class data flow construct, one can express all data transfer patterns defined in [4].

The upper part of Figure 2 provides an example using a workflow named **WorkflowA**. Data ports are visualized by extending the YAWL notation with small rectangles at the sides of tasks and workflows. Data input ports are marked with the letter I, while data output ports are marked with the letter O. **WorkflowA** contains a data transformation named **Transformation** between the data output port of **TaskA** and one of the data input ports of **TaskC**, and an anonymous data transfer between the data output port of **TaskB** and the other data input port of **TaskC**. The data flows are visualized by extending the YAWL notation with thick arrows; the transformation is differentiated from the transfer by the square in the middle of its arrow.

By default, the workflow’s tasks’ unconnected data ports are implicitly exposed as the workflow’s data ports. This is shown in the example by the dashed lines, which are not part of the notation. If a more advanced mapping between the workflow’s tasks’ data ports and the workflow’s data ports is desired, it can be specified in the workflow, but this is not visualized using the notation.

The data transformation construct greatly simplifies CAE workflows, as these typically contain a large amount of data transformation logic. Depending on the application domain, a specific engine for our language might support a number of built-in transformation strategies, such as XSLT transformations for XML data, but in general, the user can specify data transformation strategies by using the workflow language itself: because data transformations are first-class, data transformations can be composites, and can be connected to a workflow using a connector. Figure 2 illustrates this scenario by showing a connector that links **Transformation** to a workflow named **WorkflowB**, which is not shown. **WorkflowB** can be any workflow that has exactly one data input port and one data output port. When **Transformation** is executed, its incoming data will be sent to **WorkflowB**’s data input port, and its outgoing data will be retrieved from **WorkflowB**’s data output port.

## 4 Related Work

The Abstract Grid Workflow Language (AGWL) [14] is an interesting approach which also identifies the need for an powerful, separate data perspective. However, the approach does not address the requirement of separation of concerns.

Data flow languages [15] have long been available as a paradigm for expressing computations according to its data perspective. However, existing workflow research [13,14] has shown that having only a single workflow perspective is insufficient. Nevertheless, the data perspective of our approach can be seen as a coarse grained data flow language where tasks are the macro actors.

In the modeling community, UML activity diagrams [16] have been developed as a means of expressing workflows. Data flow can be modeled separate from control flow using pins, and there is a distinction between streaming and non-streaming data transfer. However, the approach does not consider separation of concerns or the specific needs of data-intensive applications.

## 5 Conclusions

In this paper, we propose a conceptual framework for advanced modularization and data flow in workflow systems. We describe a workflow language which introduces four language elements: control ports, data ports, data flow, and connectors. A workflow's data flow is specified separate from its control flow by connecting tasks' data ports using a first-class data flow construct. Connectors allow expressing that a task in one workflow should be executed by another workflow, in a way that minimizes dependencies between these workflows and thus facilitates their independent evolution and reuse. Additionally, connectors allow connecting data flow constructs to workflows. The inversion-of-control connector allows augmenting a workflow with concerns that were not considered when it was designed, and again facilitates independent evolution and reuse of these workflows.

Many of the concepts we introduce are not present in current workflow approaches. In particular, our inversion-of-control mechanism is significantly more powerful than existing aspect-oriented approaches for workflows [1,2,3]. Future work will be directed at providing a proof-of-concept implementation for our approach.

## Acknowledgements

The research presented in this paper is funded by the Research Foundation – Flanders through the DyBroWS project.

## References

1. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In: Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 168–182. Springer, Heidelberg (2004)

2. Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), St. Louis, MO, USA. ACM Press, New York (2005)
3. Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W., Jonckers, V.: Isolating process-level concerns using Padus. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 113–128. Springer, Heidelberg (2006)
4. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns. QUT Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia (2004)
5. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 15(12), 1053–1058 (1972)
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
7. Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services: Version 1.1 (2003), <http://www.ibm.com/developerworks/library/specification/ws-bpel/>
8. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 327–354. Springer, Heidelberg (2001)
9. Tarr, P., Ossher, H., Harrison, W., Stanley, M., Sutton, J.: N degrees of separation: Multi-dimensional separation of concerns. In: Proceedings of the 21st International Conference on Software Engineering (ICSE 1999), Los Angeles, CA, USA, pp. 107–119. IEEE Computer Society, Los Alamitos (1999)
10. Suvée, D., De Fraine, B., Vanderperren, W.: A symmetric and unified approach towards combining aspect-oriented and component-based software development. In: Gorton, I., Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Szyper-ski, C., Wallnau, K. (eds.) CBSE 2006. LNCS, vol. 4063, pp. 114–122. Springer, Heidelberg (2006)
11. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. *Information Systems* 30(4), 245–275 (2005)
12. Shaw, M., Garlan, D.: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River (1996)
13. Axenath, B., Kindler, E., Rubin, V.: AMFIBIA: A meta-model for the integration of business process modelling aspects. *International Journal of Business Process Integration and Management* 2(2), 120–131 (2007)
14. Fahringer, T., Pillana, S., Villazon, A.: AGWL: Abstract Grid Workflow Language. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3038, pp. 42–49. Springer, Heidelberg (2004)
15. Johnston, W.M., Hanna, J.R.P., Millar, R.J.: Advances in dataflow programming languages. *ACM Computing Surveys* 36(1), 1–34 (2004)
16. Object Management Group: UML superstructure, version 2.1.2 (2007), <http://www.omg.org/spec/UML/2.1.2/>

# Model Identification for Energy-Aware Management of Web Service Systems

Mara Tanelli<sup>1,2</sup>, Danilo Ardagna<sup>1</sup>, Marco Lovera<sup>1</sup>, and Li Zhang<sup>3,\*</sup>

<sup>1</sup> Dipartimento di Elettronica e Informazione,  
Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
{tanelli, ardagna, lovera}@elet.polimi.it

<sup>2</sup> Dipartimento di Ingegneria dell'Informazione e Metodi Matematici,  
Università degli studi di Bergamo, Via Marconi 5, 24044, Dalmine (BG), Italy

<sup>3</sup> IBM Research, T.J. Watson Research Center, Yorktown Heights, NY 10598  
zhangli@us.ibm.com

**Abstract.** In SOA environments, service providers need to comply with the service level objectives stipulated in contracts with their customers while minimizing the operating costs of the physical infrastructure, mainly related to energy costs. The problem can be effectively formalized by using system identification and control theory: the service levels are translated into set-points for the response times of the hosted applications, and performance are traded-off with energy saving objectives based on suitable models for server dynamics. As the behavior of the incoming workload changes significantly within a single business day, control-oriented system identification approaches are very promising to model such systems, especially at a very fine grained time scales and in transient conditions. In this paper Linear Parameter Varying (LPV) state space system identification algorithms are analyzed for modeling Web services systems. The suitability of LPV models is investigated and their performance assessed by experimental data.

## 1 Introduction

Energy management is rapidly becoming a priority in the design and operation of complex service-based information systems, as the impact of energy consumption associated with IT infrastructures increases. The growth in the number of servers and the increasing complexity of the network infrastructure have caused an enormous spike in electricity usage. IT analysts predict that, by 2012, up to 40% of IT budget will be consumed by energy costs, [12]. This trend is striving green computing activities in the industry research agenda (see for example IBM's project *Big Green* [8] and HP's *Green up* initiative [7]).

In the context of Web services and SOA based systems, service centers need to comply also to the Service Level Agreements (SLAs) stipulated with their customers. At run

---

\* The work of M. Tanelli and M. Lovera has been partially supported by MIUR project "New methods for Identification and Adaptive Control for Industrial Systems." The work of D. Ardagna has been partially supported by the EU FP7 Q-ImPrESS project.



time, service requestors address their invocation to the most suitable provider according to their Quality of Service (QoS) preferences. QoS requirements are difficult to satisfy because of the high variability of Internet workloads. It is difficult to estimate workload requirements in advance, as they may vary by several orders of magnitude within the same business day, [6]. To handle workload variations, many service centers employ autonomic techniques [4,18] such that resources are dynamically allocated among running Web services based on short-term demand estimates. The goal is to meet the application requirements while adapting the IT system. This leads to the study of how to efficiently use resources and reduce energy consumption.

Early autonomic techniques switched servers on and off based on the service center workload, [4]. More recent proposals, see e.g. [13,10], have started reducing the frequency of operation of servers by exploiting the Dynamic Voltage Scaling (DVS) mechanisms implemented in new servers. DVS varies both CPU supply voltage and operating frequency. The adoption of DVS is very promising, as power consumption is proportional to the cube of the operating frequency, while servers performance varies linearly with the operating frequency. Furthermore, DVS does not introduce any system overhead (vice versa, hibernating and restoring a server require time and energy).

Several research contributions have proposed autonomic self-managing techniques and can be classified mainly in two categories: (i) utility-based optimization techniques, and (ii) feedback control-theoretic approaches. Utility-based approaches have been introduced to optimize the degree of user satisfaction by expressing their goals in terms of user-level performance metrics. Typically, the system is described by means of a performance model based on queueing theory, embedded within an optimization framework. Utility based approaches can handle multiple decision variables (e.g., admission control, application placement, load balancing, etc.) but are based on the assumption that the system is at *steady state*. Hence, these techniques are effective on a medium term control time horizon, e.g., half an hour, [4], [13]. Vice versa, genuine control-theoretic approaches can accurately model system transients and can adjust the system configuration within a very short time frame. Thus, control-theoretic approaches are effective over fine grained time horizons, e.g., minutes and, furthermore, can effectively employ DVS as control variable and formally guarantee both closed-loop stability and performance specifications.

In this paper, the adoption of Linear Parametrically Varying (LPV) models for the performance control of Web services will be addressed. A LPV model is linear in the parameters and a vector of scheduling variables enters the system matrices in an affine or linear fractional way ([11,19,16]). Such a representation for general nonlinear systems can be useful in view of control design using modern robust control and gain-scheduling control techniques [3]. Models are identified from experimental data measured on a micro-benchmarking Web service application, adopting DVS of CPUs as control variable.

The structure of the paper is as follows. Section 2 provides a review of the literature, while Section 3 formally states the problem addressed in this paper and introduces the needed notation. Section 4 briefly describes discrete time state space dynamical models and illustrates the LPV models adopted in this work. Experimental results are presented in Section 5. Conclusions are finally drawn in Section 6.

## 2 Related Work

Autonomic management of service center infrastructure is receiving great interest by the control theory research community. The first control-oriented contributions applied to the management of Web services are reported in [115], and use feedback control to limit the utilization of bottleneck resources by means of admission control and resource allocation. In the practice of control engineering, when a single control system must be designed to guarantee closed-loop operation of a given plant in many different operating conditions, two broad classes of methods are available: gain scheduling and adaptive control.

The gain scheduling approach to the problem can be summarised as follows: find one or more *scheduling variables* which can completely parameterise the operating space of interest for the system to be controlled; define a parametric *family* of linearised models for the plant associated with the set of operating points of interest; finally, design a *parametric* controller which can both ensure the desired control objectives in each operating point and an acceptable behaviour during (slow) transients between one operating condition and the other. A wide body of design techniques is now available for this problem (see, e.g., [3]), which can be reliably solved provided that a suitable model in parameter-dependent form has been derived. This modelling problem, however, raises a number of significant issues. While the literature on non-linear identification can now provide advanced tools for the estimation of a wide variety of model classes, in such a case it would be useful to separate conventional input variables from scheduling variables (i.e., variables defining the operating point of the plant), by letting them enter the model in distinct ways. LPV models have been recently proposed as a way of dealing with this kind of problems and have been adopted recently in [13] to implement an autonomic controller able to provide performance guarantees by means of DVS.

With respect to that approach, where input/output (I/O) LPV identification was considered, the method adopted here is more appropriate to provide system models tailored to LPV control design, as they are directly identified in state space form and avoid all the issues - not addressed in [13] - related with equivalence notions between I/O and state space LPV realizations, [17]. Furthermore, state space LPV identification allows a straightforward extension to the multiple input, multiple output case, which is needed if more than one class of customers needs to be taken into account.

Control theoretic approaches are suitable to model Web systems both in stationary and transient conditions. In the queueing theory literature, some recent proposals address the problem of modelling queue network transient behaviour by means of Markov models in order to study burstiness and long range dependency in system workloads [5][14]. The main limitation of Markovian models is their complexity, which makes, even for very simple systems, e.g., a first come first served (FCFS) single server queue, the number of parameters to be determined quite large. These models suffer for high computation overhead and, hence, are presently not suitable for the implementation of real-time controllers.

## 3 Problem Statement

In this paper, a CPU bounded Web service application will be considered where, without loss of generality, the resource scheduler implements a FCFS policy (LPV models can

consider also other scheduling policies, e.g., processor sharing or generalized processor sharing). In the remainder of the paper the following notation will be adopted:

- $\Delta t$ : sampling interval;
- $k$ : discrete time index over the interval  $[k\Delta t, (k+1)\Delta t]$ ;
- $\lambda_k$ : requests arrival rate at the server in the  $k$ -th interval;
- $s_k$ : requests service time, i.e., overall CPU time needed to process a request in the  $k$ -th interval;
- $R_k$ : average server response time in the  $k$ -th interval, i.e., the overall time a request stays in the system.

We assume that  $s_k$  is inversely proportional to the physical server CPU frequency. As such, when physical servers are endowed with DVS capabilities, the effect of - say - lowering the CPU frequency when a light workload is present in the system causes an increase of the effective CPU time needed to serve a request [10]. This assumption is supported by current technology trends, since in modern systems (e.g., AMD Operon 2347HE Barcelona core) CPUs and RAM clock can be scaled independently (otherwise, RAM access could become a bottleneck and CPUs could stall for memory accesses). If we denote with  $f_k$  the ratio of the frequency applied during the time interval  $k$  to the physical server maximum frequency, the *effective* service time can be defined as  $s_{f,k} = s_k / f_k$ .

The goal of this paper is to derive a dynamic model of an application server capable of capturing system behaviour at a very fine-grained time resolution (seconds), with an accuracy suitable for control purposes. This identification process provides a control-oriented dynamical description of the server behavior and it is the first step to be achieved in order to design a closed-loop controller for service center infrastructures able to meet SLAs requirements while minimizing energy costs. The design of the closed-loop controller is the focus of our ongoing work. The adoption of control oriented techniques is motivated since the workload of SOA systems is characterized by highly varying conditions [18]. The LPV framework is adopted since it seems very promising for modeling such systems. Furthermore once the modelling and control problem in the LPV framework is solved, the closed-loop system will not require to be complemented with workload predictors, whose design is hard to carry out, as the best models have been shown to require nonlinear and non-stationary workload description [2]. In fact, the workload variability is embedded in the LPV system representation, which tunes on-line both the model and the control action taking into explicit account the current workload condition.

## 4 Identification of Discrete-Time State Space Models

The problem of model identification can be formulated as the one of deriving a mathematical representation for the behaviour of a physical system on the basis of input-output data collected in dedicated experiments. As far as linear models are concerned, the main “ingredients” of an identification problem are essentially: 1) a definition of the class of models to be considered and 2) a suitable algorithm for the estimation of the model parameters on the basis of the available data. Classical model identification problems for Single-Input Single-Output time-invariant systems are formulated using models in input-output form (i.e., difference equations relating the measured input and

output variables in a direct way), the parameters of which are estimated using least squares techniques. Whenever Multiple-Input Multiple-Output and/or time-varying systems must be dealt with, state-space representations turn out to be a more flexible and reliable model class. In this work discrete-time linear state-space models will be considered, in the *innovation form*:

$$\begin{aligned}x_{k+1} &= A_k x_k + B_k u_k + K_k e_k \\y_k &= C_k x_k + D_k u_k + e_k,\end{aligned}\tag{1}$$

where  $x \in \mathbb{R}^n$  is the state vector,  $u \in \mathbb{R}^m$  is the vector of control inputs and  $y \in \mathbb{R}^l$  is the vector of measured outputs and  $e$  is a white process noise. More precisely, with reference to the specific modelling problem considered in this study,  $u_k = s_{f,k}$  and  $y_k = R_k$ , i.e., the goal of the model identification problem considered in this paper is to derive a state-space model describing the dynamic relationship between the effective service time and the server response time. In [11] generically time-varying state space matrices  $\{A_k, B_k, C_k, D_k\}$  have been considered. In what follows we will introduce different, additional assumptions on the time-variability of the model dynamics, suitably tailored for the model identification problems associated with the management of Web services.

More precisely, the generic time-variability will be restricted to the LPV class. LPV systems are linear time-varying plants whose state space matrices are fixed functions of some vector of varying parameters. LPV model identification algorithms are available in the literature both for input-output and state-space representations of parametrically-varying dynamics. In this work state-space LPV models will be adopted:

$$\begin{aligned}x_{k+1} &= A(\delta_k)x_k + B(\delta_k)u_k \\y_k &= C(\delta_k)x_k + D(\delta_k)u_k,\end{aligned}\tag{2}$$

where  $\delta \in \mathbb{R}^s$  is the parameter vector. For the sake of simplicity, in the following we will deliberately focus on purely deterministic models, i.e., we will ignore the presence of the noise terms in the model representation. It is important to point out, however, that the theory underlying the parameter estimation algorithms used in this work can effectively deal with the presence of process and measurement noise [19]. It is often necessary to introduce additional assumptions regarding the way in which  $\delta_k$  enters the system matrices. The most common are the following:

1. Affine parameter dependence (LPV-A), where  $A(\delta_k) = A_0 + A_1\delta_{1,k} + \dots + A_s\delta_{s,k}$  and similarly for  $B, C$  and  $D$ .  $\delta_{i,k}, i = 1, \dots, s$  denotes the  $i$ -th component of vector  $\delta_k$ . This form can be immediately generalised to polynomial parameter dependence.
2. Input-affine parameter dependence (LPV-IA): this is a particular case of the LPV-A parameter dependence in which only the  $B$  and  $D$  matrices are considered as parametrically-varying, while  $A$  and  $C$  are assumed to be constant:  $A = A_0, C = C_0$ .

Identifying LPV models in general state space form is a difficult task. It is usually convenient to consider first the simplest form, i.e., the LPV-IA one, as its parameters can be retrieved by using subspace model identification (SMI) algorithms for linear time invariant systems (which are significantly easier to use and available in commercial

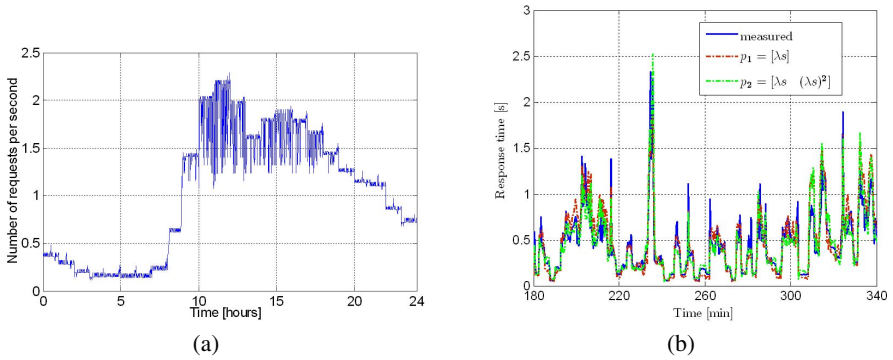
software packages) by suitably extending the input vector. In this work the MOESP class of SMI algorithms [20] has been considered. The classical way to perform linear system identification is by minimizing the error between the real output and the predicted output of the model. A similar approach can be used for LPV state-space systems of the form (2). Letting the system matrices of (2) be completely described by a set of parameters  $\theta$ , identification can be carried out by minimizing the cost function  $V_N(\theta) := \sum_{k=1}^N \|y_k - \hat{y}_k(\theta)\|_2^2 = E_N^T(\theta)E_N(\theta)$  with respect to  $\theta$ , where  $E_N^T(\theta) = \left[ \left( y_1 - \hat{y}_1(\theta) \right)^T \cdots \left( y_N - \hat{y}_N(\theta) \right)^T \right]$ , while  $y_k$  denotes the measured output and  $\hat{y}_k(\theta)$  denotes the output of the LPV model to be identified. In general, the minimization of  $V_N(\theta)$  is a nonlinear, nonconvex optimization problem. Many algorithms have been proposed, in this work the gradient search method based on the Levenberg-Marquardt algorithm has been adopted [11].

## 5 Experimental Results

In the experimental framework, a workload generator and a micro-benchmarking Web service application have been used. The workload generator is based on a custom extension of the Apache *JMeter* 2.3.1 workload injector, which allows to generate workload according to an open model [9] with a Poisson arrival process. The Web service is a Java servlet designed to consume a fixed amount of CPU time generated according to deterministic (for identification purposes), Poisson, Pareto and log-normal distributions (for validation). The adoption of a micro benchmarking application allows the validation of the effectiveness of our approach both for workload intensive and for computationally intensive applications. Furthermore, the CPU time standard deviation of the micro benchmarking application has been varied in order to verify if LPV models performance depends on the variability of the CPU time distribution: the standard deviation  $\sigma[s]$  has been chosen as  $q$  times the average of the service time distribution  $E[s]$ , i.e.,  $\sigma[s] = q E[s]$ , where  $q$  was set equal to 2, 4 and 6. For model validation, the incoming workload reproduces a 24 hour trace obtained from a large Internet Web site. The log was collected on a hourly basis. The workload injector is configured to follow a Poisson process with request rate changing every minute where the requests rate is obtained from the log trace superimposed with a Gaussian noise proportional to the workload intensity, as in [10].

To quantitatively evaluate the models, two metrics have been considered: the percentage Variance Accounted For (VAF), defined as  $VAF = \left( 1 - \frac{Var[y_k - \hat{y}_k(\theta)]}{Var[y(k)]} \right)$ , where  $y_k$  is the measured signal (i.e., application response time), and  $\hat{y}_k(\theta)$  is the output obtained from the simulation of the identified model, and the percentage average simulation error  $e_{avg}$ , computed as  $e_{avg} = \left( \frac{E[\|y_k - \hat{y}_k(\theta)\|]}{E[\|y_k\|]} \right)$ .

The identification data were processed to extract the average values over a sampling interval  $\Delta t = 10s$  and two LPV second order models, one with  $p_1 = \lambda s$  and the other with  $p_2 = [\lambda s \quad (\lambda s)^2]$  have been identified (see Figure 1(b) for a plot of a detail of the results). The plot shows that the models are capable of providing a response time which correctly follows the peaks of the measured one. Results reported in Table 1 also show that the performance of LPV models are almost independent on the value of  $q$ , i.e., the models are robust to the variability of the service time distribution.



**Fig. 1.** (a) Time history of the request rate applied during a validation test; (b) Detail of the measured (solid line) and the response time obtained with  $\Delta t = 10$  s an LPV model with  $p_1 = \lambda s$  (dashed line) and  $p_2 = [\lambda s (\lambda s)^2]$  (dash-dotted line) on identification data in with  $q = 4$

**Table 1.** Performance of the identified models with  $\Delta t = 10$ s on validation data

Valid. Performance - LPV	q=2		q=4		q=6	
	$(p_1)$	$(p_2)$	$(p_1)$	$(p_2)$	$(p_1)$	$(p_2)$
$\Delta t = 10$ s						
VAF	58.31%	74.14%	54.01%	71.50%	58.85%	74.52%
$e_{avg}$	25.70%	18.36%	20.30%	7.40%	31.87%	31.67%

## 6 Concluding Remarks and Future Work

This paper presents the results obtained in the application of LPV model identification techniques for the performance control of Web services. Specifically, the suitability of subspace LPV methods has been checked against experimental data measured on a custom implementation of a workload generator and a micro-benchmarking Web service application. Future work will develop along two directions: on the modelling side, we aim to further validate our approach on real applications and to extend the models considering a multi-class framework in virtualized environments, whereas on the control design side, work will be devoted to devise both admission control policies and response time regulation in the LPV framework.

## References

1. Abdelzaher, T., Shin, K.G., Bhatti, N.: Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE Trans. on Parallel and Distributed Systems* 15(2) (March 2002)
2. Andreolini, M., Casolari, S.: Load prediction models in web-based systems. In: Proc. of the 1st international conference on Perf. evaluation methodologies and tools, Pisa, Italy (2006)
3. Apkarian, P., Adams, R.J.: Advanced Gain-Scheduling Techniques for Uncertain Systems. *IEEE Trans. on Control System Technology* 6, 21–32 (1998)
4. Ardagna, D., Trubian, M., Zhang, L.: SLA based resource allocation policies in autonomic environments. *Journal of Parallel and Distributed Computing* 67(3), 259–270 (2007)

5. Casale, G., Mi, N., Smirni, E.: Bound Analysis of Closed Queueing Networks with Workload Burstiness. In: Proc. of SIGMETRICS (2008)
6. Chase, J.S., Anderson, D.C.: Managing Energy and Server Resources in Hosting Centers. In: ACM Symposium on Operating Systems principles (2001)
7. HP. Green up initiative,  
<http://www.hp.com/hpinfo/newsroom/feature-stories/2007/07-360-greenup.html>
8. IBM. Project Big Green,  
<http://www-03.ibm.com/press/us/en/photo/21514.wss>
9. Kleinrock, L.: Queueing Systems. John Wiley and Sons, Chichester (1975)
10. Kusic, D., Kandasamy, N.: Risk-Aware Limited Lookahead Control for Dynamic Resource Provisioning in Enterprise Computing Systems. In: ICSOC 2006 Proc. (2006)
11. Lee, L.H., Poolla, K.: Identification of linear parameter-varying systems using nonlinear programming. ASME J. of Dynamic Systems, Measurement and Control 121(1), 71–78 (1999)
12. Metha, V.: A Holistic Solution to the IT Energy Crisis (2007),  
<http://greenercomputing.com/>
13. Qin, W., Wang, Q.: Modeling and control design for performance management of web servers via an LPV approach. IEEE Trans. on Control Systems Tech. 15(2), 259–275 (2007)
14. Riska, A., Squillante, M., Yu, S.Z., Liu, Z., Zhang, L.: Matrix-Analytic Analysis of a MAP/PH/1 Queue Fitted to Web Server Data. In: Latouche, G., Taylor, P. (eds.) Matrix-Analytic Methods: Theory and Applications, pp. 335–356. World Scientific, Singapore (2002)
15. Robertsson, A., Wittenmark, B., Kihl, M., Andersson, M.: Admission control for web server systems - design and experimental evaluation. In: 43rd IEEE Conference on Decision and Control (2004)
16. Tanelli, M., Ardagna, D., Lovera, M.: LPV model identification for power management of web service systems. In: 2008 IEEE Multi-conference on Systems and Control, San Antonio, USA (2008)
17. Toth, R., Felici, F., Heuberger, P.S.C., Van den Hof, P.M.J.: Discrete time LPV I/O and state space representations, differences of behavior and pitfalls of interpolation. In: Proc. of the 2007 European Control Conference, Kos, Greece (2007)
18. Urgaonkar, B., Pacifici, G., Shenoy, P.J., Spreitzer, M., Tantawi, A.N.: Analytic modeling of multitier Internet applications. ACM Transaction on Web I(1) (January 2007)
19. Verdult, V.: Nonlinear System Identification: A State-Space Approach. PhD thesis, University of Twente, Faculty of Applied Physics, Enschede, The Netherlands (2002)
20. Verhaegen, M.: Identification of the deterministic part of MIMO state space models given in innovations form from input output data. Automatica 30, 61–74 (1994)

# LASS – License Aware Service Selection: Methodology and Framework


G.R. Gangadharan<sup>1</sup>, Marco Comerio<sup>2</sup>, Hong-Linh Truong<sup>3</sup>,  
Vincenzo D’Andrea<sup>4</sup>, Flavio De Paoli<sup>2</sup>, and Shahram Dustdar<sup>3</sup>

<sup>1</sup> Telematica Institute, Enschede, The Netherlands  
gr@telin.nl


<sup>2</sup> University of Milano - Bicocca, Milano, Italy  
{comerio, depaoli}@disco.unimib.it

<sup>3</sup> Vienna University of Technology, Vienna, Austria  
{truong, dustdar}@infosys.tuwien.ac.at

<sup>4</sup> University of Trento, Trento, Italy  
dandrea@disi.unitn.it

**Abstract.** A service provider defines individual services with corresponding service licenses which consumers should follow. Often, service consumers are interested in selecting a service based on certain licensing clauses. For a set of requested licensing clauses by a consumer, there can be several licenses that differ in the set of offered license specifications. Thus, a license aware service selection process includes the discovery of a set of services that meets certain functional parameters and, in addition, the process evaluates these services in order to identify the ones that fulfill a set of license specifications as requested by a consumer. In this paper, we present a methodology and framework for service selection process, based on matching the offered licensing specifications by providers with the requested licensing specifications by consumers .

## 1 Introduction

Selecting different services to fulfill a consumer’s need is a fundamental issue that has attracted much research efforts. Most existing works concentrate on developing selection techniques based on functional properties (FPs) and non-functional properties (NFPs). However, the selection of a service usage is also dependent on other clauses, such as scope of rights and warranties, that are important in deciding whether a service should be used. One of the relevant issues from this perspective is the role of service licensing in service selection. A service license includes all transactions between the licensor and the licensee, in which the licensor agrees to grant the licensee the right to use and access the service under predefined terms and conditions. Various aspects of service licensing are described in .

---

<sup>1</sup> This work is partially supported by the IST COMPAS project, funded by the European Commission, FP7-ICT-2007-1 contract number 215175.



We propose to extend the traditional FPs and NFPs based service selection process with an additional selection based on licenses. In our previous work on license-aware selection [2], we discussed steps in license-aware service selection and introduced the ranking of services by applying degree and distance indicators to scope of rights and financial terms in service licenses. This paper proposes an extension of the work discussed in [2] by introducing an approach for enhancing service selection with a novel process based on matching the offered licensing specifications with the requested licensing specifications that takes into account not only *Scope of Rights* and *Financial Terms* but also *Warranty*, *Indemnity*, *Limitation of Liability* and *Evolution* clauses. We also introduce in detail the steps in license-aware service selection, including selection algorithm, and present a framework realizing the service selection process.

## 2 License Aware Service Selection (LASS) Methodology

The LASS methodology includes the phases of *Matching Evaluation* and *Filtering*. The *Matching Evaluation* phase computes two different indicators (*Degree* and *Distance*) between the clauses of offered license specifications and the clauses of requested license specifications. The *Degree* shows if an offer matches a request and is expressed by a value in the range  $[0..1]$ , where 0 means ‘no match’ and 1 means ‘exact match’. The *Distance* indicator is used to capture additional information about the matching. In the case of ‘exact match’, it points out how much the offer dominates the request ( $Distance \geq 0$ ) and in the case of ‘no match’, how much the offer is far from satisfying the request ( $Distance \leq 0$ ).

Evaluating the values for license specifications involves the following kinds of data: (i) *Scope of Rights* clauses expressed as distinct values (e.g., *adaptation*, *composition*, and *derivation*); (ii) *Financial Terms* and *Warranty* clauses expressed in a range of values; (iii) *Indemnity* and *Limitation of Liability* clauses expressed as a set of qualitative values; (iv) *Evolution* clauses expressed as an integer values.

The evaluation of *Degree* and *Distance* indicators for *Scope of Rights* and *Financial Terms* is explained in [2].

*Warranty* clauses can be expressed by consumers using the constraint operators  $\geq$  or  $\leq$  (for example, amount  $\leq 2$  Euros) or an interval of values (for example,  $97\% \leq$  availability rate  $\leq 99\%$ ). The service provider usually publishes a service license with the specifications of the offered *Warranties* expressed as an interval of values. The *Degree* and *Distance* indicators are evaluated according to the constraint operators of the requested license specifications. Examples of these formulas are discussed in [3].

A service license also specifies *Indemnity* and *Limitation of liability* clauses. *Indemnity* clauses specify the provision of defense by the licensor to the licensee. An example is the *Third Party Infringements Claims* clause that represents the statement provided by the licensor to the licensee to protect against the claims of a third party if any infringements over the intellectual property rights arise. *Limitation of liability* clauses limit the liability of the licensor and the

licensee under the license agreement. An example is the *Non-Network Errors* clause specifying that the licensor will not be liable if any problem with the network occur during the service provisioning. For *Indemnity* and *Limitation of liability* clauses the *Degree* is set to 1 if the requested clause matches exactly with the offered one and it is set to 0 elsewhere. These clauses differ from the ones on *Scope of Rights* because subsumption cannot be used. For this reason, the *Distance* is always set to 0.

The last kind of data that our methodology is able to manage is *Evolution* clauses that specify the modifications on the license by future releases or versions of the service. Examples are the *Maximum Upgrades* and the *Maximum Versions* clauses that indicate the allowed number of upgrades and versions of the service before the license becomes invalid. As these clauses are expressed as a fixed integer number, the *Degree* is equal to 1 if the requested value is  $\leq$  than the offered value and it is set to 0 elsewhere. The *Distance* indicator is evaluated subtracting the requested value from the offered value.

The second phase of the LASS methodology, consists of three activities. The first activity is in charge of discarding unsuitable services and starts considering the *Degree* evaluated for each license clause. A service whose license has a clause with *Degree* equal to 0 is discarded and no longer considered in the selection process. Thus, the number of candidate services in the set of functionally matched services can be reduced. If all services are discarded (no services are able to satisfy the requested license specifications), a service with a license closer to a consumer's requested specifications will be recommended. The service recommended to a consumer is selected based on the *Distance* indicator. The consumer can accept or deny the recommended service (as not exactly satisfying the requested license specifications). The second activity of the *Filtering* phase is in charge of evaluating net indicators for each license, which is a distinguishing characteristic of our approach. The *Net Degree* provides information about how much an offered license matches a requested license. The *Net Distance* provides additional information about how the required clauses are matched. Details about how to calculate *Net Degree* and *Net Distance* are described in [2].

Finally, in the *Filtering* phase the list of services is sorted according to their *Net Degree* values. If two or more services have equal *Net Degree* value, their *Net Distance* values are considered for ranking.

Service license selection algorithm is listed in Algorithm 1. The inputs provided by a consumer are a set of requested functionalities  $F$  and the requested license clauses  $lc$ .

Let  $F = \{f_1, f_2, \dots, f_n\}$  denotes a set of functional parameters. Functional parameters, specified by consumers, represent the requested operations performed by services. For each  $f_i$ , we assume that there exists a category of services,  $t_i$ , that offers the functionality specified by  $f_i$ .

Let  $T = \{t_1, t_2, \dots, t_n\}$  denotes categories of services associated to  $F$ , where  $t_i$  provides the functionality required by  $f_i$ . Given a  $t_i \in T$ , there exists many services belonging to this service type, each offering the functionality  $f_i$  but with

possibly different implementations and associated licenses. We denote this set of services with  $S(t_i) = \{S_1, S_2, \dots, S_m\}$ .

Let  $\Lambda(S(t_i)) = \{L(S_1), L(S_2), \dots, L(S_m)\}$  be the set of licenses in which  $L(S_i)$  indicates the license associated with the service  $S_i$ . Let  $lc$  be the requested license specifications that need to be considered along the selection process. Let  $\Upsilon(t_i)$  be a set of filtered services having the requested license clauses that match with the offered license specifications. Our objective is to select a service license in  $\Lambda(S(t_i))$  that best matches  $lc$ .

---

**Algorithm 1.** Service License Selection
 

---

```

1: for all  $t_i \in T$  do
2:   for all  $S_j \in S(t_i)$  do
3:      $Degree(L(S_j)) \leftarrow DgEval(lc, L(S_j))$ 
4:      $Distance(L(S_j)) \leftarrow DsEval(lc, L(S_j))$ 
5:   end for
6:    $\Upsilon(t_i) \leftarrow Filter(S(t_i))$ 
7:   Recommend a service closer to  $lc$  if  $\Upsilon(t_i) = \phi$ 
8:    $\Psi(S(t_i)) \leftarrow PsiEval(\Upsilon(t_i))$ 
9:    $\Delta(S(t_i)) \leftarrow DeltaEval(\Upsilon(t_i))$ 
10:   $\Xi(t_i) \leftarrow Rank(\Psi(S(t_i)), \Delta(S(t_i)))$ 
11: end for

```

---

The algorithm starts (lines 3-4) with the evaluation of the clauses. *Degree* and *Distance* indicators for a requested clause specified in  $lc$  and an offered clause specified in  $L(S_j)$  are evaluated as described previously. Line 6 discards services having offered license specifications that do not match any requested clauses. The services that are not discarded are saved in  $\Upsilon(t_i)$ . The algorithm proceeds checking if the set of filtered services is empty (line 7). The emptiness of this set indicates that  $S(t_i)$  does not contain services that are able to satisfy the requested license specifications mentioned in  $lc$ . In this case, a service closer to the request is recommended. The consumer can decide to terminate the process or to accept the proposed service (changing his/her license specifications). The algorithm proceeds (lines 8-9) evaluating *Net Degree* ( $\Psi$ ) and *Net Distance* ( $\Delta$ ) in order to link each services in  $\Upsilon(t_i)$ . The services in  $\Upsilon(t_i)$  are ordered based on  $\Psi$  and  $\Delta$ . A set of ranked services are placed in  $\Xi(t_i)$  (line 10).

### 3 License Aware Service Selection (LASS) Framework

LASS framework supports the selection of services based on licensing specification in addition to performing service selection with functional and non-functional properties. Based on the FPs, NFPs, and licensing specifications by a service consumer, the LASS framework selects a service that best matches with the requested specifications. The LASS framework tries to find if any service

advertisements given by providers match the request of consumers in service functionality at first, then followed to match the specified NFPs. There may be always the possibility of more than one services, offering similar functionality that differ in their licenses. Figure 1 depicts the LASS framework which comprises of the following components.

- *User Interface*: supports consumers to specify FPs, NFPs, and license clauses based on which services would be selected.
- *Service Selection Request Handler*: receives FPs, NFPs, and license specifications from consumers.
- *FP/NFP Selector*: discovers a set of services satisfying the required functional and non-functional parameters. The techniques for selecting web services based on FPs and NFPs are not the focus of this paper and they are built on well-established works and considered as plug-ins of the LASS framework.
- *License Selector*: discovers a set of services based on matching the offered licensing specifications of functionally matching services against the requested licensing specifications. This component includes the algorithm introduced in this paper.
- *Service Information*: an XML-based repository where information associated with services are stored. In our framework, we utilize SEMF [4] which is able to manage different types of web service related information, including license and QoS.

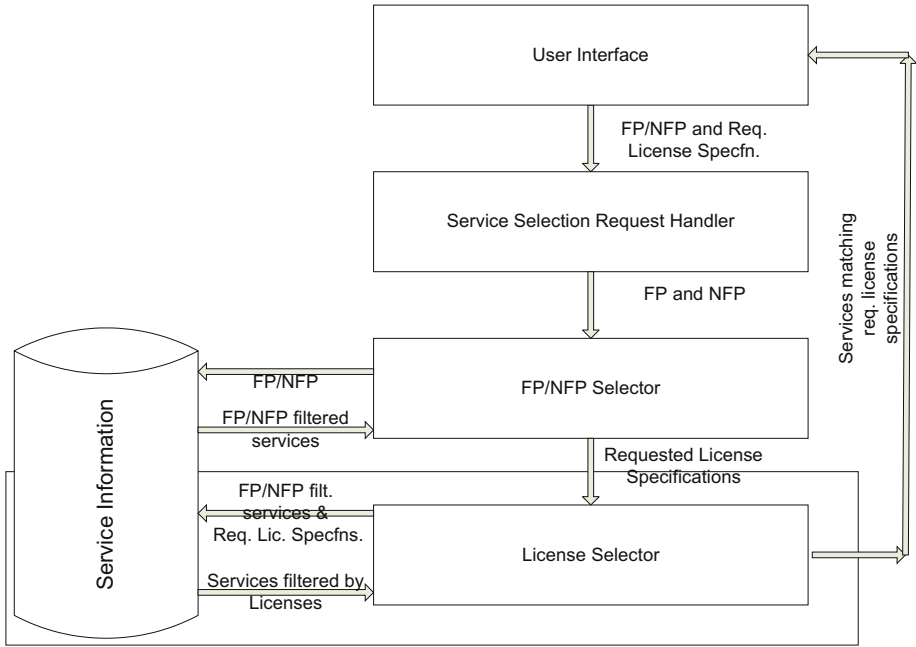
For a given functionality (expressed by a set of functional parameters), NFPs, and a set of requested license specifications, the framework performs license-aware selection of services in the following two steps.

1. A set of services are retrieved that match with functional parameters and NFPs.
2. These set of services having offered license specifications are filtered in order to retrieve a set of services that satisfy the requested license specifications.

## 4 Related Work and Discussions

The increasing availability of services that offer similar functionalities requires to enhance the traditional functionality-based service selection process [5,6] with an additional selection phase that identifies the services that better fulfill a set of NFPs requested by the actual user.

The selection of services based on non-functional specifications has been studied intensively by the research community. Several approaches are based on semantically rich descriptions of non-functional parameters. The approach proposed in [7] represents a solution for matching NFPs of web services represented using WS-Policy. In [8], declarative logic-based matching rules and optimization methods are applied for optimal service selection. A dynamic web service



**Fig. 1.** License aware service selection (LASS) framework

selection based on semantic interpretation of offered service capabilities and the parameters specifying the actual request is proposed in [9].

There are also several proposals to address service selection without the use of semantics. A modified logic scoring preference method of evaluating non-functional aspects is proposed in [10]. A framework supporting brokers in selecting web services based on the required QoS for autonomic grid environments is proposed in [11].

In our earlier work [3], we have presented a semantic approach for selection of services by evaluating both qualitative and quantitative NFPs. In this paper, we have extended our approach described in [3] that can be used for selection of services whose descriptions are not semantic (as in the case of ODRL-S descriptions). Moreover, we evaluate the degree of match when service offers are expressed as interval of values, thereby overcoming a limitation of [3]. This makes the process of service selection more realistic allowing the description of NFPs that can assume any value in the given interval.

To the best of our knowledge, there exists no work on selection of services based on their license specifications. Google<sup>2</sup> and Yahoo<sup>3</sup> search engines provide advanced options to retrieve contents based on requested licenses. However, these options restrict consumers with limited specifications of licenses.

<sup>2</sup> [http://www.google.com/advanced\\_search?hl=en](http://www.google.com/advanced_search?hl=en)

<sup>3</sup> <http://search.yahoo.com/web/advanced?ei=UTF-8>

## 5 Concluding Remarks

Being a way to manage the intellectual rights between service consumers and service providers, licenses are critical to be considered in services. In this paper, we have illustrated a novel methodology for selection of services by matchmaking of license clauses requested by a consumer and offered by several providers. Our LASS methodology describes the selection of services based on all possible clauses of licenses. Following the LASS methodology, we have presented the LASS framework that integrates the process of license aware service selection with functional and non-functional properties. We are currently extending the framework to allow for specifying multiple functionalities, each functionality with differing license specifications and then finding a composite service associated with a composite service license that meets the requested license specifications.

## References

1. Gangadharan, G.R., Weiss, M., D'Andrea, V., Iannella, R.: Service License Composition and Compatibility Analysis. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 257–269. Springer, Heidelberg (2007)
2. Gangadharan, G.R., Comerio, M., Truong, H.L., D'Andrea, V., De Paoli, F., Dustdar, S.: License-aware Service Selection. In: Proc. of the IEEE Conf. on Enterprise Computing, E-Commerce and E-Services, EEE 2008 (2008)
3. Comerio, M., De Paoli, F., Maurino, A., Palmonari, M.: NFP-aware Semantic Web Services Selection. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC (2007)
4. Treiber, M., Truong, H.L., Dustdar, S.: SEMF - Service Evolution Management Framework. In: Proc. of the 34th Euromicro Conf. on Software Engineering and Advanced Applications (2008)
5. Brogi, A., Corfini, S.: Behaviour-aware Discovery of Web service Compositions. *International Journal of Web Services Research* 4(3) (2007)
6. Aversano, L., Canfora, G., Ciampi, A.: An Algorithm for Web service Discovery through their Composition. In: Proceedings of the IEEE International Conference on Web Services, ICWS (2004)
7. Verma, K., Akkiraj, R., Goodwin, R.: Semantic Matching of Web Service Policies. In: Proc. of the Second Intl. Workshop on Semantic and Dynamic Web Processes (2005)
8. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based Selection of Highly Configurable Web Services. In: Proc. of the 16th Intl. Conf. on World Wide Web (2007)
9. Friesen, A., Namiri, K.: Towards Semantic Selection for B2B Integration. In: Proceedings of the 12th International Conference on Web Engineering (2006)
10. Reiff-Marganiec, S., Yu, H.Q., Tilly, M.: Service Selection based on Non-Functional Properties. In: Proceedings of the NFPSLA-SOC Workshop (ICSOC 2007) (2007)
11. Anselmi, J., Ardagna, D., Cremonesi, P.: A QoS-Based Selection Approach of Autonomic Grid Services. In: Proceedings of the 2007 workshop on SOCP (2007)

# Integrated and Composable Supervision of BPEL Processes

Luciano Baresi, Sam Guinea, and Liliana Pasquale

Politecnico di Milano - Dipartimento di Elettronica e Informazione  
via Golgi, 40 – 20133 Milano, Italy  
{baresi,guinea,pasquale}@elet.polimi.it

**Abstract.** In the past few years many supervision techniques were developed to provide reliable business processes and guarantee the established SLAs. Since none of them provided a definitive solution, the paper proposes a new composable approach, where a single framework provides the glue for different probing, analysis, and recovery capabilities. The paper introduces the framework and exemplifies its main features.

## 1 Introduction

Since the uptake of service technology as a means to develop complex distributed systems, *monitoring* and *recovery* tools have played a very significant role. In this paper we refer to the combination of the two as *supervision*, and concentrate on systems designed using BPEL (Business Process Execution Language [4]).

In the past years many supervision techniques were developed, with different requirements they deem important. As for monitoring, the authors themselves have proposed two very different solutions. The first is Dynamo [3], a synchronous and assertion-based approach to monitoring. Even if it is very intrusive, since the process is blocked every time it performs a service invocation, it is able to discover anomalous behaviors as soon as they occur. The second is ALBERT [1], an asynchronous approach based on temporal logic. It is less intrusive, since all the assertions are checked in a separate thread but it can capture anomalies only when the process has already proceeded beyond the point in which they were generated. Other works, like VieDAME [7], check non functional properties (e.g., response time, accuracy) by analyzing the messages the process exchanges with partner services. If we move to recovery, the authors have tackled it with WSReL [2], which offers a set of pre-defined atomic recovery actions (e.g., service substitution, email notification, rollback), that can be mixed to create complex recovery strategies. Other approaches focused on service substitution, applying dynamic binding techniques, as proposed by Colombo et al. [5], or using an AOP-based extension of ActiveBPEL, as proposed by Moser et al. [7].

Although each of these approaches is particularly effective in its own sub-domain, none of them provides a holistic solution that easily accommodates all different clients, in terms of qualities of interest and required analysis. Instead of searching for one definitive solution, we provide an integrated framework in

which diverse solutions can be combined to exploit their main advantages and meet different users' needs.

Our vision of a unified framework builds upon the decoupling of data collection, monitoring, and recovery. Data collection is independent of the types of supervision approaches combined. Monitoring uses these data to check functional and non-functional properties, while recovery uses them, together with the monitoring results, to attempt to fix the process, produce a log, or perform post-mortem activities to prevent them from happening again.

The paper is organized as follows. Section 2 presents our integrated framework and shows. Section 3 describes some example rules for the interplay of data collection, monitoring, and recovery. Section 4 concludes the paper.

## 2 Integrated Supervision Framework

When conceiving our framework, we wanted to satisfy different requirements. It had to support different quality dimensions and different analyses. Synchronous analysis can be used to evaluate punctual process properties (e.g., the response time of partner services). Asynchronous analysis can be exploited to measure temporal process properties (e.g., the number of times a synchronous check is violated). Post-mortem analysis can be used to construct a symptom model of process failures. Our framework should apply suitable recoveries with different timeliness depending on the analysis that signaled the violation.

The distinction among data collection, monitoring and recovery allowed us to conceive a neater architecture. Data collection fosters the neat separation between monitoring and recovery activities. We support *internal data*, which carry the internal state of the process, *external data*, which provide information from the surrounding environment, and *historical data*, which represent information collected in past executions. Different monitoring approaches are also able to share partial results and collaborate towards a more complete final assessment. Our framework can trigger corrective actions on the same process instance, on different instances (of different processes), and also on the process definition. To this end, we allow the interplay between synchronous and asynchronous actions and we provide conflict resolution mechanisms associating them to a priority.

Figure 1 shows the overall architecture of our solution. Each *BPEL Engine* is an instance of *ActiveBPEL Community Edition Engine* augmented with AOP (aspect-oriented) probes to collect process state data. The *Data Manager* is responsible for collecting external data, and for retrieving and storing historical data from/in the *Data Repository*. The *Monitoring Farm* holds the monitoring plug-ins we want to use, while the *Recovery Farm* holds the recovery capabilities.

The *Event Controller* is the central element of our architecture and it is based on rule engine technology [6]. It is in charge of activating external and historical data collection, as well as any monitoring and recovery activity. While internal variables are passively received from the AOP probes embedded in ActiveBPEL, external and historical variable collection, like also monitoring and recovery activities, are triggered when the process starts or when it reaches a particular



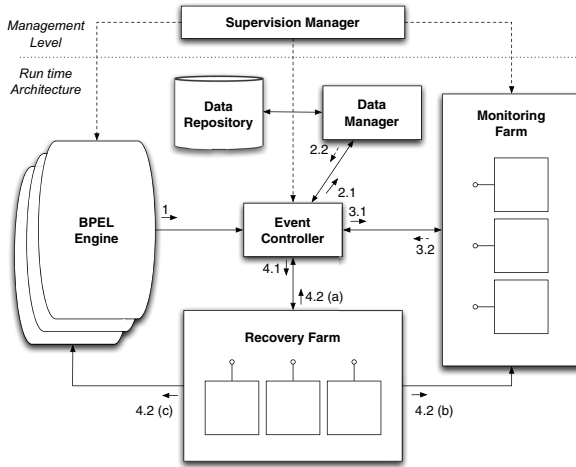


Fig. 1. The Unified Framework

state. This is achieved by defining rules on the data contained in the working memory, that is, process state data, external and historical variables, collected through the *Data Manager*, and analysis results.

Our framework also provides a configuration tool called *Supervision Manager*, used to configure the various components of the framework. In particular, it configures the AOP probes to collect internal data, the *Event Controller*, to retrieve external and historical data, and defines how monitoring and recovery are activated. Finally, it also sets the *Monitoring Farm* with the constraints each plug-in is supposed to check.

The numbered arrows of Figure 1 explain how the framework works. Before starting the execution of any process, the *Supervision Manager* configures the different components to allow them to correctly perform supervision tasks. After this, every time an executing process terminates an activity, the AOP probes collect internal data and give them to the *Event Controller* (transition 1), which inserts them in its embedded working memory. The data available in the working memory of the *Event Controller* can always activate suitable rules to require the *Data Manager* to retrieve external or historical data (transition 2.1) and store them in its working memory (transition 2.2).

### 3 Example Configuration Rules

This Section shows some configuration rules to exemplify how the framework works. The following rule shows how to collect an external variable.

```
rule "external_data_collection"
no-loop true
when
```

```

p : ProcessState(process = "SMS_Process",
  engineID = "localhost:8080", instanceID = -1,
  username = "jack.burton", /process/sequence/Invoke_SMS_Send)
then
  ExternalVariable e = DataCollector.pull(wSDL, operationName, input);
  insert(e);

```

The external variable is collected when process `SMS_Process` reaches activity `/process/sequence/Invoke_SMS_Send` and `ProcessState p` appears in the working memory. Since we currently support external probes implemented as Web services, we need the endpoint reference (`wSDL`), the name of the operation to invoke (`operationName`), and its input parameters (`input`). The datum is then inserted in the working memory through operation `insert`, defined in the *then* clause. Notice that the input parameters given to an external probe can include any data present at that time in the working memory of the *Event Controller*.

Besides data collection and storage, rules are also responsible for dispatching data to interested monitoring plugins (transition 3.1). This feeds the creation of special dispatching rules in the *Event Controller*'s rule engine. For each monitoring plugin being used, designers must then express the actual assertions they want them to check. Each expression is given in the language used by the plugin.

The following rule uses ALBERT to asynchronously check whether the accuracy of a service endpoint is greater than a predefined threshold. It passes ALBERT all necessary data to perform monitoring: the current partner service endpoint (variable `pLink`) and its current accuracy (variable `mResult`) calculated by VieDAME. Variable `pState` detects the process location in which the monitoring result and the service endpoint need to be forwarded.

```

rule "ALBERT data"
when
  pState : ProcessState(processName == SMS_Process,
    location == "/process/sequence/Receive_SMS_Notification") &&
  pLink : PartnerLink(processName == SMS_Process,
    location == "/process/sequence/Receive_SMS_Notification",
    pLinkN : name == "SMS_sending")
  mResult : MonitoringResult(
    pluginName == "VieDAME", n : rule == "accuracy",
    processName=="SMS_Process",
    location == "/process/sequence/Receive_SMS_Notification")
then
  List<Datum> data = new ArrayList<Datum>();
  data.add(n, mResult);
  data.add(pLinkN:pLink);
  dispatchData("ALBERT", pState, data);

```

ALBERT checks the following rule:

```

processName: SMS_Process
rule: onEvent(/process/sequence/Receive_SMS_Notification) ->
  $accuracy/partnerLink[@name == $SMS_sending/name]
  /endpoint[@value == $SMS_sending/address]/accuracy > THRESHOLD;

```

where `$accuracy` contains the monitoring results provided by VieDAME, while `$SMS_sending` contains the partner link being used by process `SMS_Process`.

As soon as a monitoring result is produced by the *Monitoring Farm*, it is sent to the *Event Controller* to insert it in its working memory (transition 3.2). These new data can fire rules that request the *Recovery Farm* to apply some recovery actions (transition 4.1) to modify the executing process instances, change how the *Event Controller* works, or modify the checks performed by the monitoring plug-ins (transition 4.2). At this point, the *Event Controller* has all the monitoring results and can activate different recovery strategies by communicating with the *Recovery Farm*. The recovery plugins in the *Recovery Farm* can access the process internals using AOP probes. In our current implementation we provide the same set of recovery actions defined in [2], to tune the overall degree of monitoring, both in terms of activities being performed and approaches being used, and also asynchronous recovery, to work on process definitions directly and change how future process instances are configured.

Synchronous recovery needs the process to be blocked the process execution until it completes, while asynchronous recovery can be performed in parallel with the process activities till the process reaches a specific target location. In has to stop until recovery completes. If there are multiple asynchronous recoveries ready to be executed, the *Event Controller* chooses among them depending on their priority level as well as on the strategy the framework must use to interpret these priority values. For example, if the designer decides to use an *exclusive* strategy, a recovery with the highest priority disables all the others. This strategy is realized by assigning the same activation-group attribute to potentially conflicting rules. The activation-group attribute guarantees that only the rule with highest priority is executed. If the designer decides to go with an *all* strategy, all recoveries are executed and the order is given by their priority values. Recovery strategies with the same priority level can execute in any order. If the designer decides not to provide priorities, recovery strategies can be executed in any order.

The following rule defines a recovery for when the previous ALBERT rule is not verified (variable `malbert`). This recovery has the effect of changing the process definition, substituting a partnerLink with the one suggested by VieDAME to have the best accuracy (variable `mviedame`).

```
rule 'SMS_asynch_substitute'
activation-group "asynch_substitution"
salience 2
  when
    malbert : MonitoringResult(plug-in == "ALBERT",
      processName == "SMS_Process",
      location == "/process/sequence/Receive_SMS_Notification",
      result == false) &&
    mviedame : MonitoringResult(plug-in == "VieDAME",
      type == "bestAccuracy"
      processName=="SMS_Process",
      location == "/process/sequence/Receive_SMS_Notification")
  then
```

```

recovery.setProcess('SMS_Process');
recovery.setTargetState('/process/sequence/Invoke_SMS_Service');
recovery.asyncSubstitute(mviedame.getPartnerLinkName,
    mviedame.getWsdL(), mviedame.getOperation(), mviedame.getInput(),
    mviedame.getTransformationRule());

```

The asynchronous recovery is assigned a priority of 2 (**salience**) and an activation-group named **async\_Substitution**, and it is configured to execute before activity **/process/sequence/Invoke\_SMS\_Service**. The recovery consists in an invocation of method **asyncSubstitute** provided by our framework. We pass to the recovery method the information gathered from ViEDAME, as well as an indication of the process on which the recovery has to take place (**SMS\_Process**).

## 4 Conclusions and Future Work

We propose a flexible and customizable supervision framework for BPEL processes, integrating different monitoring and recovery techniques. The framework architecture conceptually decouples data collection, monitoring, and recovery. Leveraging rule engine technology we dispatch data to various monitoring components, aggregate different monitoring results, and choose the most appropriate recovery. In our future work, we will investigate high level supervision languages and tools that can help hide the complexities that lie in the definition of the rules used by our *Event Controller*. We also want to discover suitable ways to synthesize these rules starting from requirements and SLA standards.

## References

1. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of Web Service Compositions. *IET Software* 1(6), 219–232 (2007)
2. Baresi, L., Guinea, S.: A Dynamic and Reactive Approach to the Supervision of BPEL Processes. In: *Proceedings of the 4th India Software Engineering Conference* (2008)
3. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: *Proceedings of the 3rd International Conference on Service Oriented Computing*, Amsterdam, The Netherlands, December 12–15, pp. 269–282
4. OASIS, Business Process Execution Language for Web Services, Version 1.1
5. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In: *Proceedings of the 3rd International Conference on Service Oriented Computing*, Chicago, IL, USA, December 4–7, 2006, pp. 191–202 (2006)
6. Proctor, M., et al.: Drools, <http://www.jboss.org/drools/>
7. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for WS-BPEL. In: *Proceedings of the 17th International Conference on World Wide Web*, Beijing, China, April 21–25, 2008, pp. 815–824 (2008)

# Optimised Semantic Reasoning for Pervasive Service Discovery

Luke Steller and Shonali Krishnaswamy

Faculty of Information Technology, Monash University, Melbourne, Australia  
laste4@student.monash.edu.au,  
Shonali.Krishnaswamy@infotech.monash.edu

**Abstract.** A key challenge in delivering mobile services is to improve the relevance of discovered services, as mobile environments are very dynamic with rapid changes to user context. This paper presents m-Tableaux - an optimised semantic reasoning approach to support pervasive service discovery which aims to efficiently leverage the computational resources available of mobile devices. We present performance evaluation of the m-Tableaux optimisation strategies which clearly demonstrate its operational feasibility on a mobile device.

## 1 Introduction and Related Work

A mobile user arriving in Sydney airport can currently utilise kiosk touch screens to search for stores and points of interest in the airport. The increasing computational capacity of small devices such as PDAs and mobile phones provide new opportunities for “on board” service discovery taking user context and request complexity into consideration, rather than limited and inconvenient fixed point kiosks. There are two models for discovery in this context: 1. the kiosk becomes a centralised high-end server which performs all matching on behalf of the user or 2. a decentralised model where the kiosk acts only as a repository of information and the user’s device performs all matching “on-board”, on a needs basis. We advocate the decentralised model for environments where a central authority does not exist or does not want the responsibility of providing and maintaining a centralised service and the user’s do not want to incur the costs in using such a service.

While current service discovery architectures such as Jini [1] and UPnP [2] use either interface or string based syntactic matching, there is a growing emergence of DAML-S/OWL-S semantic matchmakers such as CMU Matchmaker [3] and DIANE [4] which support varying levels of semantic reasoning, but require a centralised high-end node to perform reasoning. The reasoners which matchmaking architectures use, such as FaCT++ [5] and RacerPro [6]), quickly give “Out of Memory” errors when ported directly to resource limited devices in their current form. In this paper, we present our mTableaux algorithm with incorporates optimisation strategies to enable reasoning on resource constrained mobile devices. Section 2 describes our architecture and our mTableaux algorithm, section 3 provides a performance evaluation and in section 4 we conclude the paper.

## 2 Resource-Aware and Cost-Efficient Pervasive Service Discovery

Our decentralised pervasive service discovery architecture is illustrated in figure 1, which resides on the user’s resource constrained device. The remainder of this paper concentrates on the semantic mTableaux, while the adaptive discovery manager is future work.

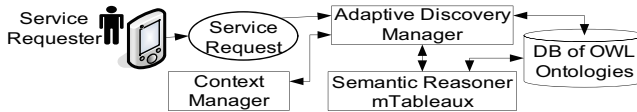


Fig. 1. Pervasive Service Discovery Architecture

### 2.1 Semantic Reasoners

The effective employment of semantic languages requires the use of semantic reasoners such as Pellet [7], FaCT++ [5] and RacerPro [6]. Most of these reasoners utilise the widely used Tableaux [8] algorithm. DL Tableaux reasoners, such as Pellet, reduce all reasoning tasks to a consistency check. Tableaux is a branching algorithm, in which disjunctions form combinations of branches in the tree. If all branches contain a clash, meaning a fact and its negation are both asserted, then a clash exists for all models of the knowledge base. Inferred membership for an individual  $I$  to class type  $RQ$ :  $I \in RQ$ , is checked by asserting that  $I$  is a member of the negation of  $RQ$ . If a clash exists for all models of the knowledge base then the membership is proven.

This paper concentrates on strategies to optimise the Tableaux algorithm to enable reasoning on small/resource constrained devices with significant improvements to response time and avoiding “Out of Memory” errors as encountered in [9]. Our mTableaux algorithm relates to optimising membership type checks ( $I \in RQ$ ). It involves a range of optimisation strategies such as: 1. selective application of consistency rules, 2. skipping disjunctions, 3. establishing pathways of individuals and disjunctions which if applied would lead to potential clashes and associating weights values to these elements such that the most likely disjunctions are applied first, by 3a. ranking individuals and 3b. ranking disjunctions.

Application of consistency rules to a subset of individuals only, reduces the reasoning task. Consider the universal quantifier construct of the form  $\forall R.C = \{ b.(a, b) \in R \rightarrow b C \}$  [10], where  $R$  denotes a relation and  $C$  denotes a class concept, which implies object fillers for  $R$  must be of type  $C$ . We define the individual subset to  $I$  and those individuals which fill  $R$  for  $I$ . Disjunctions are only applied when they contain a concept which is contained within or can be unfolded from the request type  $RQ$ , including quantifier role filler types. A weighted queue is established to rank individuals and disjunctions such that the highest are applied first. Rankings are established by recursively checking

the class types contained in a disjunction to see whether these potentially lead to future clash. In the next section we more formally describe each of these strategies.

## 2.2 mTableaux Strategies

**Selective Consistency:** Let  $S$  denote a set of individuals  $e$  which can have completion rules applied to them,  $S \equiv \{e_1, e_2, e_n\}$ .  $S$  initially contains individual  $I$ . Let  $avR$  denote the distinct set of roles  $r$  contained in any universal quantifier construct which will be applied to any individual  $e$  in  $S$ , where  $avR \equiv \{r_1, r_2, r_m\}$ . Add any object individual  $o$  which fills any role  $r$  contained within  $avR$ , to  $S$ . The same strategy is recursively applied to all individuals added to  $S$ , or whenever a new universal quantifier is applied to an individual in  $S$ . For example consider that individual `LaserPrinter1` has a role `hasComm` which contains the individual `Modem1`. Initially `LaserPrinter1` is contained in  $S$ , and the assertion of  $\forall hasComm. \neg (Modem)$ , results in `Modem1` also being added to  $S$ .

**Disjunction Skipping:** Where a service request  $RQ$  is a conjunction class type, let  $T$  denote the set of conjunct terms  $t$ , where  $T \equiv \{t_1, t_2, t_n\}$ . Let  $DS$  denote a set of types derived from  $RQ$ . For each  $t_i$  in  $T$ , add  $_i t$  to  $DS$  as well as all expressions which can be unfolded from  $t_i$ . Where any universal or existential quantifier, or cardinality restriction is encountered, add its role filler type  $C$  and all expressions which can be unfolded from  $C$ . During reasoning apply only those disjunctions  $D$ , containing disjunct terms  $d$ , where any  $d$  is contained within  $DS$ , otherwise skip it. For example, where  $RQ \equiv SupportsModem \cap SupportsColour$ , and  $SupportsModem \equiv \exists hasComm.Modem$ ,  $DS \equiv \{SupportsModem, SupportsColour, Modem\}$ .

**Weighted Individuals and Disjunctions:** A single weighted queue  $QI$  of individuals  $i$  is established and each individual has a weighted queue  $QD$  of disjunctions  $d$ , such that  $QD \equiv \{i_1, i_2, i_n\}$  and  $QD \equiv \{d_1, d_2, d_m\}$  and there are  $n$   $QD$ s. Each element  $i$  and  $d$ , is associated with a weight value which is used to rank the elements in decending order. Disjunctions with the highest weight in the  $QD$ , which relates to the individual with the highest weight in  $QI$ , are applied first. For any disjunct type element  $c$  of a disjunction  $D$  relating to individual  $I$  (rank disjunctions) or any type  $c$  applied to individual  $I$  (arising from the application of a disjunction) which did not give rise to a clash (rank individuals), search for a pathway to a potential clash. This occurs by checking to see if the application of any expression which can be unfolded from  $c$ , may give rise to a future clash. Where  $c$  unfolds into a disjunction or conjunction, the negation of their constitute elements is checked for in individual  $I$ . Where the expression is a universal quantifier, each object individual for  $I$  is checked for a potential clash for the quantifier's role filler. If potential clash path is detected the weight of all individuals and disjunctions involved in the path is increased. For example, for the disjunction.  $D \equiv SupportsModem \cup SupportsColour$  where  $SupportsModem \equiv \forall hasComm.Modem$  and Individual  $I$  has the type  $\neg Modem$  a potential clash is found and the weighting for disjunction  $D$  and individual  $I$  is increased in their queues,  $QD$  and  $QI$  respectively.

### 3 Implementation and Performance Evaluation

In this section we provide two case studies in order to evaluate our mTableaux algorithm. In case study 1, Bob is walking around at his university campus and wishes to locate a fax machine. This was implemented into an ontology containing 141 classes, 337 individuals and 126 roles. In the second case study Bob, in a foreign city centre, wants to find a movie cinema with an Internet café. The ontologies for this scenario contain 204 classes, 241 individuals and 93 roles.

We implemented the selective consistency, rank individuals, rank disjunctions and skip disjunctions strategies defined in the previous section, in the Pellet v1.5 reasoner. We selected Pellet because it is open source and implemented in java, allowing for easy portability to small devices. Pellet supports OWL-DL with SHOIN expressivity. Table 1 presents a request for each case study and a positive/matching A and negative/non-matching B service individual, to be compared with each request.

**Table 1.** Type checks

Case	Request	Individual	Expected Result
1	Fax Laser Printer	A: #LaserPrinter1	Match
		B: #LaserPrinter2	No Match
2	Movie Cinema	A: #MovieCinema1	Match
		B: #MovieCinema2	No Match

Each of the four match checks in table 1 was executed 16 times using various randomly selected combinations of our optimisation strategies, as outlined in table 2. Test 16 represents normal execution of the Tableaux algorithm, with none of our optimisations strategies enabled.

**Table 2.** Optimisation tests

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C: Selective Consistency	✓	✓	✓	✓	✓	✓		✓	✓							
S: Skip Disjunctions	✓	✓	✓	✓			✓			✓	✓	✓				
D: Rank Disjunctions			✓	✓				✓	✓	✓	✓		✓	✓		
I: Rank Individuals	✓	✓				✓		✓	✓		✓	✓	✓			✓

We performed an evaluation on a HP iPAQ hx2700 PDA, with Intel PXA270 624Mhz processor, 64MB RAM, J2SE JVM, allocated 15MB of memory. Successfully executed tests returned the expected result shown in table 1. Figure 2 shows two graphs, which each show the consistency time to perform a type membership check for individual A and B against the request for the tests in table



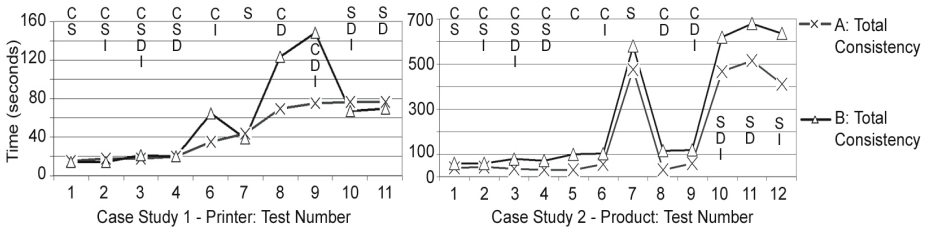


Fig. 2. Processing time taken to perform each test

2, for case study 1 and 2, respectively. Test 16, with no optimisations resulted in the “Out Of Memory” exception, necessitating our optimisations.

The figure shows that mTableaux optimisation can complete case study 1 and 2 in 18 and 35-70 seconds, respectively. This illustrates significant performance improvements in both scenarios. The selective consistency and skip disjunctions strategies were the most effective when used together. However, while we found that rank disjunctions and individuals did reduce the number of branches applied, these did not significantly reduce reasoning time. Results also showed that the optimisations were less effective in improving performance for non-matching individuals B than matching individuals A, because the algorithm continues applying branches and completion rules until a clash is found.

Figure 3 illustrates the overhead cost incurred by each optimisation strategy.

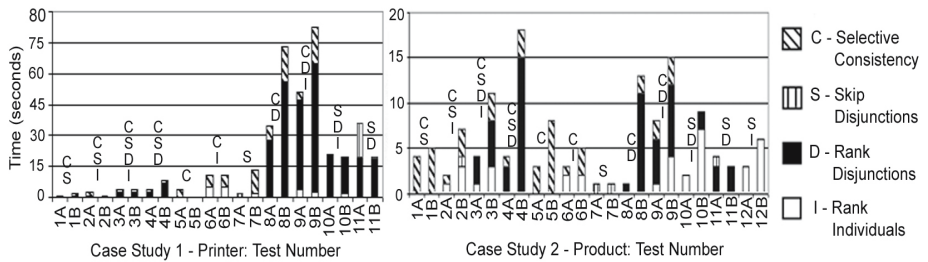


Fig. 3. Optimisation overhead breakdown. Each test was conducted for matching individual A and non-matching individual B.

We observed that selective consistency and skip disjunctions incurred low overhead, especially when used together. Rank disjunction overhead was significantly higher for tests 8 and 9 for both case studies due to the skip disjunction strategy being disabled, resulting in more disjunctions to evaluate. In addition to these results, we present a comparison between mTableaux, Pellet and Racer Pro in [11] which shows mTableaux out performs these, without reducing recall or precision.

## 4 Conclusion and Future Work

mTableaux was shown to significantly improve the performance of pervasive discovery reasoning tasks in two case studies, enabling them to be completed on small resource constrained devices. In future work we are leveraging our rank disjunctions and individuals strategies to adaptively reduce the number of branches applied when resources become low, to provide a less accurate result with a level of confidence to avoid “Out Of Memory” errors.

## References

1. Arnold, K., O’Sullivan, B., Scheifler, R.W., Waldo, J., Woolrath, A.: The Jini Specification. Addison-Wesley, Reading (1999)
2. UPnP. Universal Plug and Play (UPnP), [cited March 12, 2007] (2007), <http://www.upnp.org>
3. Srinivasan, N., Paolucci, M., Sycara, K.: Semantic Web Service Discovery in the OWL-S IDE. In: 39th Hawaii International Conference on System Sciences, 2005, Hawaii (2005)
4. Küster, U., König-Ries, B., Klein, M.: Discovery and Mediation using DIANE Service Descriptions. In: Second Semantic Web Service Challenge 2006 Workshop, Budva, Montenegro, June 15-16 (2006)
5. FaCT++ [cited May 1, 2007] (2007), <http://owl.man.ac.uk/factplusplus>
6. RacerPro. [cited May 23, 2007] (2007), <http://www.racer-systems.com>
7. Pellet (2003), <http://www.mindswap.org/2003/pellet>
8. Horrocks, I., Sattler, U.: A Tableaux Decision Procedure for SHOIQ. In: 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), Morgan Kaufmann, San Francisco (2005)
9. Kleemann, T.: Towards Mobile Reasoning. In: International Workshop on Description Logics (DL 2006), Windermere, Lake District, UK, May 30 - June 1 (2006)
10. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
11. Steller, L., Krishnaswamy, S.: Pervasive Service Discovery: mTableaux Mobile Reasoning. In: International Conference on Semantic Systems (I-Semantics). Graz, Austria (2008)

# COSMA – An Approach for Managing SLAs in Composite Services

André Ludwig and Bogdan Franczyk

University of Leipzig  
Information Systems Institute  
Marschnerstr. 31, 04109 Leipzig, Germany  
{ludwig, franczyk}@wifa.uni-leipzig.de

**Abstract.** Service provisioning is largely built on agreements specifying the mutual responsibilities of service providers and their customers with respect to functional and non-functional parameters. Current SLA management approaches, i.e. WSLA, WS-Agreement, or WSOL, provide extensive SLA language formalizations and management frameworks. However, they focus on bi-lateral service requester/provider constellations neglecting the SLA management requirements of composite service providers, i.e. managing SLAs with atomic service providers and with composite service requesters and aligning both with each other. A SLA management solution for composite services has to consider the contribution of sourced services - formalized in their (atomic) SLAs (ASLA) - in the management of the provided service - formalized in its respective (composite) SLA (CSLA). This paper presents the novel COmposite Sla MAnagement (COSMA) approach for an integrated management of atomic and composite SLAs during their entire lifecycle. It can be utilized for controlling the relationships between ASLAs/CSLAs and thus serves as the basis for managing and optimizing the SLAs involved in composite services.

## 1 Introduction

Service-oriented computing (SOC) has emerged as the most promising design paradigm for next-generation distributed information systems. The vision that goes along with SOC is that once standards have established themselves and become widely adopted by service providers and requesters, a globally available infrastructure for hosting and accessing services will be created [1]. This infrastructure will allow service providers to offer multiple services with individually adapted service capabilities to their changing customers that can dynamically and on-demand bind these services into their own applications; thus forming a market of services. The advent of service markets on the basis of the SOC paradigm will pave the way for a service-oriented business model which is referred to as composite service provider (CSP) [2]. A composite service provider requests services from external service providers (atomic services) and provides these services according to a process flow as composite service to service requesters.

In this constellation, a composite service provider acts as an independent, self-interested business entity, motivated to fulfil own goals, i.e. be profitable and maximize customer satisfaction.

In order to control the interface between service requesters and providers, a contractual basis in form of service level agreements (SLA) is needed. Current SLA management approaches applicable for SOC environments, i.e. WSLA [3], WS-Agreement [4], or WSOL [5], provide extensive SLA language formalizations and management frameworks. However, they focus on bi-lateral service requester/provider constellations neglecting the SLA management requirements of composite service providers, i.e. managing SLAs with atomic service providers and with composite service requesters and aligning both with each other. A SLA management solution for composite services has to consider the contribution of sourced services - formalized in their (atomic) SLAs (ASLA) - in the management of the provided service - formalized in its respective (composite) SLA (CSLA). Since composite services are created on-the-fly also their SLA management must be realized on-the-fly. Manual SLA management is not appropriate for composite service providers and automation support is required, i.e. for creation, monitoring and evaluation of SLAs.

In this paper, an approach for the management of SLAs involved in composite services during their entire lifecycle is presented (COSMA - COmposite SLA Management). COSMA can be utilized for controlling the relationships between SLAs and thus serves as a basis for managing and optimizing the dynamics between SLAs of composite services. The paper is structured as follows: section 2 presents the central idea behind the COSMA approach and briefly presents its constitutional elements. Section 3 refers to a use case and demonstrator. Section 4 concludes the paper.

## 2 Composite SLA Management Approach (COSMA)

The central idea behind the COSMA approach is the integration of all SLAs a composite service provider has to deal with into one composite SLA management document. This composite SLA management document, which is defined as COSMA<sub>doc</sub>, contains all contractual information of all involved SLAs and in addition the relationships and dependencies that exist between the different aspects of atomic and composite SLAs. On the basis of a COSMA<sub>doc</sub>, the SLA management system of a CSP is able to understand how atomic SLAs contribute to the provision of the composite SLA. This knowledge enables a CSP to control and optimize its SLA management activities in providing a composite service. This includes, in particular, planning and negotiating SLAs, monitoring and evaluating SLAs (cf. SLA lifecycle as outlined in [6]). The COSMA approach consists of the following three parts:

- COSMA<sub>doc</sub>: A generic information model that integrates contractual data, SLA management data, and elements for the expression of dependencies and relationships between SLA elements.

- COSMAframe: A conceptual framework that outlines the components that are necessary for the management of the composite SLA lifecycle on the basis of a COSMAdoc instance.
- COSMALife: An integrated set of SLA management practices that use COSMAdoc instances to cover the phases of the SLA lifecycle.

COSMAframe, COSMAdoc, and COSMALife are embedded into an operation and management system (OMS) of a CSP, i.e. platform as proposed in [7].

## 2.1 Information Model COSMAdoc

As the informational heart of COSMA, COSMAdoc provides a generic information model that encapsulates contractual and management information of SLAs. It comprises a set of SLA documents (CSLA and ASLAs) and elements for the expression of relationships and dependencies between those SLAs. For every composite service an individual instance of COSMAdoc is created and bound with the service. For all management activities defined in COSMALife, the COSMAdoc instance is used. Different components of COSMAframe execute their tasks on the basis of the COSMAdoc instance; thus, it is the basis for performing complete, composition-wide SLA negotiations and compliance evaluations. COSMAdoc serves as an internal composite SLA management tool used by a CSP; embedded SLA documents carry only contractual data and can be exposed to involved parties without publishing COSMAdoc instance-internal management information. At the top-level, COSMAdoc consists of the following six core sections (Fig. 1):

- Header: The purpose of the Header section is to declare contextual information and parameters for the COSMAdoc instance (description, owner, version number, definition of semantic models and languages).
- ServiceComposition: The ServiceComposition section includes the orchestration script. Since the script expresses the general relationships and structure between all involved services, it governs the mapping and dependencies between SLA parameters of composite and atomic SLAs.
- SlaSetAssembly: The purpose of the SlaSetAssembly section is to capture all SLAs involved in a composite service. A SLA element carries the contractual information between the involved two parties. It carries no information that may be used by the CSP to manage the mapping of ASLA elements to CSLA elements. Hence, each SLA document can be used to represent a public agreement between the involved two parties and can be accessed by both parties. This ensures separation of concerns and provides a sound contractual basis by means of a valid SLA document. The COSMAdoc SLA model is based on the comprehensive WS-Agreement SLA model [4] for bilateral agreements. The COSMAdoc SLA model comprises of the sections Name, Context, Terms as defined in WS-Agreement and extends the specification with diverse aspects, i.e. MonitoringTerms, NegotiationTerms, and FinancialTerms in the Terms section.

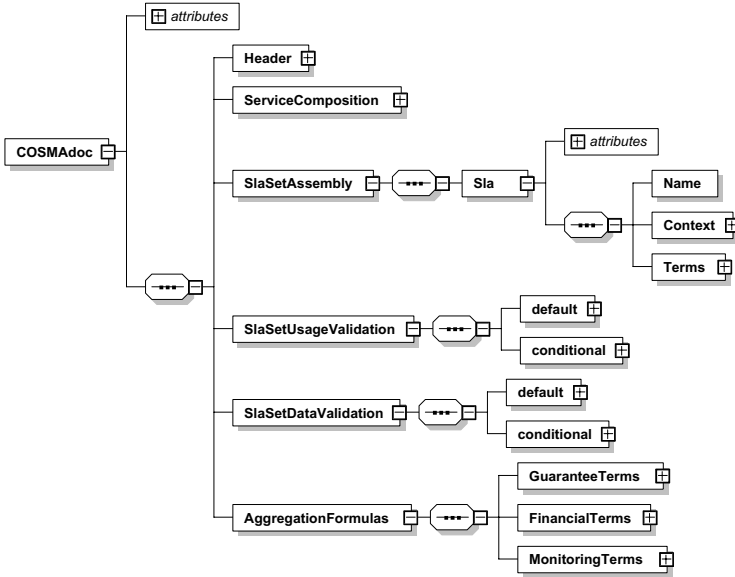
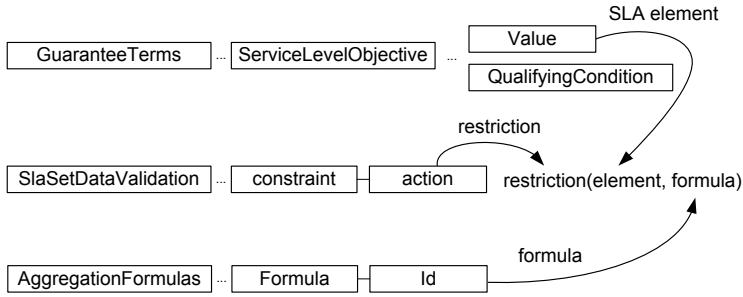


Fig. 1. COSMA doc information model

- SlaSetUsageValidation: The SlaSetUsageValidation section is used to define specific requirements and constraints on the SLA elements of the SlaSetAssembly. These requirements and constraints regard only the content usage of the involved SLA documents (not the data). For instance, elements of a SLA can be declared as mandatory, negotiable, or optional. The technique used to identify elements of the SLA is the definition of pointers. Predicates can be defined as default or depending on a condition.
- SlaSetDataValidation: While the SlaSetUsageValidation section controls the usage of the involved SLA elements, the SlaSetDataValidation controls the data of the SLA elements. It provides means to explicitly enforce, validate, and check the data values of the involved SLAs by defining predicates on them, i.e. setMaxValue, setValueRange etc. Since most of the data validation restrictions on the contents of a CSLA result from the contents of ASLAs, predicates on CSLA elements may refer to aggregation formulas that calculate the value from other ASLA elements. Thus, a predicate with a link to an aggregation formula connects and restricts values of SLAs, i.e. defined in GuaranteeTerms, with aggregation formulas defined in the AggregationFormulas section (cf. Fig. 2).
- AggregationFormulas: Formulas stored in the AggregationFormulas section reflect the relationships and dependencies between SLA elements in composite services. Aggregation formulas can use all types of algorithmic operators and reference SLA elements using pointers. The complexity of these formulas



**Fig. 2.** Connecting SLA elements, aggregation formulas, and predicates

depends on the structure of the service composition script and the number and type of SLA parameters. Generic aggregation patterns for different SLA parameters were proposed i.e. in [9].

### 2.2 Conceptual Framework COSMAframe

COSMAframe is a conceptual framework that outlines the components that are required for an automated processing of COSMA doc instances. COSMAframe presents these components in terms of their generic interfaces, basic behaviours, and functions. COSMAframe consists of the following components (cf. Fig. 3):

- The COSMA Manager is the central management component that triggers all components depending on the necessary steps within the SLA lifecycle. It is the central port of COSMAframe towards external components and provides a COSMA doc validation interface that processes COSMA doc aggregation formulas and predicates.
- The COSMA doc Creator is responsible for creation of COSMA doc instances. Based on a given generic service composition script provided by an external Service Composer, it creates composition-specific COSMA doc instances using a composition decomposer.
- The composition-specific COSMA doc instance is integrated by the COSMA doc Integrator. The COSMA doc Integrator adds different SLA contents, like quality of service parameters or financial parameters, to the included SLA documents. Afterwards, the component inserts different types of restrictions that evolve from the structure of the service composition script and the types of SLA parameters to the COSMA doc instance.
- COSMA doc instances are stored in the COSMA doc Repository.
- The COSMA doc Validator and Violation Detector compares service level measurements against service level objectives defined in ASLAs. It normalizes and aggregates the monitoring data across all ASLAs using stored aggregation formulas and is then able to state whether service levels of the CSLA are violated or not. If a SLA violation is detected, a proposal how to deal with the violation is created and returned to the COSMA Manager.

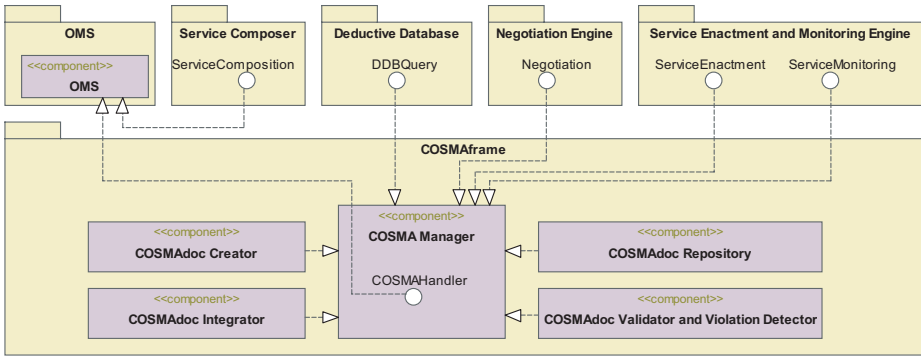


Fig. 3. Components of COSMAframe and external components

External components, i.e. Service Composer, Negotiation Engine etc., are described on the Adaptive Services Grid project website [7].

### 2.3 Composite SLA Management Lifecycle Mechanisms of COSMAlife

COSMAlife presents a set of composite SLA management mechanisms that cover different phases of the SLA lifecycle presented in [9]. The focus in COSMAlife is given to phases SLA creation and negotiation and SLA monitoring and evaluation. Briefly, a successful COSMAlife run-through consists of three steps: (1) creating and integrating, (2) negotiating, and (3) enacting/monitoring/validating a COSMAAdoc instance. First, the COSMAAdoc Creator creates a composition-specific instance from the generic COSMAAdoc template. It uses an internal composition decomposer to atomize the generic service composition script into its generic composition patterns. Afterwards, the COSMAAdoc Integrator integrates SLA content pre-settings, the service composition script, and SLA parameter-specific usage and data validation restrictions. Second, a Negotiation Engine determines the concrete atomic service implementations to use in the composite service (the Negotiation Engine hosts negotiation agents, whereas each negotiation agent negotiates a single SLA document). To ensure that the negotiation outcomes of ASLA negotiation processes are optimized with regard to the CSLA negotiation, the Negotiation Engine iteratively uses a validation interfaces provided by the COSMA Manager which processes predicates/aggregation formulas of the COSMAAdoc instance. Third, after enactment of the composite service, monitoring and evaluation of the involved SLAs is executed. The COSMA Manager receives monitoring results from the Monitoring Engine (as specified in COSMAAdoc) and sends these data to the COSMA Validator and Violation Detector for validation. In case of SLA violations, proposals for consequential actions are determined. The component uses the AggregationFormulas of COSMAAdoc to detect service level violations and decide on the best type of action.



### 3 Use Case and Demonstrator

In order to illustrate the application of the COSMA approach and to support the understanding of COSMA<sub>doc</sub>, COSMA<sub>frame</sub>, and COSMA<sub>life</sub>, a use case was developed. The use case is published in [10]. In addition, a demonstrator was developed which implements the key elements of COSMA.

### 4 Conclusions

The paper addressed the topic of managing the dynamics of SLAs in composite services. For this, the novel composite SLA management approach COSMA was outlined in brief. With COSMA, a composite service provider can control and optimize its SLA management activities, pro-actively plan financial consequences, and dynamically calculate the expected service level objectives from dynamically varying service composition scripts. COSMA represents a conceptual approach for management of SLAs in composite services and is to be interpreted as a starting point that must be extended and adapted to individual requirements of arbitrary scenarios. Thus, the proposed, theoretical approach has to be tested extensively and employed on a number of different scenarios.

### References

1. Papazoglou, M.P.: *Web Services: Principles and Technology*. Prentice Hall, Essex (2007)
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: *Web services: concepts, architectures and applications*. Springer, New York (2004)
3. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. of Network and Systems Management* 11, 57–81 (2003)
4. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: *Web Service Agreement Specification (WS-Agreement)*, <http://www.gridforum.org/documents/GFD.107.pdf>
5. Tasic, V., Patel, K., Pagurek, B.: WSOL - Web Service Offerings Language. In: Bussler, C.J., McIlraith, S.A., Orłowska, M.E., Pernici, B., Yang, J. (eds.) *CAiSE 2002 and WES 2002*. LNCS, vol. 2512, pp. 57–67. Springer, Heidelberg (2002)
6. Ludwig, A., Braun, P., Kowalczyk, R., Franczyk, B.: A Framework for Automated Negotiation of Service Level Agreements in Service Grids. In: Bussler, C.J., Haller, A. (eds.) *BPM 2005*. LNCS, vol. 3812, pp. 89–101. Springer, Heidelberg (2006)
7. Integrated Project Adaptive Services Grid (ASG), <http://www.asg-platform.org>
8. Momotko, M., Gajewski, M., Ludwig, A., Kowalczyk, R., Kowalkiewicz, M., Zhang, J.Y.: Towards adaptive management of QoS-aware service compositions. *J. of Multiagent and Grid Systems* 3, 299–312 (2007)
9. Jaeger, M.C., Rojec-Goldmann, G., Muehl, G.: QoS aggregation for Web service composition using workflow patterns. In: *8th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2004)*, Monterey, pp. 149–159 (2004)

# Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications

Thomas Kwok and Ajay Mohindra

IBM Research Division  
Thomas J. Watson Research Center  
19 Skyline Drive, Hawthorne, NY 10532  
{kwok, ajaym}@us.ibm.com

**Abstract.** Cost of customization, deployment and operation of a software application supporting multiple tenants can be lowered through multi-tenancy in a new application business model called Software as a Service (SaaS). However, there are a number of technical challenges that need to be tackled before these benefits can be realized. These challenges include calculations of resource requirements for multi-tenants with applied constraints in a shared application instance, the optimal placement of tenants and instances with maximum cost savings but without violating any requirements of service level agreements for all tenants in a set of servers. Moreover, previously reported capacity planning and resource allocation methods and tools are not tenant aware. This paper will address and provide novel solutions to these challenges. We also describe the first of a kind, a multi-tenant placement tool for application deployment in a distributed computing environment.

**Keywords:** capacity planning, resource allocation and management, tenant placement, constraint, multi-tenant, software as a service, SaaS.

## 1 Introduction

Recently, a new business model of software applications called Software as a Service (SaaS), which offers benefits of lower cost of customization, deployment and operation over the Internet has evolved [1-3]. In general, SaaS is associated with business software applications that are Web-based, deployed and operated as a hosted service accessed by users over the Internet. Multi-tenants in addition to multi-users support, installation of application on a managed Internet data center with remote management capability are a few characteristics of a multi-tenant SaaS application [4-8]. In the SaaS business model, the ownership of technology infrastructure and management responsibility of the application has moved to application service providers (ASPs) from tenants. The multi-tenant SaaS model benefits ASPs by reducing hosting costs as the same application is shared among multi-tenants. It also benefits tenants through eliminating their costs of owning and managing the infrastructure and applications. Tenants can gain immediate access to the latest information technology (IT) innovations and improvements provided by the

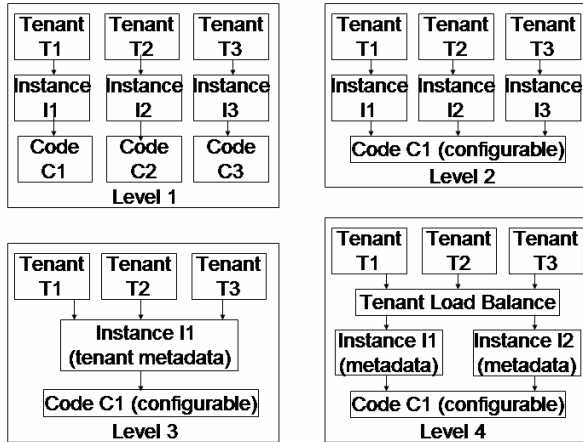


Fig. 1. Four levels of the multi-tenant support model in an application layer

ASP without spending their own IT budgets. In the SaaS business model, usages of the application can either be on a per user basis or pay as you go basis.

In a multi-tenant SaaS model, the multi-tenant support can be applied to four different software layers: application, middleware, virtual machine (VM) and operating system [4]. For the application layer, there are four levels of the multi-tenant support model as shown in Figure 1 [5]. Level 1 has a separate instance for each tenant’s customized code base and it is similar to the ASP model. Level 2 also has a separate instance for each tenant but all instances come from a single code base with configuration options. With a single application code base to support several tenants, the total deployment time is shorter. Level 3 has a single instance for all tenants with configurable metadata for each tenant. In this level, the updating of application features and functions are simpler and centralized because there is only one instance of a single source code base. Level 4 has a load-balanced farm of identical instances with configurable metadata for its tenants. There are many benefits of multi-tenant application deployment in Levels 3 and 4. The main benefits are the reduction of IT expenses, and cost savings in software license fees and hardware expenses by reducing the number of development, staging, training, disaster recovery servers. Other benefits are in deployment, provision, on-board and maintenance by reducing IT processes, such as server and application setup and configuration, and reducing support staffs in server and application tuning, backup, patch and upgrade. Costs in cooling and HVAC, power and lighting are reduced due to fewer servers [9].

However, several initial setup and configuration steps have to be carried out in order for the application to support multi-tenants in a SaaS operational structure [8]. There are also a number of challenges that require solutions before full benefits of multi-tenant application deployment can be realized. First, it is difficult to calculate resource requirements for each additional new tenant with a number of new users, and at the same time meeting constraints for all tenants in a shared application instance. Second, limiting factors or bottlenecks on computing resources required for multiple instances, each with multi-tenants having different constraints, have to be determined.

Third, an administrator needs the advice on the placement of a group of multi-tenant applications on a set of servers without violating any service level agreement (SLA) requirements of all tenants. Fourth, the placement of tenants and instances in a distributed computing environment has to be automated. Fifth, cost savings among different multi-tenant placements have to be compared and optimized even though there are many variables involved. Other challenges are multi-tenant data models, tenant isolation and security related issues. This paper will address and provide novel solutions to the first four challenges. We also describe the first of a kind, a placement tool for multi-tenant application deployment in the third level of the multi-tenant support model.

## 2 Prior Related Work

Capacity planning and resource allocations to satisfy application requirements, and resource constrained project scheduling are a generalization of the static job shop problem, and have been reviewed thoroughly by researchers [10-12]. Most of these studies are based on traditional exact methods, priority rule schedule [13] and meta-heuristic approach [14]. A priority rule schedule consists of two parts, a priority rule and a scheduling scheme. In the meta-heuristic approach, an activity list is usually first created. Then, a neighboring schedule is identified by changing the order of tasks in the list. A quality of service (QoS) based resource allocation model has also been used to make sure that different constraints of concurrently running applications are satisfied [15]. However, all these research reports are not tenant aware, and do not take into account characteristics of multi-tenant application with a single instance supporting multi-tenants. Calculations of computing resources, such as central processing units (CPU) and memory, required for an instance supporting multi-users are rather straight forward and simple [10]. Calculations of resources required for an instance supporting multi-tenants with multi-users in each tenant are new and complicated. Up to now, there is no reported multi-tenant resource data model that can be used to calculate resource requirements for multi-tenants in a shared instance. Again, calculations of the maximum number of users in an instance on any server of specific resources without violating any SLA requirements for a single tenant has been outlined [11]. Calculations of maximum numbers of users and tenants on a shared instance in any server of specific resources satisfying all constraints listed on SLAs of all tenants are new and complicated.

A hierarchical and extensible resource management system has been built to allow the execution of multiple scheduling paradigms concurrently [16]. A number of commercial software tools for capacity planning, resource allocation and performance analysis for multiple applications on a set of servers are also available [16,17]. However, these commercial tools do not apply to the placement of tenants and instances for multi-tenant SaaS applications. They primarily focus on the placement of applications to available servers based on physical resources. Since they do not know how much resource requirements for a shared instance with additional tenants and constraints, a new application instance is always created and deployed for each new tenant. Furthermore, most manual resource capacity estimates on servers that work satisfactorily are those that are oversized and thus more expensive [9].

### 3 Resource Calculations for Multi-users and Multi-tenants

An application can demand a number of computing resources, such as CPU, memory, storage disk and network bandwidth, from physical hardware of the host server in a distributed computing environment. There are several different characteristics of these computing resources. A hard disk is considered the primary permanent storage device. An application instance requires an initial amount of storage, such as initialization of tables. These tables are shared among tenants and users. Additional amount of storage is required to store data for each additional tenant or user, such as adding additional tables and/or rows in different tables. Thus, storage usage can be assumed to be load dependent and proportional to numbers of tenants or users. Similarly, significant amount of memory in dynamic random access memory (DRAM) is consumed by an instance even if there is no active tenant or user. There are paged and non-paged memory. Non-paged memory consists of a range of virtual addresses guaranteed to be in DRAM at all times, and paged memory can be swapped to slower system resources, such as hard disk. As a result, it is very difficult to accurately project memory usage based on the number of tenants and users in a shared instance. Above all, many applications cannot run when the system is out of memory. Thus, only the maximum memory usage can be assumed slightly dependent on the number of tenants and users. Hence, an estimated upper limit on memory usage is often used. In some advance VMs, each instance may be able to use multiple CPUs if the host hardware physically contains multiple CPUs. Unlike storage and memory, CPU usage with same clock speed and same amount of Levels 1, 2 and 3 static RAM (SRAM) caches can be assumed to be linearly proportional to the number of active tenants and users because the processing speed of a CPU depends on both clock speed and cache.

For practical reasons, CPU and storage are used to illustrate calculations of resource requirements in this paper. For an instance supporting multi-users, calculations are rather straight forward and simple [10]. Let  $r$  be the number of users, and  $C(r)$  and  $M(r)$  be the CPU and storage required by an instance with multi-users, respectively. Then,

$$\begin{aligned} C(r) &= f_{CU}(r) . \\ M(r) &= M_0 + f_{MU}(r) . \end{aligned} \tag{1}$$

where  $f_{CU}(r)$  and  $f_{MU}(r)$  are functions of  $r$ , assuming that the CPU instance is idle if there is no active user.  $M_0$  is a constant representing overhead storage used by the instance without any users. However, calculations of resources required for a shared instance supporting multi-tenants and multi-users are new and complicated. Let  $t$  be the number of tenants in a shared instance and  $n$  be the total number of users. Then,

$$\begin{aligned} C(n,t) &= f_{CU}(n) + f_{CT}(t) . \\ M(n,t) &= M_0 + f_{MU}(n) + f_{MT}(t) . \end{aligned} \tag{2}$$

where  $f_{CT}(t)$  and  $f_{MT}(t)$  are functions of  $t$ . These two functions are additional CPU and storage required to isolated tenants from each other in a shared instance. For a special case where there are two tenants  $t = 2$  and the number of users in the two tenants are both equal to  $r$  such that  $n = 2r$ . Let us compare resources required in this

special case deployed in two different computing environments. First, in two application instances, each with a tenant and  $r$  users, and from Equations (1):

$$\begin{aligned} 2C(r,1) &= 2C(r) = 2f_{CU}(r) . \\ 2M(r,1) &= 2M(r) = 2M_0 + 2f_{MU}(r) . \end{aligned} \quad (3)$$

Second, in one application instance with two tenants and  $r$  users in each tenant, and from Equations (2):

$$\begin{aligned} C(2r,2) &= f_{CU}(2r) + f_{CT}(2) . \\ M(2r,2) &= M_0 + f_{MU}(2r) + f_{MT}(2) . \end{aligned} \quad (4)$$

Assuming  $f_{CU}(r)$  and  $f_{MU}(r)$  are linearly proportional to  $r$ , taking the first order approximation:

$$\begin{aligned} f_{CU}(2r) &= 2f_{CU}(r) . \\ f_{MU}(2r) &= 2f_{MU}(r) . \end{aligned} \quad (5)$$

Since a certain amount of storage is shared by both tenants, and additional amount of storage required to isolate tenants from each other is relative small in a shared instance, then:

$$f_{MT}(2) \ll M_0 . \quad (6)$$

According to Equations (5) and (6):

$$\begin{aligned} f_{CU}(2r) + f_{CT}(2) &> 2f_{CU}(r) . \\ M_0 + f_{MU}(2r) + f_{MT}(2) &\ll 2M_0 + 2f_{MU}(r) . \end{aligned} \quad (7)$$

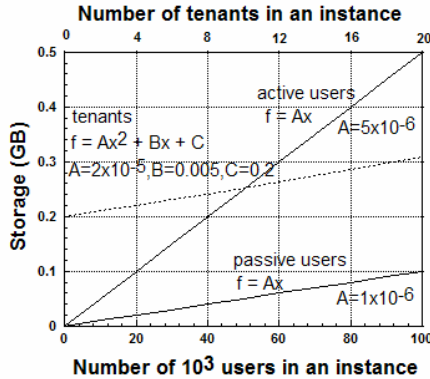
Thus from Equations (3), (4) and (7):

$$\begin{aligned} C(2r,1) &> 2C(r,1) . \\ M(2r,2) &\ll 2M(r,1) . \end{aligned} \quad (8)$$

As a result, there are relatively large savings in those resources shared by multi-tenants, such as storage and memory, but at the same time a little bit more usage of resources, such as CPU and network bandwidth, for deploying multi-tenants in a shared instance. Accordingly, many tenants should be deployed in a shared instance instead of only one tenant per instance in a server for a multi-tenant SaaS application.

## 4 Resource Data Models for Multi-tenants in a Shared Instance

Up to now, there is no reported computing resource data model for multi-tenants in a shared application instance. Functions  $f_{CU}(n)$ ,  $f_{MU}(n)$ ,  $f_{CT}(t)$  and  $f_{MT}(t)$  can be in the form of curves or tables of measured data. Equations based partially on theory and partially on these empirical data can be obtained by fitting these curves or tables with interpolation or extrapolation algorithms [18]. These semi-empirical equations can be linear, polynomial, power, exponential, logarithmic or any other types depending on fractions of different activities, such as Web, computation, transaction and database, involved in the application. Hypothetical data of storage requirements as a function of



**Fig. 2.** Hypothetical data of storage requirements as a function of active and passive users, and tenants

users and tenants in a shared instance are shown in Figure 2. Assuming that storage usage by each user is independent from other users, and from the total number of users, semi-empirical parameters based on the first order approximation are obtained by fitting solid curves in Figure 2. However, storage usage by each tenant may increase with the total number of tenants in the shared instance because additional storage is required to isolate each tenant from the increasing number of other tenants. Thus, semi-empirical parameters based on the second order approximation are obtained by fitting the dotted curve in Figure 2.

As shown in Figure 2, passive users also demand storage usage but their usage is much less than that of active users. This is also true for memory usage. Let  $x$  be the concurrent user or peak load rate of an application instance and  $y$  be the utilization rate of a server. Lowering the utilization rate below 1.0 will provide higher service reliability and increase uptime, which will eliminate or reduce fines caused by missed SLA requirements. Let  $u$  and  $p$  be total numbers of active and passive users in a shared instance, respectively. Thus,

$$\begin{aligned} u &= n * x . \\ p &= n * ( 1.0 - x ) . \end{aligned} \tag{9}$$

According to Equations (2) and (9), the total storage required by  $t$  tenants with total number of users  $n$  in a shared instance is given by

$$M(n,t) = ( f_{MU}(u) + f_{MU}(p) + M_0 + f_{MT}(t) ) / y . \tag{10}$$

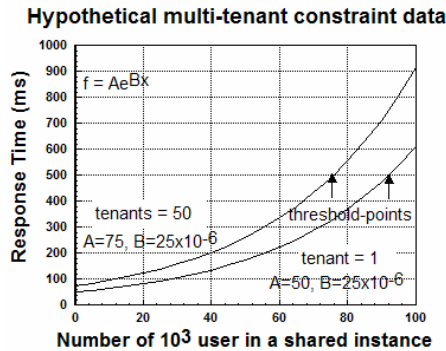
where  $f_{MU}(u)$  and  $f_{MU}(p)$  are obtained from two solid curves while  $M_0$  and  $f_{MT}(t)$  are obtained from the dotted curve. As shown in Figure 2,  $M_0$  is the intercept on the Y-axis when  $t = 0$  or  $1$  as an application instance either requires no tenant or a minimum of one tenant for initialization. Similarly, the total CPU required by  $t$  tenants with total number of users  $n$  in a shared instance is given by

$$C(n,t) = ( f_{CU}(u) + f_{CT}(t) ) / y . \tag{11}$$

assuming that the CPU instance is idle if there is no active user. Other computing resources, such as memory and network bandwidth, required by multi-tenants in a shared instance can be calculated in similar ways using either Equations (10) or (11).

## 5 Constraints on a Multi-tenant Application Instance

From previous two sections, calculated computing resources, such as CPU and storage, based on the number of users and tenants in a shared instance are the basic or minimum requirements of available resources in any server on which the shared instance would run. However, there are also a number of constraints on limiting the maximum number of users and tenants on this shared instance running in any server of specific resource. These constraints can be response time, availability, user arrival and page view rates, business transaction rate, request and transfer rates for database, input and output operational rates on file system, as well as disaster recovery, cost and reporting metrics, in SLA specifications. Operating the shared instance within these constraints will reduce or eliminate fines caused by missed SLA requirements.



**Fig. 3.** Hypothetical data of response time as functions of user and tenant numbers

Again, calculations of the maximum number of users on an instance in any server of specific resources without violating any SLA requirements have been outlined [11]. However, calculations of maximum numbers of users and tenants on a shared instance running in any server of specific resources satisfying all constraints listed on SLA specifications of all tenants are new and complicated. For practical reasons, response time is used to illustrate calculations of resource requirements with applied constraints in this paper. Hypothetical data of response time limiting the maximum number of users and tenants on a shared instance in any server of specific resources is shown in Figure 3. Semi-empirical parameters based on the exponential approximation are obtained by fitting these two curves. Let the constraint on response time listed on SLA specifications be 500 ms, then the maximum number of users allowed in an instance with only 1 tenant is around  $92 \times 10^3$  while that in a shared instance with 50 tenants is around  $75 \times 10^3$ . The maximum number of users on a shared instance with  $t$  tenants can be found by interpolation or extrapolation of these two curves [18]. The



maximum number of users and tenants allowed in a shared instance running on any server of specific resources for other constraints listed on SLA specifications of all tenants can be carried out in a similar way.

The algorithm for multi-tenant resource calculations with applied constraints is illustrated with block diagrams in Figure 4. There are three main parts of this algorithm. Let  $J$  be the total number of resource types and its index  $j$  is from  $1$  to  $J$ . For practical reasons, only two resource types ( $J=2$ ), CPU  $C(j=1)$  and storage  $M(j=2)$ , are used to demonstrate multi-tenant resource calculations in this paper. Other resource parameters, such as network bandwidth and memory, can be added into these calculations in similar ways. Let  $S$  be the total number of available servers and its index  $s$  is from  $1$  to  $S$ . In the first part, the multi-tenant active placement sensor will provide information on current resource usages of tenants  $T_i$  with users  $N_i$  in each  $i$  of shared instances  $I$  of applications  $A$ , as well as residual resources in available servers  $S$ . This first part is to calculate resource demands due to new tenants  $\Delta T_i$  with new users  $\Delta N_i$  on an active instance  $i$  of an application  $a$  in a specific server  $s$ . Let  $C_{0i}(N+\Delta N, T+\Delta T)$  and  $M_{0i}(N+\Delta N, T+\Delta T)$  represent two sets of resource demand vectors for minimum CPU and storage requirements due to additional tenants  $\Delta T$  and additional users  $\Delta N$  on an instance  $i$  of an application  $a$  in a server  $s$ . According to Equations (2),

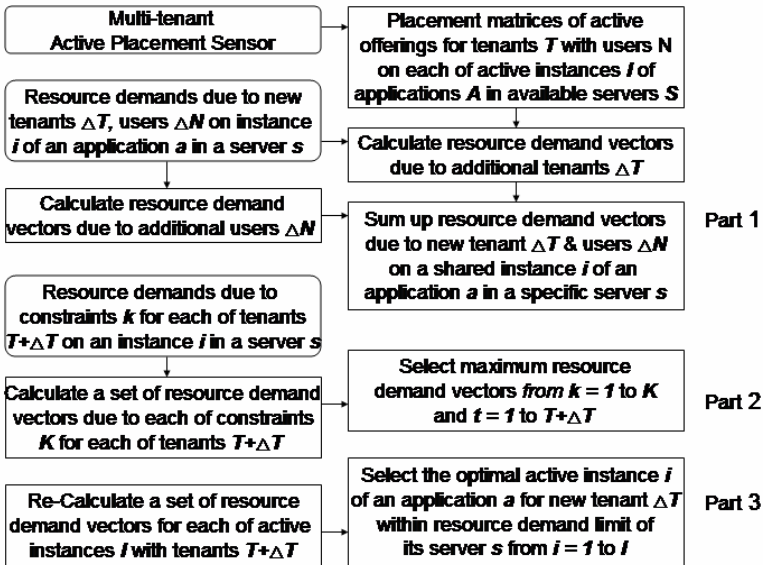


Fig. 4. An algorithm for multi-tenancy resource calculations with applied constraints

$$\begin{aligned}
 C_{0i}(N+\Delta N, T+\Delta T) &= C_{0i}(N+\Delta N) + C_{0i}(T+\Delta T); \\
 M_{0i}(N+\Delta N, T+\Delta T) &= M_{0i}(N+\Delta N) + M_0 + M_{0i}(T+\Delta T); \quad i \in I.
 \end{aligned}
 \tag{12}$$

The second part is to calculate resource demand vectors due to each of tenants  $T+\Delta T$  with users  $N+\Delta N$  in an active instance  $i$  with each  $k$  of a set of applied constraints  $K$ ,

such as response time and transaction rate. Then, the maximum resource demand vectors are selected from  $k = 1$  to  $K$  and  $t = 1$  to  $T + \Delta T$ :

$$\begin{aligned} C_{\max,i}(N+\Delta N, T+\Delta T) &= \text{Max}\{C_{0i}(N+\Delta N, T+\Delta T), C_{ki}(t)\}; \forall k \in K. \\ M_{\max,i}(N+\Delta N, T+\Delta T) &= \text{Max}\{M_{0i}(N+\Delta N, T+\Delta T), M_{ki}(t)\}; \forall t \in T+\Delta T. \end{aligned} \quad (13)$$

where  $C_{ki}(t)$  and  $M_{ki}(t)$  are resource demand vectors of CPU and storage due to constraint  $k$  on tenant  $t$  of instance  $i$ . The third part is to re-calculate a set of resource demand vectors for each  $i$  of shared instances  $I$  with tenants  $T+\Delta T$  and users  $N+\Delta N$ . Then, the resource demand for a server  $s$  is calculated by sum over  $i = 1$  to  $I$  and  $a = 1$  to  $A$  on each  $s$  of available servers  $S$ . The residual resource of a server  $s$  is given by Equations (14):

$$\begin{aligned} C_{\text{residual}}(s) &= C_{\text{initial}}(s) - \sum_{ia} C_{\max,i}(N+\Delta N, T+\Delta T); s \in S. \\ M_{\text{residual}}(s) &= M_{\text{initial}}(s) - \sum_{ia} M_{\max,i}(N+\Delta N, T+\Delta T); \forall i \in I, \forall a \in A. \end{aligned} \quad (14)$$

where  $C_{\text{initial}}(s)$  and  $M_{\text{initial}}(s)$  are the initial resource of CPU and storage in a server  $s$ , respectively.  $C_{\text{residual}}(s)$  and  $M_{\text{residual}}(s)$  are the residual resource in CPU and storage in a server  $s$ , respectively if additional tenants  $\Delta T_i$  and users  $\Delta N_i$  are deployed in its shared instance  $i$  of an application  $a$ . The initial source must meet or exceed the total resource demand of a server  $s$  for each  $j$  of all resource types  $J$ . The effective residual resource score  $E_{\text{residual}}(s)$  for all resource types  $J$  in a specific server  $s$  can then be calculated using several different methods. In our multi-tenant resource placement tool, a total score over all resource types  $J$  with their weighting factors  $w_j$  between  $0.0$  and  $1.0$  is used. From equations (14):

$$\begin{aligned} E_{\text{residual}}(s) &= w_{j=1} * C_{\text{residual}}(s) + w_{j=2} * M_{\text{residual}}(s) + \dots ; s \in S. \\ \sum_j w_j &= 1.0; \forall j \in J. \end{aligned} \quad (15)$$

Priority rules can be used to set weighting factors  $w_j$  of resource type  $j$  in the order of its importance and contributions to the effective residual resource score in the list of resource types  $J$ . Finally, specific server  $s^*$  with minimum effective residual resource score is selected for deployment of additional tenants  $\Delta T_i$  and users  $\Delta N_i$  in its shared instance  $i$  of an application  $a$ .

## 6 The Multi-tenant Placement Model

The placement of multiple applications in a set of available servers with optimization is illustrated in Case 1 of Figure 5. There are six available servers, namely S1, S2, S3, S4, S5 and S6 with different initial resources and five applications, namely A1, A2, A3, A4 and A5. Four instances of the same application A1, namely I1, I2, I3 and I4, have been deployed on S1, S2 and S4. Instances of applications A2, A3, A4 and A5 have also been deployed on S2, S3 and S5. The principle rule of optimization in the placement is to deploy a new instance on the server with the smallest residual resource left after meeting the resource requirement of this new instance. As a result, larger chunks of residual resource will be retained in other servers for later use by an

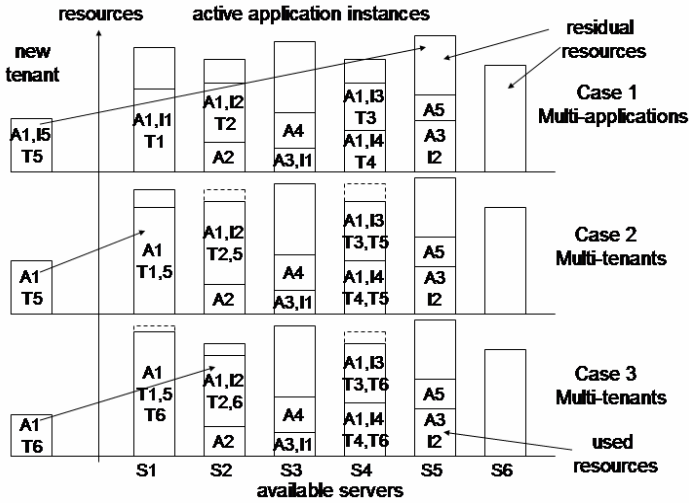


Fig. 5. Comparison of our new multi-tenant placement model with other previously reported placement models for multiple applications

application instance with a higher resource demand. First, let us assume that these application instances in Case 1 only support multi-users but not multi-tenants. Thus, a new instance I5 of A1 has to be created and deployed for a new tenant T5 even though there are existing instances I1, I2, I3 and I4 of the same application A1 running in S1, S2 and S4. Obviously, servers that have large enough residual resource to meet the resource requirement of I5 are S3, S5 and S6. With optimization, the traditional placement methods [12-14] or commercial products [16,17] will deploy I5 on S5 to leave larger chunks of residual resource on S3 and S6.

Now, let us assume that all these applications also support multi-tenants in addition to multi-users. Once again, traditional placement methods [12-14] or commercial products [16,17] with optimization will still deploy I5 on S5. However, the placement result using our new multi-tenant placement model is very different. As illustrated in Case 2, we may not need to create a new instance I5 of A1 for a new tenant T5 because there are existing instances I1, I2, I3 and I4 of the same application A1 running on S1, S2 and S4. First, we need to test whether the residual resource in one of servers S1, S2 and S4 would be large enough to meet the expanded resource requirement of I1 with an additional new tenant T5. As shown in Case 2, the expanded resource requirement of I1 with two tenants T1 and T5 will be within the resource limit on S1 while that of I2 with two tenants T2 and T5 will exceed the resource limit on S2. The expanded resource requirement of either I3 or I4 with two tenants T3 and T5 or T4 and T5 will also exceed the resource limit on S4. Obviously, our multi-tenant placement model with optimization will deploy the new tenant T5 into the instance I1 as the second tenant without creating another application instance I5. Case 3 illustrates the placement of another new tenant T6 for an application A1. Once again, the expanded resource requirement of I1 with three tenants T1, T5 and T6 will exceed the resource limit on S1 while that of I2 with two tenants T2 and T6 will be

within the resource limit on S2 this time because the resource requirement for T6 is smaller than that of T5. The expanded resource requirement of either I3 or I4 with two tenants T3 and T6 or T4 and T6 will also exceed the resource limit on S4. Instead of creating a new instance I6, this new tenant T6 for an application A1 will be placed on an existing instance I2 as the second tenant.

## 7 The Framework and Algorithm of a Multi-tenant Placement Tool

In an Internet data center, multiple SaaS offerings of application instances are active on shared physical resources, such as CPU and storage, of a set of computing servers in a distributed computing environment. When new tenants subscribe to a new SaaS offering, these new tenants need to be assigned to new or specific active instances under constraints due to SLA specifications of all tenants. Any server devoted to a new offering must have the required capacity of computing resource to host a new instance or an active instance with additional tenants and users without compromising SLA requirements of all tenants. Moreover, security restrictions on tenants in a shared instance cannot be violated. However, traditional application placement tools are not tenant aware [16,17]. Their approaches primarily focus on static or dynamic placement of applications to available servers based on their physical resources with or without load balance or rebalance. In these placement tools, a new application instance is always created and deployed for a new tenant. They cannot assign a new tenant into an active instance because they do not know how much extra resource requirements of an active instance with additional tenants and users.

An architectural framework of our multi-tenant placement tool is shown as block diagrams in Figure 6. This framework provides capabilities of multi-tenant resource calculations with applied constraints and the placement of tenants and instances for

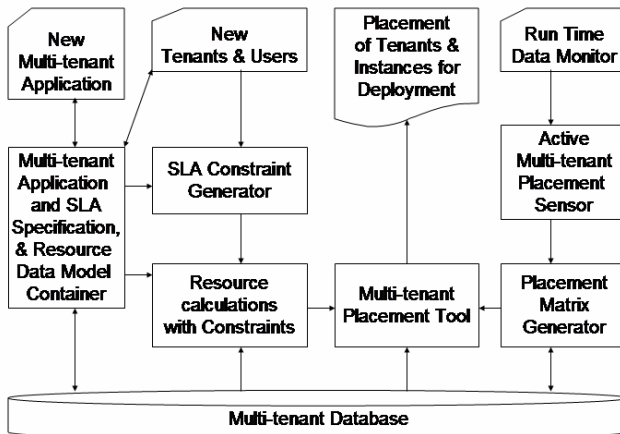


Fig. 6. An architecture framework of a multi-tenancy placement tool

multi-tenant application deployment. It consists of one output and three input modules, six essential functional modules and a multi-tenant database. The flow diagrams in Figure 6 also show the logical flows of information among modules and database. The “New Multi-tenant Application” module provides graphical user interface (GUI) and scripts for an administrator to input specifications and multi-tenant data models of a new software application or to modify existing ones. The “New Tenants and Users” module provides GUI for an administrator to enter numbers of new tenants and users in each tenant, and select the application required for deployment. The administrator also enters the SLA specification for each new tenant. The “Multi-tenant Application and SLA specification, & Data Model Container” module holds, stores, retrieves and deliveries these multi-tenant data from and to other modules and the multi-tenant database. The “Run Time Data Monitor” module constantly monitors and collects resource usage profile of each active instance in each server. It also provides information on performance parameters and utilization rate of each server. The “Multi-tenant Active Placement Sensor” calculates resource usages of each active instance and residual resource of each server. The “Placement Matrix Generator” constructs and stores resource usage and residual matrices. The dimension of these matrices is two with  $J \times I$ , where  $J$  is the number of resource types and  $I$  is the number of instances in a server. The initial resource matrix  $O_{initial}(j,s)$  and residual resource matrix  $O_{residual}(j,s)$  of resource type  $j$  and server  $s$  are then constructed based on information from the active placement sensor. They are used in Equations 15 of Section 5. The “SLA Constraint Generator” module constructs and stores constraints due to SLA requirements for new tenants, and retrieves constraints for active tenants on a shared instance in a specific server  $s$ . The “Resource Calculations with Constraints” module

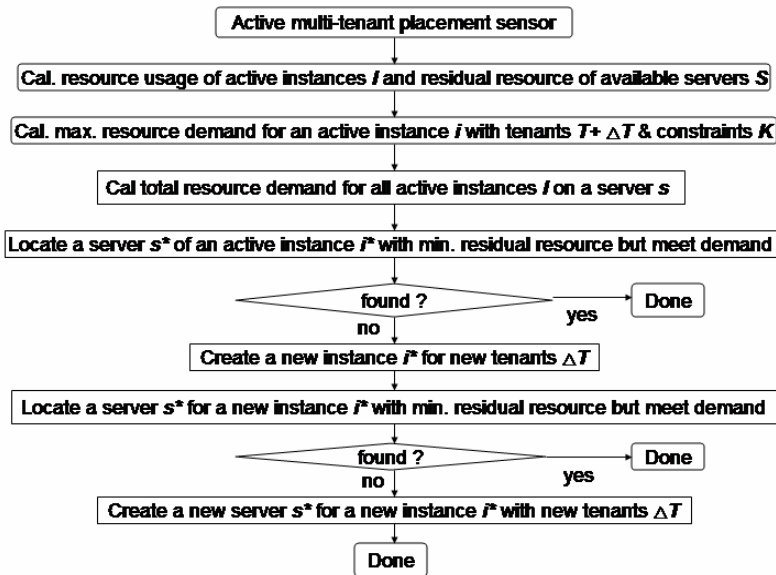


Fig. 7. A multi-tenancy placement algorithm

calculates required physical resources, such as CPU, memory, storage and network bandwidth, with applied constraints for new and active tenants in a shared instance on a specific server as described in Sections 3, 4 and 5. This is to make sure that SLA requirements are met for all tenants in a shared instance. Moreover, cross tenant security restrictions are not violated, such as prohibition of tenant T1 and T2 deployed in the same application instance I1 and/or on the same server S1.

The “Multi-tenant Placement Tool” module constructs a package for placements of new tenants on specific instances and/or new instances on specific servers for multi-tenant application deployment according to our proposed multi-tenant placement model described in Section 6. The flow chart of a multi-tenant placement algorithm is shown in Figure 7. First, resource usage of all instances  $I$ , initial and residual resources of all available servers  $S$  are calculated based on information from the active placement sensor. Second, the maximum resource demand for each shared instance  $i$  with tenants  $T+\Delta T$  and constraints  $K$  are calculated as described in Section 5. Third, the total resource demand for each server  $s$  with its shared instance  $i$  of tenants  $T+\Delta T$  and its other instances  $I-I$  are calculated. Fourth, a particular server  $s^*$  with its particular instance  $i^*$  of tenants  $T+\Delta T$  is located because its residual resource is the minimum among all servers  $s$ , and within its resource limit. Fifth, a new instance  $i^*$  is created for new tenants  $\Delta T$  if a particular  $s^*$  is not found. Sixth, a particular server  $s^*$  with the new instance  $i^*$  is located because its residual resource is the minimum among all servers  $s$  with the same new instance  $i^*$ , and within its resource limit. Finally, a new instance  $i^*$  is created in a new server  $s^*$  for new tenants  $\Delta T$  if a particular  $s^*$  is still not found.

## 8 Implementation and Industrial Experiences

Most features and functions of the multi-tenant placement tool described in this paper have been implemented in Java. A generic sorting algorithm has been used to match the demand list from high to low with the residual source list from low to high. The residual source list is sorted once again after each match or placement. Optimizations based on the placement of one, two or three tenants at a time have been investigated. This multi-tenant placement tool is being integrated within the provisioning subsystem of an IBM internal project. Since the tool is integrated in a fully automated end-to-end provisioning, it saves time for administrators, increases their efficiency and productivity in managing the placement of tenants and instances for multi-tenant application deployment by simplifying and automating the placement processes. Its performance with two applied constraints, CPU and storage, as a function of new tenants or servers is shown in Figure 8. The number of new tenants equals the number of servers while the number of active tenants equals to half the number of servers. The CPU spent on the placement algorithm is found to depend on the number of new and active tenants, and the number of servers. Our preliminary data based just on one set of data has indicated that the CPU spent on the placement increases linearly with the number of new tenants or servers up to 100 tenants or servers. Results from more data sets will be collected in the future. Our preliminary results have confirmed the stability and usefulness of this multi-tenant placement tool. This tool has shown to minimize the number of servers deployed, resulting in maximum cost savings, in a

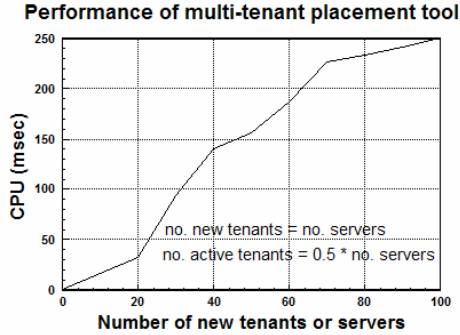


Fig. 8. The performance of multi-tenant placement tool

distributed computing environment. It has also shown to meet constraints of all tenants, and provide higher service reliability and increase uptime.

For our industrial experiences, we have found that it is very time consuming and tedious to measure tenant or user specific resource data for a new multi-tenant application. It will be useful if we can calculate new resource data based on available resource data of other active multi-tenant applications using their activity correlation functions. We have also found that it is hard to accurate project memory requirement for multiple tenants in a shared instance. As a result, an estimated upper limit on memory requirement is often used. Our preliminary results have revealed that several crucial constraints, such as response time and transaction rate, always play important roles in determining multi-tenant resource requirements. They have also indicated that limiting factors or bottlenecks in multi-tenant applications depend on its computing activities, such as Web, transaction, computing and database. Moreover, users of this multi-tenant placement tool have request additional functions, such as adding an additional placement rule to assign each server with a minimum load, merging several instances of the same application in a server into one, and migrating tenants among instances. They would also like to use this placement tool to tell whether a specific application would benefit from the multi-tenant deployment.

## 9 Conclusion and Discussion

In this paper, we have outlined calculations of resource requirements for multiple tenants in a shared application instance with applied constraints using our hypothetical multi-tenant resource data models. We have also described novel methods for the optimal placement of tenants and instances based on our proposed multi-tenant placement model without violating any SLA requirements of all tenants in a set of servers. We have architected and implemented the first of a kind, a multi-tenant placement tool for application deployment using a minimum number of servers, and thus with maximum cost savings in a distributed computing environment. However, other challenges, such as database security and data isolation, tenant view filter and data encryption, remain un-tackled. In the future, these challenges should be addressed with proved solutions.

Characteristics of multi-tenancy in four different software layers: application, middleware, VM and operating system, are important factors in studying and understanding limiting factors or bottlenecks in multi-tenant SaaS applications. Based on our intuitions, the resource sharing and cost savings are relatively high for multiple tenants in a shared instance while the security isolation is high and performance impact is low for multiple instances or VMs per server. In the future, this multi-tenant placement tool can be used to verify these multi-tenant characteristics once multi-tenant resource usage data on these four different layers have been measured.

**Acknowledgments.** The authors would like to thank A. Karve for his work on the integration of this multi-tenant placement tool with other IBM internal projects. The authors would also like to thank J. Batstone for her support.

## References

1. Iod: *Sotware as a Service*, Director Publications Ltd., London (2002)
2. Iyar, S.: *Why Buy the Cow*, Santa Clara (2007)
3. Kobilsky, N.: *SAP CRM on-demand*, SAP Forum (2006)
4. Gianforte, G.: *Multiple-Tenancy Hosted Applications: The Death and Rebirth of the Software Industry*. RightNow Technologies Inc. (2005), <http://www.rightnow.com>
5. Chong, F., Gianpaolo, C., Wolter, R.: *Multi-Tenant Data Architecture*, Microsoft Corporation (2006), <http://www.msdn2.microsoft.com/>
6. Fisher, S.: *The Architecture of the Apex Platform*, salesforce.com's Platform for Building On-Demand Applications. In: Proc. of the 29th IEEE Int'l Conference on Software Engineering, p. 3. IEEE Press, New York (2007)
7. Guo, J.G., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: *A Framework for Native Multi-Tenancy Application Development and Management*. In: Proc. of the 9th IEEE Int'l Conference on E-Commerce Technology, pp. 551–558. IEEE Press, New York (2007)
8. Kwok, T., Nguyen, T., Lam, L.: *A Software as a Service with Multi-Tenancy Support for an Electronic Contract Application*. In: Proc. of IEEE Int'l Conference on Services Computing, pp. 28–33. IEEE Press, New York (2008)
9. Mendoza, A.: *Utility Computing Technologies, Standards, and Strategies*. Artech House Publishers, Norwood (2007)
10. Herroelen, W., Reyck, B.D., Demeulemeester, E.: *Resource-Constrained Project Scheduling: A Survey of Recent Developments*. *Computers and Operations Research* 25(4), 279–302 (1998)
11. Brucker, P., Drexel, A., Mohring, R., Neumann, K., Pesch, E.: *Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods*. *European Journal of Operational Research* 112, 3–41 (1999)
12. Hartmann, S., Kolisch, R.: *Experimental Evaluation of State-of-the-Art Heuristics for Resource-Constrained Project Scheduling Problem*. *European Journal of Operational Research* 127, 394–407 (2000)
13. Kolisch, R.: *Efficient Priority Rules for the Resource-Constrained Project Scheduling Problem*. *Journal of Operations Management* 14, 179–192 (1996)
14. Bouleimen, K., Lecocq, H.: *A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem and its Multiple Mode Version*. *European Journal of Operational Research* 149, 268–281 (2003)



15. Rajkumar, R., Lee, C., Lehoczky, J., Siewiorek, D.: A Resource Allocation Model for QoS Management. In: Proc. of the 18th IEEE Real-Time Systems Symposium, pp. 298–307. IEEE Press, New York (1997)
16. Islam, N., Prodromidis, A., Squillante, M., Fong, L., Gopal, A.: Extensible Resource Management for Cluster Computing. In: Proc. of the 17th Int'l Conference on Distributed Computing Systems, pp. 561–568. IEEE Press, New York (1997)
17. Bagchi, S., Hung, E., Iyengar, A., Vogl, N., Wadia, N.: Capacity Planning Tools for Web and Grid Environments. In: Proc. of the 1st Int'l Conference on Performance Evaluation Methodologies and Tools, pp. 25–34. ACM Press, New York (2006)
18. Bhatti, M.A.: Practical Optimization Methods. Springer, New York (2000)

# SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment

Xin Hui Li, Tian Cheng Liu, Ying Li, and Ying Chen

IBM China Research Lab, Software Park  
Beijing 100193, China  
{lixinhui, liutc, lying, yingchen}@cn.ibm.com

**Abstract.** The flourish of SaaS brings about a pressing requirement for Multi-tenancy to avoid dedicated installation for each tenant and benefit from reduced service delivery costs. Multi-tenancy's intention is to satisfy requests from different tenants concurrently by a single service instance over shared hosting resources. However, extensive resource sharing easily causes inter-tenant performance interference. Therefore, Performance isolation is crucial for Multi-tenancy environment to prevent the potentially bad behaviors of one tenant from adversely affecting the performance of others in an unpredictable manner and prevent the unbalanced situation where some tenants achieve very high performance at the cost of others. Current technologies fail to achieve the goals of both performance isolation and resource share. This paper proposes a Service Performance Isolation Infrastructure (SPIN) which allows extensive resource sharing on hosting systems. Once some aggressive tenants interfere with others' performance, SPIN gives anomaly report, identifies the aggressive tenants, and enables a self-adaptive moderation to remove their negative impacts on others. We have implemented SPIN prototype and demonstrate its isolation efficiency on the Trade6 benchmark which is revised to support Multi-tenancy. SPIN fits industry practice for a performance overhead less than 5%.

**Keywords:** Multi-tenancy, performance monitoring, resource accounting and management, byte code instrumentation.

## 1 Introduction

Software-as-a-service (SaaS) [1] permits customers to consume software applications in a hosting mode as an emerging software delivery model with the capabilities of lowering total cost of ownership, fast enablement, and seamless scale-up per business needed, especially by Small and Medium Businesses (SMB). SaaS is typically associated with "multi-tenant architecture", which is a prerequisite for a SaaS application [2, 3]. Traditionally, there would be only one instance of an application running on a server, and this instance would only serve one customer, organization, or company (tenant). In the SaaS world, giving each tenant a dedicated server is a huge waste of resources and service providers want to put as many tenants on the same server as possible. Multi-tenancy aims to enable a service environment that user requests from different tenants are served concurrently by the least amount of hosted service

instances running on the shared hardware and software infrastructure. It requires deployment of a much smaller infrastructure, in contrast to having a dedicated installation for each tenant, which bring in a number of benefits including improved profit margin for service provider through reduced delivery costs and decreased service subscription costs for clients.

Multi-tenancy has two different maturity patterns: the first pattern supports each tenant with its dedicated service instance over a shared hardware, Operating System (OS) or a middleware server in a hosting environment whereas the second pattern can support all tenants by a single service instance over shared hosting resources. The second pattern is more consistent with the intention of Multi-tenancy. In the environment of pattern two, the tenant would naturally desire to access and use the service as if there were dedicated ones. However, extensive resource sharing easily causes inter-tenant interference on performance. To evolve from pattern one to pattern two and achieve more efficient Multi-tenancy hosting, performance isolation is crucial to prevent the potentially bad behaviors of one tenant from adversely affecting the performance of others in an unpredictable manner and prevent the unbalanced situation where some tenants achieve very high performance at the cost of others.

At present, virtualization technology is used to enable the isolation needed by Multi-tenancy and isolation management [4, 5, 6]. Some directly depend on Virtual Machines to create service hosting environments that provide logical boundaries between tenants. Although these works can help to adapt current software and hardware for Multi-tenancy with the least cost, virtualization technology belongs to the pattern one and is not able to cater for the demands of Multi-tenancy. They restrict resource sharing between different tenants and cause additional management cost.

There are improvements on the original adoption of virtual machine to reduce the amount of exclusive and dedicated computing resource. SWSOft's Virtuozzo product [7] is an example of that technology. This architecture allows service partitions to be created and configured differently from one another. Although it is tremendously valuable to SaaS hosters for optimizing their machine allocation, especially for these ISVs having the same system requirements, virtual service partition in general is not an effective method for Multi-tenancy, since it needs a different service instance for each partition and does not support multi-tenants to share all resources of hosting platform.

Normally, hosting systems have sufficient resources to meet basic requirements of tenants, but they can not provide enough resources to meet everything that every tenant might want with variations on the workload from some aggressive tenants. The performance experienced by the workload from a given tenant suffers from such resource unavailability. We call this situation as Instable state where the hosting systems have acted their processing capability to the full extent, even exhaustingly to crash, but can not satisfy requests of every tenant. With this in mind, we advocate a service performance isolation infrastructure (SPIN) which achieves efficient isolation and extensive resource sharing simultaneously. SPIN makes anomaly detection for instable state, identifies aggressive tenants, and enables the multi-tenancy system to self-adaptively remove negative impacts of the aggressive tenants on others. SPIN has been implemented as a Plug-in, independent of server and service implementations. The practice of SPIN in the revised Trade6 [8] demonstrates its accuracy of anomaly report, effectiveness of performance isolation and little perturbation to the running

service (less than 5% performance overhead). The implementation and experiment of SPIN is made based on Web service, a specific kind of popular service.

The rest of this paper proceeds as follows. Section 2 studies the requirements on performance isolation infrastructure and design of SPIN. Section 3 describes our prototype implementation. Section 4 evaluates the effect of SPIN with business service benchmark. Finally, Sections 5 and 6 discuss related work and conclusions.

## 2 Service Performance Isolation Infrastructure (SPIN)

In this section, we first outline the requirements of performance isolation in the complex multi-tenant environments. We then examine how to meet these requirements in our design.

**Isolation.** The infrastructure should prevent aggressive tenants from interfering with others. Request from different tenant will trigger different execution in the backend modules because the instances of these modules are shared among all the tenants sharing the same suite of resources. For this propose, it is necessary to account resource usage for different tenants during their access the shared modules to understand the performance factors of hosting platform. It is not permitted for some aggressive tenants to encroach resources and cause others' performance decreasing.

**Efficiency.** The infrastructure should maximize the overall utilization of resources on hosting platform, which is the intention to induce Multi-tenancy. This might be achieved by placing a loose upper bound on resource usage of different tenants. Pre-preservation of resources [4, 9, 10] can not satisfy this point. This goal is motivated by the fact that systems are likely to have sufficient resources to meet basic requirements of tenants, but they probably do not have sufficient resources to meet everything that tenants might want.

**Self-adaptability.** The infrastructure should not require user intervention or manual tuning. This goal is motivated by the large number of tenants which a hoster may serves as well as the wide range of services and system configurations likely to be involved.

To meet the above requirements, three principal functionalities, performance anomaly detection, system monitoring and adaptation decision, are provided in SPIN:

The **anomaly detection functionality** of SPIN is responsible for signaling the occurrence of instable status in the execution environment. The anomaly detection facility identifies and analyzes any significant variations happening on the performance metrics, especially the variations those might potentially affect the system's behavior in the immediate future. In this way, the infrastructure can preemptively adapt the system to prevent predicted performance problems from actually occurring.

Although some work [11,12,13,14] has been made on the performance of service system in the passed years, there are primarily two points preventing these technologies applied in our anomaly detection. One point is their dependency on threshold. It is difficult to give threshold values in a deterministic and automatic way since the presence of Multi-tenancy brings out the complexity and randomness into present service system. The other point is that they pay no attention on the prediction of

instability. In practice, it is more important to predict and avoid problems on resource usage in the near future than to remove the negative impact already caused.

The *monitoring functionality* is responsible for collecting runtime data from the service components and their execution environment. On one hand, the data are used for detecting performance anomalies in service hosting platform. On the other hand, resource consumptions of each tenant must be accounted to identify which tenant is aggressive and whether the system serves that tenant with effective resource usage.

Under the complex Multi-tenancy environment, existing resource accounting technologies are impracticable. They [15,16,17,18] uses processes or threads as the accounting unit, while the real-life service applications run with multiple threads' concurrent execution. An individual thread may traverse various modules of services in the system. Further more, in the widely used thread pool one thread at one time serves one service and it will serve another service soon. And one instance of service serves several tenants and its execution is unaware to underlying thread or process.

The *adaptation decision functionality* of the infrastructure applies optimal solutions to those detected or predicted performance problems. If the state is instable, we identify the aggressive tenants from others based on their abnormal resources usage. Optimal moderation policy will be adopted automatically to isolate the negative effect and prevent the interference with other tenants.

SPIN starts System monitor and anomaly detection from the beginning. Anomaly detection is made on the data got by system monitor and reports anomaly state of the single service instance near before instability's happening. Simultaneously, the resource consumptions on the hosting system are also monitored and accounted on each tenant's behalf during their service usage. Once anomaly report is given, the resource consumption trend of different tenants is analyzed to identify which tenant's behavior causes the instability. Then, adaptive decision function is activated and moderation policy is executed automatically on the identified tenant to remove its negative effect on the performance of other tenants. These main functionalities are discussed in details over the following sections.

## 2.1 Anomaly Detection Model

For SPIN, we propose a new model to achieve sensitive anomaly detection. Wallace has proven in his paper [4] that the relationship between the mean arrival rate of service requests and the mean service rate is proven tightly correlated with the stability of system. If the difference between the arrival rate and service rate is positive, the system is stable and the request arrivals do not beyond the processing capability of the system; otherwise, the system is instable. In this model, the value of difference is used to execute the anomaly detection instead of the dependency on any threshold. Besides Queueing Theory, the model adopts a combination of Discrete Wavelet Transform (DWT) multi-resolution analysis [19] and Autoregressive (AR) model [20] to make a prediction on the difference value. It is named with WAQ accordingly. In this way, instability of service system in the short term is predictable, which provides convenience for system moderation and is favorable to health maintenance. Followings are the detailed calculation process.

For model WAQ, the following data items are got by monitoring as the inputs:

$$(c_i, t_i) \quad i = 0, 1, 2, \dots, k \tag{a}$$

Where  $c_i$  represents the number of requests that enter the service system at sampling time  $t_i$ ,

$$(b_i, t_i) \quad i = 0, 1, 2, \dots, k \tag{b}$$

Where  $b_i$  represents the number of requests that have been processed at sampling time  $t_i$ ,

From the above data items (a) and (b), we can derive the pure increasing rate by Equation (1):

$$x_i = \frac{c_i - b_i}{t_i - t_{i-1}}, i = 1, 2, 3, \dots, k \tag{1}$$

Let series  $\{X_i\} i = 1, 2, \dots, k$  be the input of prediction model. In the prediction model, the original discrete series of pure increasing rate is firstly decomposed into approximate series and several detail series. The result of single branch reconstruction of each decomposed series is more unitary than the original series in frequency, and it can be easy to predict by autoregressive method. At last, the prediction value of increasing rate can be obtained by synthesis of each reconstructed series' prediction result [20]. The prediction process works as follows:

Firstly,  $\alpha_{j,m}$  and  $d_{j,m} (1 \leq m \leq k)$  are got respectively by Mallat Algorithm [21] as the approximate series and detail series at resolution level  $j$ . In this way, the original series  $\{X_i\} i = 1, 2, \dots, k$  is transformed into a set of stationary series and AR model is a powerful tool for prediction of such series [22]. The predicting expressions can be expressed as Equation (2) and (3):

$$a_{j,k+1} = \sum_{i=1}^p \phi_i a_{j,k-i+1} \quad j = 1, 2, \dots \tag{2}$$

$$d_{j,k+1} = \sum_{i=1}^p \phi_i d_{j,k-i+1} \quad j = 1, 2, \dots \tag{3}$$

Where  $\alpha_{j,k+1}$  and  $d_{j,k+1}$  are the prediction values of approximate series  $\alpha_{j,m}$  and detail series  $d_{j,m}$  at resolution level  $j$  respectively.  $\phi_i$  is the corresponding coefficients of AR(p). Then,  $\tilde{\alpha}_{j,k+1}$  and  $\tilde{d}_{j,k+1} (i = 1, 2, \dots, j)$  are reconstructed respectively from  $\alpha_{j,k+1}$  and  $d_{j,k+1}$  by Mallat Algorithm.

As Equation (4) presents,  $\tilde{x}_{k+1}$  is derived as the prediction value of time  $t_{k+1}$  from the original increasing rate series  $\{X_i\}$ .

$$\tilde{x}_{k+1} = \tilde{d}_{1,k+1} + \tilde{d}_{2,k+1} + \dots + \tilde{d}_{j,k+1} + \tilde{a}_{j,k+1} \tag{4}$$

If  $\tilde{x}_{k+1} > 0$ , the system will be instable at the time  $t_{k+1}$  and the anomaly report is sent out.

## 2.2 System Monitoring

System Monitoring function of SPIN gets inputs to feed Anomaly Detection Model and accounts resource usage on behalf of each tenant. When Anomaly Detection Model triggers an anomaly report, aggressive tenants will be identified according to the characteristics of their resource consumption.

Model inputs are got by accounting how many requests come to the server and are processed every sampling interval. Following sections introduce the design of SPIN on resource accounting for each tenant.

### 2.2.1 Resource Consumption Accounting on Behalf of Tenants

In SPIN, we adopt a new mechanism to monitor resource usage within the service execution and allow the proper assignment of resource usage to tenants. The design of monitoring mechanism is guided by two key constraints. The first is that the monitor function must keep active during the normal service execution to provide the ability of resource consumption tracking and therefore have minimal discernible impact to the service's runtime performance. As for the resources we focus, CPU and memory are our primary focus. The second constraint originates from the fact that hoster platform is dynamically deployed with object code service applications. This means that there is no opportunity to use the source code to implement the function of monitoring. Meanwhile, we do not make any modification on service container regarding the applicability of SPIN in practice.

The whole monitor is event based. Two kinds of events are triggered and listened: one is triggered by entry or exit of a service boundary, the other event is produced when some resource is consumed, like the allocation of memory.

For every boundary change event, present executing service is put into a stack, named accounting stack, for resource accounting. The accounting stack is designed to easily find present running tenant on which resource is accounted. Otherwise we have to get the whole stack of thread and walk it down to find the present executing service and tenant. Massive performance overhead will be induced into service running to frequently get stack data and walk stack.

For every resource consumption event caught, the top unit is gotten from the accounting stack as the owner of the resource consumption. In other words, the accounting stack is active at the time of resource being consumed and used as the context within which to determine accountability. Section 3 discusses the specifics of the implementation of accounting stack and accounting process.

### 2.2.2 Aggressive Tenants Identification

After the anomaly report, it is needed to identify aggressive tenants who are apt to consuming resources faster than others. They consume resource with a trend of growth, even snatch resource from other tenants and cause the degradation of others. The identification is made offline for performance consideration.

For each tenant, all the accounting events are recorded. Based on these events, the time series of each tenant's resource consumption is provided to users. Because the

absolute volume of resource consumption of different tenants can fall in different scale ranges, the percentage of each tenant's consumption in all is calculated and used to execute clearly comparison among all tenants.

We rank tenants according to the ratio of resource volumes between two consecutive accounting events (For convenience of description, the events are presented by  $t_i$  and  $t_{i-1}$ ,  $i \geq 1$ , and the volumes of the two events are presented by  $V_{t_i}$  and  $V_{t_{i-1}}$ ), find the tenants with maximum ranks and report the tenant as aggressive ones. Because resource usage can be different from one request to another, the ratio value of one tenant may fluctuate up and down. We use a decay factor  $f$ , where  $0 < f < 1$  to adjust for the jitter. We consider only those tenants whose volumes satisfy  $V_{t_i} > (1-f) * V_{t_{i-1}}$  on consecutive accounting events as potential candidates. The decay factor keeps the ratio value of tenants that shrink a little in this accounting time, but which may ultimately be growing. We find that the decay factor is increasingly important as the size of the resource accounts decreases. Choosing the decay factor balances between too much information and not enough.

To rank tenants, we firstly calculate the growth factor ( $G$ ) of each accounting event as  $G_{t_i} = P_{t_i} * (R - 1)$ , where  $P$  is the number of accounting event ( $t_i$ ) that  $V_{t_i}$  has been potentially growing and  $R$  is the ratio of  $V_{t_i}$  at this event and  $V_{t_{i-1}}$  at the previous event such that  $R > 1$ , since  $R > 1$ ,  $G > 0$ . Each tenant's rank  $R_{t_i}$  is calculated by accumulating the growth factors  $G$  over several accounting events such that absolute growth is rewarded ( $R_{t_i} = R_{t_i} + G_{t_i}$ ) and decay is penalized ( $R_{t_i} = R_{t_i} - |G_{t_i}|$ ). Higher ranks represent a higher likelihood that the corresponding volume of the tenant grows aggressively. Since we only report tenants that have been potentially growing for some minimum number of events, we do not report the tenant related with a rise appearing firstly in a series.

### 2.3 Adaptation Decision

Design of moderation policy is not the focus of this paper, but SPIN indeed provides an open infrastructure to adopt freely policy for the moderation of service behavior to requests from aggressive tenants. It helps to set and enforce effectively proper policy to limit or isolate the negative effect of aggressive tenants on others.

## 3 Implementation of SPIN

In this section, the challenges met during our implementation of System Monitoring are described considering the goals of not modifying source code and providing a runtime monitoring with low performance overhead. Implementations of the other two functions focus on the algorithms in Section 2 and are omitted here to save space. System Monitor function is executed by two main phases, the instrumentation phase and the data collection phase. The instrumentation phase consists of bytecode instrumentation of the services and operations to be monitored. The data collection phase consists of running the program, gathering resource usage data, and accounting the consumption on proper tenants. They are respectively introduced in the following two sections.



### 3.1 Instrumentation to Maintain Accounting Stack

The Accounting Stack is implemented through the instrumentation of all service interface methods' entry and exit points with specific method calls. For every tenant, an accounting stack is built at the first time this tenant sends request for service usage. In the instrumentation phase, bytecodes of the services to be monitored is manipulated to insert the methods that maintain the Accounting Stack. On a Web service's entry a stack frame containing boundary information is pushed onto the Accounting Stack. On a Web service's exit a stack frame is popped from the stack. For every resource consumption event, resource consumed is accounted on owner of present accounting stack. In this way, a complete record is kept on the service chain accessed by this tenant and resource usage during the service time. It should be noted that we perform the instrumentation integrated with service lifecycle as a part of service deploy process. An array of statically allocated stack elements is employed here to avoid dynamic memory allocation and de-allocation during the push and pop operations. This help to address the efficiency concerns.

### 3.2 Data Collection by Resource Consumption Agent

Accounting Stack calculates the accounting unit. Meanwhile, determining resource consumption and billing the current accounting unit are the responsibilities of Resource Consumption Agent. We have built prototypes for two important resources, CPU and the Java heap.

The center of Resource consumption agent is a native agent with architecture presented in Figure 1. Two trackers are built to collect CPU usage and Memory usage respectively. CPU Tracker probes CPU for calculation of cycles by sampling. Every a sampling interval, the consumption of CPU in this interval are accumulated and accounted on the Current Accounting Tenant (CAT) which is reserved in Accounting

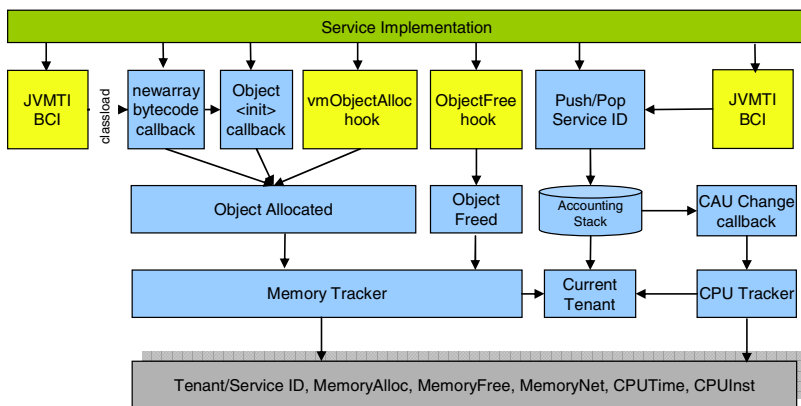


Fig. 1. Java Native Agent

Stack. Similarly, Memory tracker tracks memory allocation and de-allocation. We implement trackers by a Java Virtual Machine Tool Interface (JVMTI) agent. JVM TI [23] provides a set of standard interfaces for tracking object lifecycles and the state of JVM. A JVM TI agent can be notified of interesting occurrences through events and can control the service application through many functions, either in response to events or independent of them.

Special attention should be paid to the accounting of reusable resource, like memory allocated on Java Heap. The memory allocated will be reclaimed if it is no longer used. Each time an object is allocated, the object is also tagged with the ID of the accounting tenant. Tracking object deallocation is relatively simpler: JVM TI provides an event callback, i.e. *ObjectFree*, which notifies us every time an object is freed. On an object free event, we retrieve the tag of the freed object to determine the accounting tenant that was charged for this object. The memory consumption of that tenant is decreased accordingly.

Another point needs attention is the JNI cost caused by Accounting Stack which must communicate any change in the CAT to the Resource Consumption Agent. While our Agent runs in native space and as such the CAT must be made available in native space. To reduce the cost, we have used *java.nio.ByteBuffer*, whose instance is allocated outside the garbage-collected heap and can be accessed from both Java and native code. The use of *ByteBuffer* has been proved much more efficient to copy the CAU into native space than a Java native method does. On each object allocation, the Resource Accounting Agent retrieves the current accounting tenant from the *ByteBuffer* and charges it for the allocation.

## 4 Experiments and Evaluation

IBM Trade6 works as a performance benchmark and Web service sample application by providing a real-world workload, enabling performance research and verification test of the service Platform. It models an electronic stock brokerage providing Web services-based online securities trading. Routine stock operations, such as selling, watching holdings, and so on, are encapsulated into Web services and accessed by client at the runtime. We have revised Trade 6 to enable Multi-tenancy and adopt it in our experiments to SPIN's effectiveness on performance isolation and measure overhead of SPIN implementation.

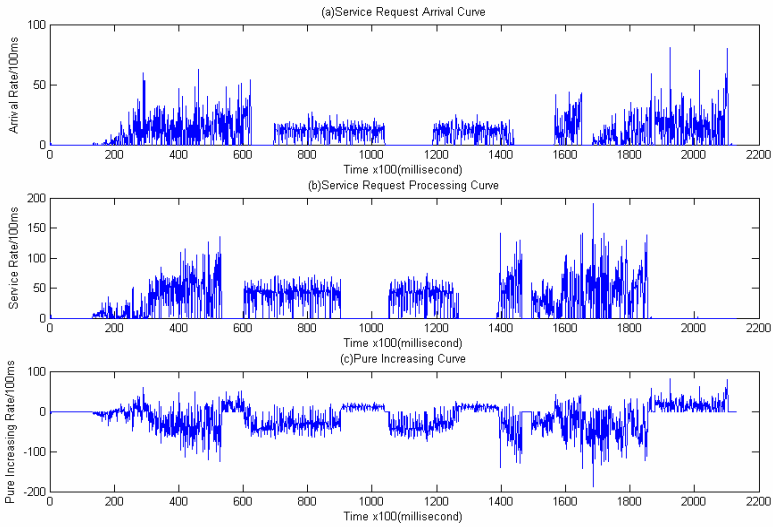
**Experiment Configuration:** We deployed Trade6 backend Web services on a 2.66GHz desktop with 2GB RAM, Windows XP, WebSphere Application Server V6.1, and DB2 V8.2. WebSphere Application Server is configured to use 1G heap. Trade 6 provides a stress client to simulate workload and service requests and we put it on another desktop with the same configuration parameters. The two machines are connected by 100Mbps LAN.

**Experiment Scenario:** We simulate the case that Trade6 serves five tenants, identified as tenant 0 to tenant 4. At the beginning, requests from 5 tenants are balanced, i.e. the request numbers of different tenants are identical. After 2 minutes, the request number from tenant 0 is increased largely so that the increase has negative impact on performance of other tenant or even pushes the whole system to the edge of crashing. What we want to see from this experiment is how SPIN can help in such scenario.

Following sections give detailed experiment data and explanation on anomaly report, isolation effectiveness and performance overhead of SPIN.

### 4.1 Anomaly Report

The experiment concerns firstly anomaly detection of WAQ model. Two time series *alpha* (Fig.2 (a)) and *beta* (Fig.2 (b)) are got by monitoring with a sampling interval of 100 milliseconds. The alpha series and beta series are respectively the arrival process and service process of hosting system. Series  $y=alpha-beta$  (Fig.2 (c)) represents the changes on processing capability of service system over time. Some bursts occur around the time points 300, 600, 1000, 1600, and 2000.



**Fig. 2.** Time series of service request arrival (a), request processing (c), and pure request increase (c) with settings of 2000 thread and 1500 times iterations

In our analysis, decomposition is made on the series at the resolution level one for clear experiment. WAQ threw out anomaly report after 190000 milliseconds since the start of the benchmark. In comparison, we do not adopt any moderation in the first running and find the service system finally collapse after about 20000 milliseconds later than the anomaly report.

Figure 3 presents that the predicted curve consists well with the monitored one, which demonstrates that WAQ can predict the changes both in arrival rate and service rate efficiently, no matter how wild fluctuations are. The accuracy of presented prediction (see Tab.1) is studied in terms of MRE (Mean Relative Error) [20]. Table.1 illustrates that the precision improves as the order of AR used in WAQ increases. Here we think that the parameter of order is properly set to “50” for the high precision and smaller computational complexity of WAQ.

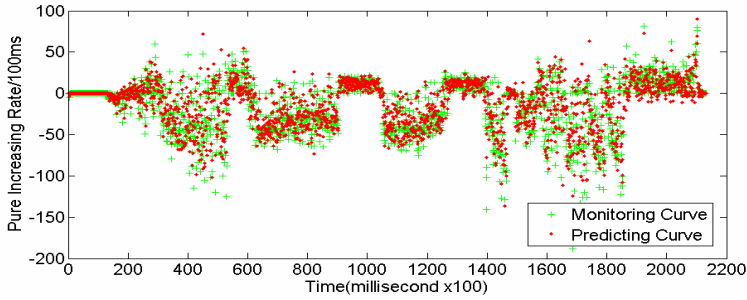


Fig. 3. Comparison of pre-series and post-series with ARP(15)

Table 1. MRE of Arrival Rate with different AR(p)

	AR(5)	AR(15)	AR(30)	AR(40)	AR(50)	AR(60)
MRE	36.94%	20.11. %	6.34%	4.68%	3.08%	2.25%

### 4.2 Isolation of Aggressive Tenant

After the anomaly report, the trace file written by system monitoring during service access is analyzed and resource volume of each tenant is collected. Figure 4 presents how CPU consumptions of every tenant change from the start to the anomaly report time. Other resources, such as memory, are omitted here to save space. In the forefront of the curves, each tenant consumes similar CPU cycles. A sudden rise occurs on the curve from the time of 1340. Tenant 0 is identified as the aggressive tenant because its consumptions of CPU cycles show the fastest increasing rate.

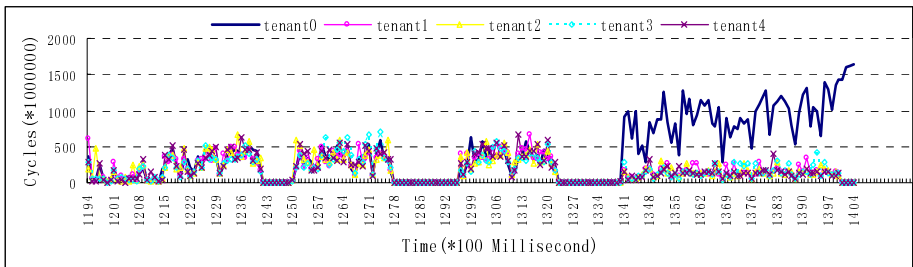


Fig. 4. CPU consumption curve for each tenant

With the rise of tenant 0, obvious drops lie on other tenants' consumption which decreases to almost 0 near the anomaly report time, 1407. Tenant 0 has already impacted negatively on other tenants. Without SPIN, the duration of this status will finally lead the non-aggressive tenants into the danger of starvation and service system into crash. The SPIN implementation adopts a direct policy to restrict and serve

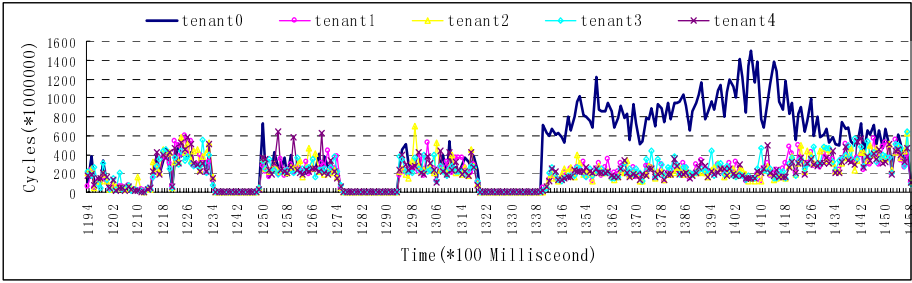


Fig. 5. CPU consumption curve with the moderation function open of SPIN

the requests from tenant 0 in the inverse ratio of its rise. Figure 5 presents the curve under the same simulation of workload with the moderation of SPIN open. The growth of Tenant 0 is limited not to reach the dangerous peak. The CPU consumptions of other tenants originally decrease and generally experience rallies. The percentages of non-aggressive tenants' consumption taking in all basically recover to the balanced state, which demonstrates the effectiveness of SPIN to keep performance isolation in Multi-tenancy environment. In practice, it is suggested to adopt a moderation policy consistent with SLA of different tenants.

### 4.3 Performance Overhead

To understand performance overhead brought into the original execution of service system, we watch the values of average response time before and after adoption of SPIN. To observe the values under different scales of workload, we tune two parameters, the number of threads (simulated clients) executing service access and iterations times executed in each thread. Performance overhead is calculated as  $(T_1 - T_0) / T_0$  for each different setting of the two parameters, where  $T_1$  and  $T_0$  are the average response time with SPIN and original benchmark respectively.

Figure 6 presents the overhead histograms with various settings of thread number (the parameter of iteration times is set to 2000) and interaction times (the parameter of thread number is set to 1500). For each setting, the benchmark runs 10 times, and the final results are obtained by calculating the geometric mean of the median of setting.

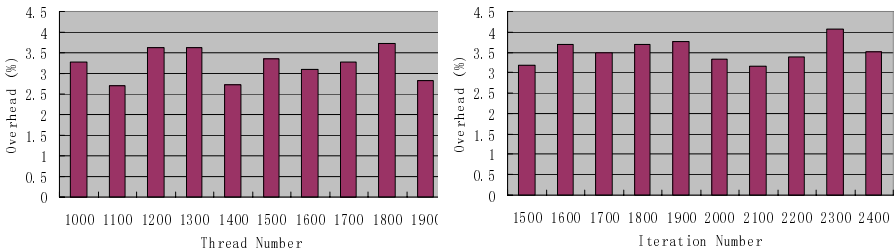


Fig. 6. Overhead histograms with various thread number and various interaction times

Overhead values with different settings are all less than 5%, which presents little negative impact caused by SPIN and its practicability under product environment.

## 5 Related Works

Performance isolation is not a new idea [24]. Jordan et al. [25] applies the concept of a Java resource accounting interface to isolate applications inside a JVM at the granularity of isolates to J2EE platforms. In comparison, our work focuses on performance isolation of tenants on the same copy of service. Work in [25] still depends on the resource reservation approaches and thresholds setting to limit the performance factors of isolations. Isolation for single Java virtual machines have been studied extensively [26, 27] and they focus on the security of multi-task in one JVM.

Authors in [11, 12] apply Queuing Theory in the study of Web service and adopt response time, throughput and reliability to evaluate performance of a service system. Three birth-death (BD) models are introduced in [13] to prove results on system throughput with the condition that the parameters of BD process have the same ratio. Renaud et al [14] address failure rate for the Web services market using Markov chain and Queueing Theory. They do not concentrate on the identifying how and when a service system becomes instable, but the comprehensive evaluation of performance.

In addition, there are some prior works for dealing with performance problems in server applications. That include request deletion in web servers [28], request prioritization or frame dropping in multi-media or real-time applications [29], and the creation of system level constructs supporting these application-level actions [30,31]. They share with adaptive techniques the use of runtime system monitoring and of dynamically reacting to certain monitoring events, but they differ in that focus is put on the decision of proper moderation policy. No attention is paid by them on the anomaly detection of service system. Moreover, they do not care about the resource monitoring at levels other than Process.

## 6 Conclusion and Future work

SPIN is purposed in this paper to achieve performance isolation of Multi-tenants on the service hosting platform with the maximum resource share. By a detection independent of any threshold, SPIN gives anomaly report in advance of the instability of service system. Resources, like CPU, consumed during service access are accounted on behalf of each tenant. Tenants whose consumption presents a trend of continuous growth are identified as aggressive ones that moderation will execute on. SPIN has been implemented open to self-tuning moderation without relying on user's input or directions. Practice of SPIN in Trade6 which has been revised to Multi-tenancy model demonstrates its accuracy of anomaly detection, effectiveness of isolation, and the low performance overhead (less than 5%).

Next step, we plan to adapt SPIN for SLA of different tenants. We will build separate anomaly detection for various tenants. Especially, proper moderation policy needs to design considering various SLA.

## References

1. Carraro, G., Chong, F.: Software as a Service (SaaS): An Enterprise Perspective, Microsoft Corporation (October 2006), <http://msdn2.microsoft.com/>
2. Gianforte, G.: Multiple-Tenancy Hosted Applications: The Death and Rebirth of the Software Industry. RightNow Technologies Inc. (2005), <http://www.rightnow.com>
3. Chong, F., Carraro, G., Wolter, R.: Multi-Tenant Data Architecture, Microsoft Corporation (2006), <http://msdn2.microsoft.com/>
4. Tsai, C.-H., Ruan, Y., Sahu, S., Shaikh, A., Shin, K.G.: Virtualization Based Techniques for Enabling Multi-tenant Management Tools. DSOM, 171–182 (2007)
5. Czajkowski, G., Daynes, L.: Multitasking without compromise: a virtual machine evolution. In: Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2001 (November 2001)
6. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
7. SWSoft, Virtuozzo, <http://www.sw-soft.com/virtuozzo>
8. IBM. WebSphere Application Server, Trade6 benchmark, <https://www14.software.ibm.com/wbapp/iwm/web/preLogin.do?source=trade6>
9. Waldspurger, C.A.: Memory resource management in vmware esx server. SIGOPS Operating Systems Review 36, 181–194 (2002)
10. Jones, S.T., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Geiger: Monitoring the buffer cache in a virtual machine environment. In: The 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII), pp. 14–24 (2006)
11. Hopp, W.J.: Single Server Queueing Models. In: Chhajed, D., Lowe, T. (eds.) When Intuition Fails: Insights From Basic Operations Management Models and Principles. Springer, Heidelberg (scheduled for publication, 2007)
12. Hall, R.W.: Queueing methods for services and manufacturing. Prentice-Hall, Englewood Cliffs (1990)
13. Feng, W.: Improving Service for Service Systems with Different Arriving Rate, PDCATapos. In: Proceedings of the Fourth International Conference on Volume, pp. 315–318 (August 2003)
14. Renaud, O., Starck, J.L., Murtagh, F.: Wavelet-based Forecasting of short and long memory time series [EB/OL]
15. Czajkowski, G., Eicken, T.V.: Internet Servers, Safe-Language Extensions, and Structured Resource Control. In: Proceedings of the Technology of Object-Oriented Languages and Systems, Nancy, France, pp. 295–304 (1999)
16. Hulaas, J., Kalas, D.: Monitoring of Resource Consumption in Java-based Application Servers. In: Proceedings of the 10th HP OpenView University Association Plenary Workshop (HPOVUA 2003), Geneva, Switzerland (2003)
17. Liang, S., Viswanathan, D.: Comprehensive Profiling Support in the Java Virtual Machine. In: Proceedings of the 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS 1999), San Diego, CA, pp. 229–240 (1999)
18. Sutherland, D.F., Greenhouse, A., Scherlis, W.L.: The Code of Many Colors: Relating Threads to Code and Shared State. ACM SIGSOFT Software Engineering Notes 28(1), 77–83 (2002)
19. Liu, Z.-X.: Short-term load forecasting method based on wavelet and reconstructed phase space. Machine Learning and Cybernetics 8, 4813–4817 (2005)

20. XiangXu, B., XinMing, Y., Hai, J.: Network Traffic predicting based on wavelet transform and autoregressive model. In: Tsui, F.-C., Sun, M., Li, C.-C., Sciabassi, R.J. (eds.) *Recurrent neural networks and discrete wavelet transform for time series modeling and prediction*, ICASSP, vol. 5(9-12), pp. 3359–3362 (May 1995)
21. Akaike, H.: Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics* 23(1) (December 1971)
22. Mallat, S.G.: *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation*. *IEEE Transactions on pattern analysis and machine intelligence* 11(7), 674–693 (1989)
23. Sun Microsystems, Inc. JVM Tool Interface (JVMTI), <http://java.sun.com/~j2se/1.5.0/docs/guide/jvmti/>
24. Barham, P., Dragovic, B., Fraser, K., et al.: Xen and the art of virtualization. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP 2003* (2003)
25. Jordan, M.J., Czajkowski, G., Kouklinski, K., et al.: Extending a J2EETM Server with Dynamic and Flexible Resource Management *International Middleware Conference, Middleware 2004* (2004)
26. Czajkowski, G.: Application isolation in the Java Virtual Machine. In: *Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA 2000* (2000)
27. Back, G., Hsieh, W., Lepreau, J.: Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. In: *Proceedings of the 4th International Conference on Operating System Design and Implementation (OSDI), San Diego, CA*, pp. 334–346 (2000)
28. Provos, N., Lever, C.: Scalable Network I/O in Linux. In: *Proceedings of the USENIX Technical Conference, FREENIX track* (2000)
29. Sundaram, V., Chandra, A., Goyal, P., et al.: Application performance in the QLinux multimedia operating system. In: *Proceedings of the 8th ACM International Conference on Multimedia 2000* (2000)
30. Poellabauer, C., Schwan, K., West, R., et al.: Flexible User/Kernel Communication For Real-Time Applications In Elinux. In: *Proceedings of the Workshop on Real Time Operating Systems and Applications* (2000)
31. West, R., Schwan, K.: Dynamic Window-Constrained Scheduling for Multimedia Applications. In: *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, ICMCS 1999* (1999)



# Management as a Service for IT Service Management

Bo Yang, Hao Wang, and Ying Chen

IBM China Research Lab., Beijing 100193, China  
{yangbbo,wanghcr1,chenying}@cn.ibm.com

**Abstract.** With the advent of the distributed computing model, IT service management has to face ranging from simple point products to entire enterprise frameworks to address the various multi-device systems management challenges. The existing technology offerings rarely solve the entire problem because they are expensive to purchase, difficult to implement, and do not incorporate the full range of features and functions to meet the systems management requirements today's organizations face. Now, with low-cost information appliances creeping their way into the data center, the only way to truly address today's heterogeneous IT challenge is to enable systems management capabilities at the service level. IT professionals should be able to simply take management service and accumulated knowledge to improve management performance. A new approach to IT service management is emerging that enables this level of integration. Management as a service (MaaS) enable IT service management to build network management appliances, services and knowledge repository that are extremely flexible and capable of evolving as customer needs evolve. MaaS implementations unify different systems management features and products into a common management environment that spans all system management types. MaaS-based management service offer greater flexibility and more cost-effective implementation than any enterprise framework can achieve. MaaS helps to make full use of the benefits offered by enterprise's converged network by identifying and resolving problems more quickly, more accurately, less expensively, and with more visibility than silos might be able to achieve on enterprise own.

## 1 Introduction

Agility of a company's response to customer demand has been recognized as a critical success factor to meet the global competition in current market environment [1][2]. More and more complex IT service assets that supporting business application require an automatic and flexible ability to react to the dynamic market environment, which forces companies to improve their efficiency and flexibility on IT service management. A key challenge for dynamic IT service management is that IT infrastructure is highly complex and configurable, and business requirement often have dynamic change on IT component configurations, which require administrator who in change of IT service management has

high professional skill about IT infrastructure in physical layer and business application in logical layer. With ever-growing complexity in the IT enterprise, IT departments have to face requirement of improving the alignment of IT efforts to business needs and managing the efficient providing of IT services with guaranteed quality [3].

To support this business requirement, more and more enterprises are adopting best practice standards in service management such as IT Infrastructure Library (ITIL) [4]. Configuration Management Database [5], CMDB, is considered as a best practice in the U.K. government's ITIL. It is used to store significant components information of the IT infrastructure [6][7][8] that helps an organization understand the IT service assets and track their configuration. Several vendors claim to provide a ready-made CMDB, such as BMC, Tideway, EMC and so on. It is common to see these vendors collaborate with the larger service players like IBM and HP who brand these solutions as their own implementations. Thus, CMDB has already been considered to be a provider that support vision on IT infrastructure information in industry field.

However, there is still no solutions for IT professionals manage IT service smartly. With ever-growing complexity in the IT infrastructure, even if all the information of the systems and operations are recorded, IT management personals can only get confused and be lost in the trivial details. IT service administrator has to face solution training for various problems in IT service management continually when business application updated. The investment on management system is always increasing with business application change.

With the question of how software delivers its functionality to users[9], Software as a service [10] (SaaS) envisages a demand-led software market in which businesses assemble and provide services when needed to address a particular requirement. The SaaS vision is a vital contribution to current thinking about software development and delivery that has arisen in the Web services and electronic-business communities.[11]

In this paper, just like as SaaS, management as a service (MaaS) is proposed to focus on separating the possession and ownership of IT service management from its use. Delivering management function as a set of distributed services that can be configured and bound at delivery time can overcome many current limitations constraining management software use, deploy, and evolution. In addition, MaaS has a special feature unlike SaaS is that it will provide the content service that help IT professionals to improve IT service management quality. Experience and knowledge about IT service management will be accumulated within MaaS. The knowledge can be reused and shared in different environment when they have been verified in prior cases by IT professionals and subject matter expert (SME).

## 2 Current IT Service Management

Configuration management database (CMDB) is provided to serve as a repository and information retrieval tool for services as IT infrastructure model, and also as a platform for information integration between the other ITSM processes.



Business application definition demands professional and rather experienced experts to define business application domain, such as what are the main components in the business application, what are the key configurations for management, etc. One of the key open challenges to experts is performing dynamic configuration management for highly configurable business application with dynamic user preferences. IT services are typically highly configurable, with significant service customization possibilities and a choice of business application purpose, e.g. CRM/Home banking, naturally each with its own application configuration. Customization is critical for them to be able to differentiate themselves from competing business services and offer a better service experience to their customers. Application administrators themselves have certain preferences on which service items and configurations they want to use and their preferences may change dynamically. Given their significance, there is a strong need to support application-oriented data services for user preference [14][15][16]. However, we are still lack of efficient methods for the representation and management on user-preference in IT service management.

Current ITSM architecture uses processes to integrate disconnected tools and configuration information into an end-to-end solution. It provides the framework to optimize the use of people, process, information and technology to deliver quality services to support business goals rather than focuses on disconnected technologies and tools. However, as we discussed above, current architecture lacks a systematic means to address an important aspect in ITSM - managing the knowledge separated in various tools and the minds of domain experts.

### 3 Sample MaaS Scenario

The following scenario demonstrates the ideas inherent in the management as a service concept.

In Fig. 2, SME individuals, communities with different skill levels and focuses get involved in service tasks through the Management Service Center platform. They provide the knowledge and solution for various business problems. Service Center staffs manage multiple remote customer sites, process service requests and conduct complex, knowledge intensive tasks, while customer employed administrator performs the routine management tasks for his organization.

At the customer site, a management appliance is deployed. It provides basic management functions for customer employed staffs to conduct routine IT management tasks. As an appliance, it hides the complexity of deploying and managing the management software itself.

Customer admin can access the global service catalog through the appliance to subscribe advanced management functions, or request services from the service center. The appliance will also be an agent for collecting the necessary information for required services.

For the management service center, the staffs can manage multiple customer sites remotely through the service portal. They can deliver advanced management services to customers by using advanced management functions and the

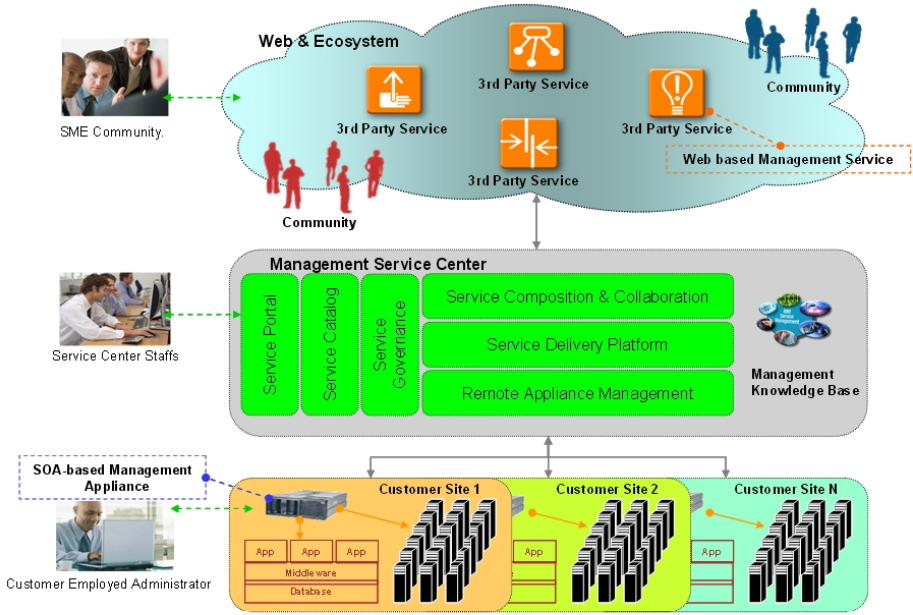


Fig. 2. Deliver management as a service

knowledge base. Some required functions to enable the services will be deployed to the appliance dynamically.

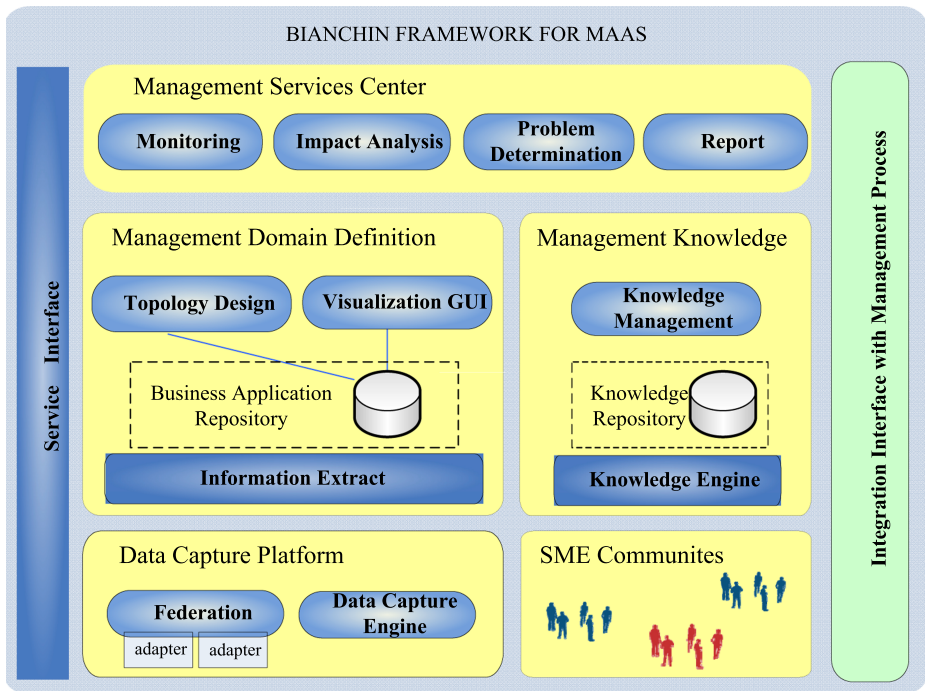
The service center is also the platform to involve external communities to serve the customers. It can integrate services through the web to complement existing services and it can involve selected expert communities for specific tasks. Meanwhile, the service center will manage the service delivery process with good governance.

## 4 Implementation of the MaaS for ITSM

In this section, a proof of concept (PoC) project BIANCHIN is developed to validate the vision of management as a service (MaaS), as shown in Fig. 3. It is built on the Eclipse Toolkit with Java technology, and implements a container for data capture platform, information extract, business application topology design and visualization, SME’s knowledge management and impact analysis service for system management in global within distributed computing environment.

### 4.1 Data Capture Platform

This platform is in charge of collecting IT infrastructure data from various domains including configuration, operation, business-level performance indicators and others. Two adapter-based mechanisms, data federation and snapshot, are



**Fig. 3.** BIANCHIN framework for MaaS

designed in the platform. For data already discovered and stored in existing monitoring tools such as IBM Tivoli Monitoring product and IBM Tivoli Application Discovery Dependency Manager (TADDM) [13], adapters can be developed for data federation. For other necessary data not stored in any existing data repositories, snapshot adapters can be developed and plugged in to broaden the width of data collection. The Data Capture Engine controls the data capture process. It decides which adapters should be enabled or not to control the capture scope. It also controls the frequency of data capture by triggering the adapters at different time points according to a pre-defined schedule.

## 4.2 Information Extract

IT infrastructure data collected from various data sources are mainly for showing the details of IT service components such as configuration items (CIs) and their relationships, normally not reflecting a special business application and its topology. The data available in an enterprise with complicated IT environment is always too huge and complex to management. What data should be provided for managing business applications, which one is in the charge of administrator, is still a pain point for ITSM.

In BIANCHIN, information extract module provides a platform to help administrator to specify the data set that belong to a certain business application. Once defined, the target management domain (subset of IT infrastructure data) can be reused for various management task associated to the business application. And it is knowledge that can be shared with other administrators in the same management domain for improve management efficiency, because target management domain definition is a time-consuming work that requiring high skill on IT infrastructure analysis.

### 4.3 Business Application Topology Design and Visualization

Business application topology vision is very helpful to user to visualize a mapping from business application to IT infrastructure, which provides a view to know what hardware, software, middle-ware, service are dependent by the application. Then a management on financial, security, resource, etc. can be operated well-founded. Moreover, it provides a potential asset report to IT infrastructure investor about the IT service utilization on the business application.

BIANCHIN provide a customized IT infrastructure topology layout for various management scenarios. End-user can design their preference visualization about the target management domain. It allows data filter in different topology to highlight key components related to specified application, transfer different layout between various user preference styles.

### 4.4 SME's Knowledge Management

Knowledge Repository and Knowledge Management module provide a service to accumulate and share with SME's knowledge about IT service management. The knowledge coming from SME is stored in the knowledge asset repository. These assets serve as basic constructing pieces to build more specific and meaningful forms of knowledge for various operational tasks. The knowledge for different management task groups is organized as knowledge management categories such as problem determination, configuration recommendation and impact analysis. For example, a service outage can be defined as a problem data pattern along with a symptom description in the knowledge asset repository. The pattern detection facility can automatically discover from the captured data to notify if the problem occurs. Once the problem is detected, data comparison facility can figure out the most possible root causes of the problem by presenting the changes between the current version and the latest healthy version. Once the root cause is confirmed as a combination of incorrect or conflicting system configurations, it can be recorded as another data pattern as a root cause to the known problem. The management staffs can associate the root cause data pattern with the problem data pattern as well as textual descriptions and the fixing solution in the knowledge asset repository. And then, when the problem happens in the future, the system can automatically check if it is caused by the same reason and if does.

## 4.5 Management Services Center

As mentioned above, IT service management is becoming more complex in distributed IT infrastructure than before. It need more experience and knowledge coming from high skill staffs for improving management quality and efficiency. Take management as a service provides a cooperation framework for sharing management knowledge with more quickly, more accurately, less expensively than silos might be able to achieve on enterprise own.

Management services center in BIANCHIN for different management task groups is organized as IT service management categories such as monitoring, impact analysis, problem determination and report service. Those services can be invoked when perform a remote IT service management work with SME knowledge on demand for various management scenario. They can be embedded in web-based management platform, SOA-based management application and so on.

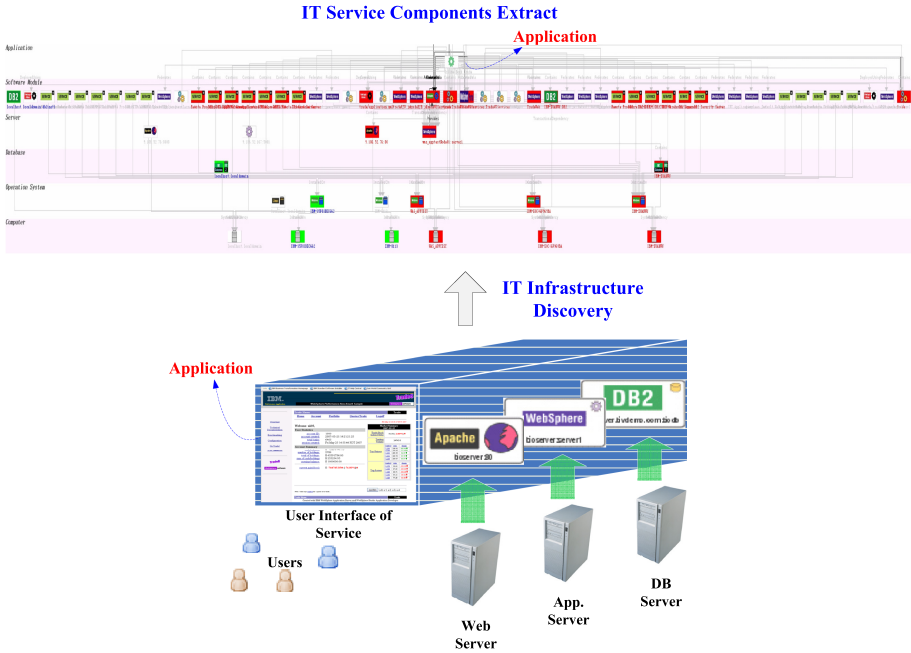
## 5 Case Study

The example selected to demonstrate the cooperation of IT service management based on management services within solution matter experts (SMEs) that sharing the management knowledge for a special business application, "On-line Trade", a benchmark of distributed J2EE business application. The services provide a platform for IT service topology of a business application, and delivering the topology defined by application experts to the business impact analysis SMEs. Then, with knowledge integration on various impact rules of business impact by IA SMEs, an impact analysis service is invoked for analyzing a starting point change, such as a database server outage, operation system patch, etc. It returns the impact scope including hardware, software, service, application and so on. Moreover, the reason can be attached on the report because of the SME's impact rules. This function was originally implemented in expert system within the scope of problem determination project. Then it was decided to reuse it as a service in SOA.

### 5.1 Extracting Business Application Management Domain

Within the CMDB of IT service management, the target components of business application that will be monitor and management were mix with other redundant IT service components as shown in Fig. 4, so the first step in ITSM was to extract target management domain from the complex environment with thousands of IT service components and to place it in a separately compliable module with its own components division. This work needs special skill that the operator is very familiar with the target application domain, especially the topology of IT infrastructure about the application. The candidates might be the engineer who designed the application, who deployed the application and the execution administrator. In the "Online Trade" application case, the application





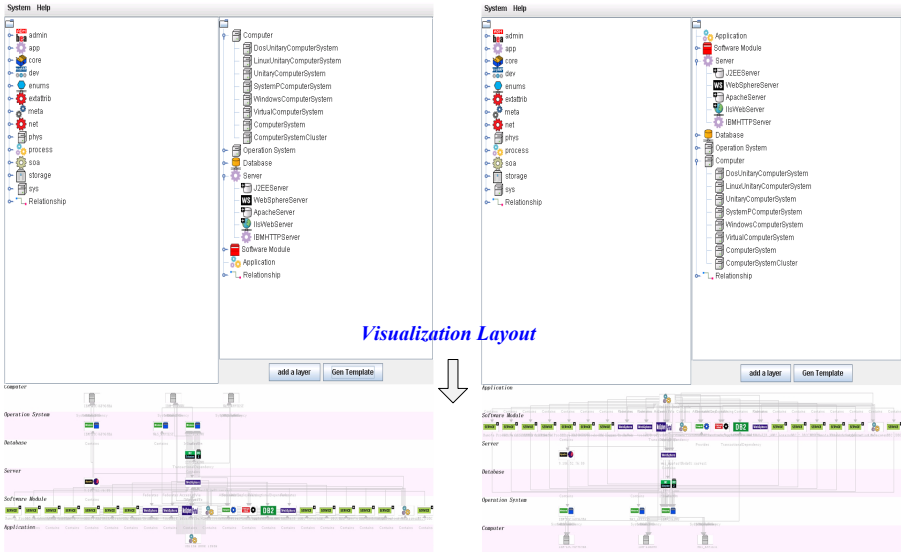
**Fig. 4.** IT service components extract for business application from IT infrastructure of “Online Trade”

is extracted that contains 32 components from 73 IT service components, and 73 relationships from 122 relationships.

Besides the target components extraction, a service of customized business application topology vision for different context is import for special scenario. Different angle of view will highlight some specified details to help audience to understand the target object easily. Especially, the user preference style will make user to learn efficiently that will improve the communication within different users based on the same data set with user-preference visualized layout. As shown in Fig. 5, a bottom-up business application topology layout and a top-town layout from application indicate two management scenarios. The former is the system administrator’s preference layout, and the later would be helpful for application administrators.

### 5.2 Impact Analysis Rules Management

A distributed business application always includes various components for composing a complex function for special business purpose. It may be include hardware components (host, hard disk, memory, CPU, etc.), operating system (Windows, Linux, UNIX, etc.), database (DB2, Oracle, MySQL, etc.), application server (Websphere Server, J2EE server, WebLogic server, etc.), software module (J2EE module, IIS module, Web module, etc.) and so on. As to impact



**Fig. 5.** Customized Business Application Topology Design GUI

rule, there is no an omniscient to define all rules about various fields. Hardware engineer should be the best candidate to define the impact rules about hardware components such as memory problem might cause blue screen problem. And windows SP2 patch will impact all software running on it because it will cause windows operating system reboot. Till now, those knowledge stored in different SMEs, and do not incorporate the full range of features and functions to meet the systems management requirements today’s organizations face.

Management as a service is not only provide service to remote system management, but also sharing the management knowledge in various SMEs’ knowledge for dealing with emerging complex business application.

In our case, different knowledge coming from various SME are combined in impact analysis knowledge repository. The BIANCHIN provide impact analysis service with professional knowledge to solve management problem. Customers can invoke the service and associated knowledge to archive their management purpose.

An impact analysis rule in BIANCHIN consisted of five parts, rule name, source, target, relationship, condition and impact. Rule name is a simple description about the impact rule. Relation denotes that a dependency between source point and target point in IT service components. Condition is the fire point of the rule, such as condition in rule 1 shown in Fig. 6., “source=normal; target=impacted” means that “if Application server that runs on a windows computer system, and the windows computer system is impacted” is true, then the rule will be fired, and the application server will be impacted too, which is as a impact result in the rule.

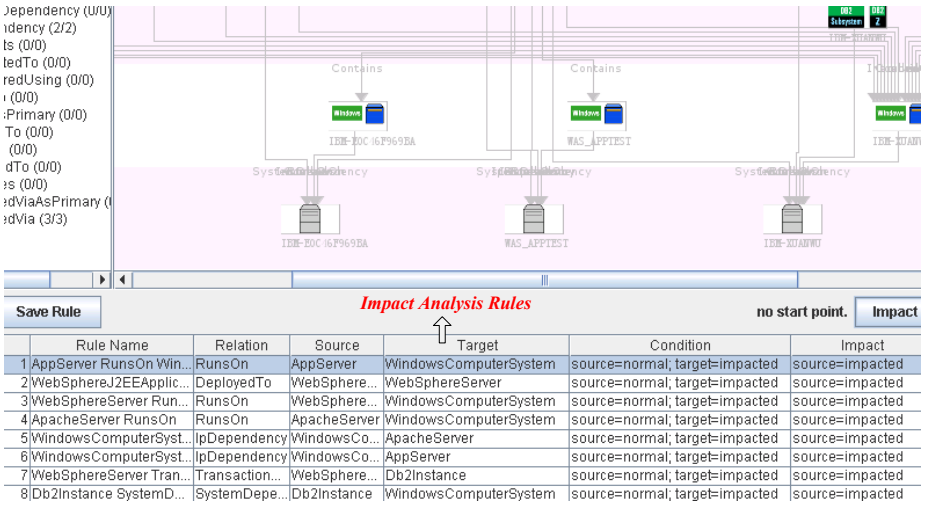


Fig. 6. Knowledge management module of impact analysis rules

With the rule management in BIANCHIN as shown in Fig.6, various SME can cooperate with each other for editing and verifying the rules through Internet in global, which accumulates the rules coming from SME to knowledge repository for various business solutions. In the case, they could be OS patch impact analysis, DB2 maintenance impact analysis, hardware updating impact analysis and so on.

### 5.3 Invoke Impact Analysis Service

After user import the target management domain that defined by domain experts and knowledge base coming from SME, impact analysis service can be invoked to deduce a impact scope for a specified starting point. In BIANCHIN, it has been packaged in applet, and use the sample code as follow to invoke the service for web application.

```
<applet archive="lib/ImpactAnalysis.jar, lib/jxl-2.4.2-s.jar, lib/UIframework.jar, lib/sdl1.0-s.jar" code="com.ibm.research.component.ImpactAnalysisApplet.class" codebase="." service_url = "http://192.168.100.33:8080/axis2/services/IAService/ieInterface" width="980" height="680" align="center"> </applet>
```

The above sample code shown that an SOA-based approach, impact analysis service “http://192.168.100.33:8080/axis2/services/IAService/ieInterface” is invoked for deducing the impact scope with the starting point and SME knowledge automatically.

For example, a management operation request will cause windows operating system outage on component “WAS-APPTEST” in Fig. 7. Expected components that will be impacted are mark with red cube when the “Impact Assessment” is

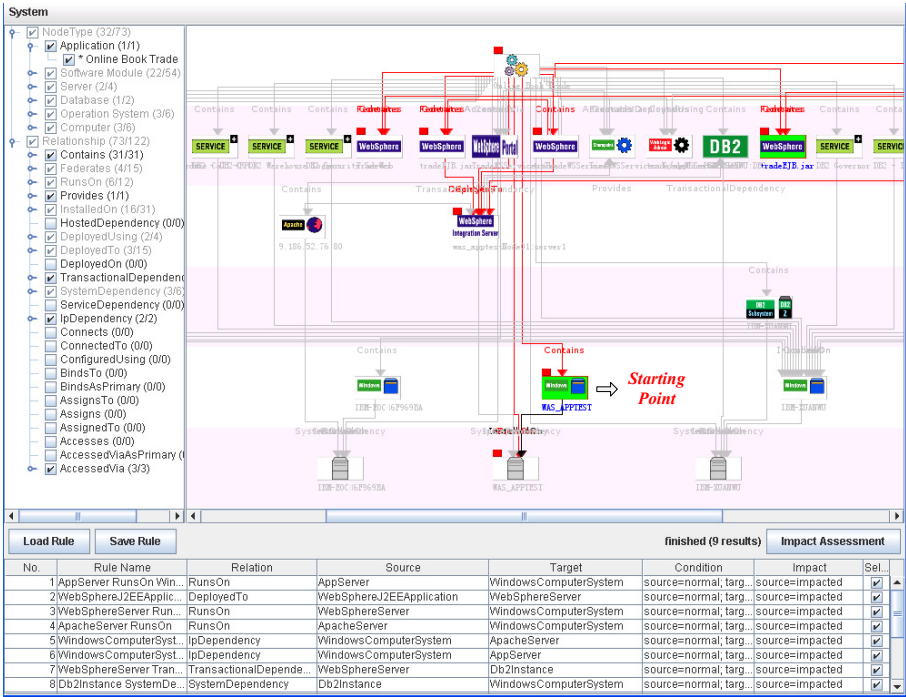


Fig. 7. Impact assessment from a specified starting point in target management domain

finished. There are 9 components will be impacted because of the starting point operation in the target application domain with 32 components. From the starting point, computer “WAS-APPTEST” will be impacted because the windows operating system running on it is outage. A websphere application server runs on “WAS-APPTEST” will be impacted because the computer is impacted. Then the components “tradeEJB.jar”, “tradeWebModule”, “tradeJ2EEapplication” will be impacted because they were deployed on the websphere application server. Farther on, a business application “Online Trade” will be impacted because its components have been impacted.

This case provides a whole picture for user to review what will happen when a component is changed, which will help to do a just operation more quickly, more accurately, less expensively, and with more visibility than silos might be able to achieve on enterprise own.

## 6 Discussion and Future Work

In this paper, we present a vision of management as a service for improving IT service management performance, and a case of impact analysis service in MaaS is shown that the performance of system management is not only depends on the

software system of management, but also the knowledge in administrator and experts. People need to change their mindsets. IT service management solution has to think in terms of providing a service itself, not just providing technology. Take management as a service will identify and resolve management problems more quickly, more accurately, less expensively, and with more visibility than silos might be able to achieve on enterprise own.

Although MaaS has some problems need to identify and solve, such as in our case, impact analysis service, how to evaluate the quality of knowledge, how to acknowledge and benefit the SME who contribute their knowledge, how to authorized the knowledge and data can be used in properly, are still open questions. As an emerging solution for improving IT service management in global enterprise, MaaS has shown some potential chance in current dynamic market environment.

Management as a service (MaaS) allows companies to maintain full internal control of their IT investment and existing infrastructure while outsourcing the day-to-day management and monitoring of their infrastructure such as the operating system, database, networking and application layer.

MaaS vision manages and supports distributed business application in enterprise that includes IT infrastructure such as operating system, application and database layers, providing experts' knowledge to ensure that IT infrastructure is performing optimally.

MaaS extends into four levels of service - conveniently packaged and easily scalable to help administrator adopt remote management for routine and time-sensitive tasks. Remote management assistance service can be used as a consultant service for explore more business chance. With continual knowledge accumulation on seamless management and technical support by the SMEs - these packages ensure that end-user experience and key business functions are improved continually. Finally, all these come to end-user at an affordable cost without maintaining complex management system reside in user's enterprise.

## Acknowledgement

We thank the members of the IBM GTS for their comments coming from customers' feedback on IT service management. We also thank the members of the visualization team in IBM CRL for their useful discussions concerning GUI design and ITSM team for the ongoing development of a service-oriented prototype.

## References

1. Changchien, S.W., Shen, H.Y.: Supply chain reengineering using a core process analysis matrix and object-oriented simulation. *Information & Management* 39(5), 345–358 (2002)
2. Liu, J., Zhang, S., Hu, J.: A case study of an inter-enterprise workflow-supported supply chain management system. *Information & Management* 42(3), 441–454 (2005)

3. HP BTO software: Optimize the business outcome of IT, White paper, 4AA0-8911ENW (2006), <http://www.hp.com>
4. Information Technology Service Management (ITSM) (2005), <http://www.cce.umn.edu/professionalcertification/itil/>
5. Madduri, H., Shi, S.S.B., Baker, R., Ayachitula, N., Shwartz, L., Surendra, M., Corley, C., Benantar, M., Patel, S.: A configuration management database architecture in support of IBM Service Management. *IBM Systems Journal* 46(33), 441–457 (2007)
6. Berkhout, M., Harrow, R., Johnson, B., Lacy, S., Lloyd, V., Page, D., van Goethem, M., van den Bent, W.G.: *Service Support: Service Desk and the Process of Incident Management, Problem Management, Configuration Management, Change Management and Release Management*. The Stationery Office, London (2000)
7. Chen, P.Y., Kataria, G., Krishnan, R.: On Software Diversification, Correlated Failures and Risk Management. SSRN (April 2006), <http://ssrn.com/abstract=906481>
8. Van Bon, J., Kemmerling, G., Pondman, D.: *IT Service Management: An Introduction*. Van Haren Publishing (September 2002)
9. Brereton, O.P., Budgen, D.: Component-Based Systems: A Classification of Issues, *Computer*, pp. 54–62 (November 2000)
10. Bennett, K.H., et al.: An Architectural Model for Service-Based Software with Ultra-Rapid Evolution. In: *Proc. Int'l Conf. Software Maintenance (ICSM 2001)*, pp. 292–300. IEEE CS Press, Los Alamitos (2001)
11. David, M.T., Brereton, B.P.: Turning Software into a Service. *Computer*, 38–44 (October 2003)
12. Sall, M.: *IT Service Management and IT Governance: Review, Comparative Analysis and their Impact on Utility Computing*. HPL-2004-98, Technical Reports, Hewlett-Packard Company (June 2004)
13. Moeller, M., James, J., Choilawala, M., Kadijevic, P., Charles, R., Satagopan, K.: *Deployment Guide Series. IBM Tivoli Change and Configuration Management Database Configuration Discovery and Tracking v1.1, SG24-7264-00*, Redbooks, IBM, Inc. (November 2006)
14. Chen, Z.Y., Li, T.: Addressing diverse user preferences in SQL-query-result navigation. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data table of contents*, Beijing, Chinan, Beijing, Chinan, pp. 641–652 (2007)
15. Jung, S.Y., Hong, J.H., Kim, T.S.: A statistical model for user preference. *IEEE Transactions on Knowledge and Data Engineering* 17(6), 834–843 (2005)
16. Wong, S.K.M., Lingras, P.: Representation of Qualitative User Preference by Quantitative Belief Functions. *IEEE Transactions on Knowledge and Data Engineering* 6(1), 72–78 (1994)

# SMART: Application of a Method for Migration of Legacy Systems to SOA Environments

Sriram Balasubramaniam, Grace A. Lewis, Ed Morris, Soumya Simanta,  
and Dennis Smith

Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa 15213  
{sri,glewis,ejm,ssimanta,dbs}@sei.cmu.edu

**Abstract.** Migration of legacy systems to service-oriented environments has been achieved within a number of domains, including banking, insurance, manufacturing, and development tools, showing that the promise is beginning to be fulfilled. While migration to Service-Oriented Architecture (SOA) environments can have significant value, any specific migration requires a concrete analysis of the feasibility, risk and cost involved. The Service Migration and Reuse Technique (SMART) is a family of processes to help organizations in making initial decisions about the feasibility of exposing legacy systems as services within a SOA environment. This paper summarizes the SMART-MP (SMART – Migration Planning) process which provides a plan for the migration of selected components to services and identifies the scope for a pilot effort. It presents an application of SMART-MP and also outlines emerging new SMART family members that have resulted from experiences in applying SMART-MP.

**Keywords:** SOA Migration, service selection, reuse of legacy components, SOA migration strategy.

## 1 Introduction

Service-Oriented Architecture (SOA) has become an increasingly popular mechanism for achieving interoperability between systems. Because it has characteristics of loose coupling, published interfaces, and a standard communication model, SOA enables existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems. Migration of legacy systems to SOA environments has been achieved within a number of domains, including banking, insurance, manufacturing, electronic payment, and development tools, showing that the promise is beginning to be fulfilled [1, 2, 3, 4]. While migration can have significant value, any specific migration requires a concrete analysis of the feasibility, risk and cost involved.

One of the most attractive promises of moving to an SOA environment is that it enables reuse of legacy systems, thereby providing a significant return on the investment in these systems. However, migrating legacy systems is neither automatic nor easy. Traditional reuse challenges apply to SOA environments, but are affected in both positive and negative ways by the granularity of what is exposed through reuse

[5]. Reuse in the SOA context is typically most effective when services correspond to coarse-grained business or mission functionality, i.e. order placement or flight path calculation, where all underlying technical details are encapsulated by a standard service interface.

The Service Migration and Reuse Technique (SMART) is a family of processes that helps organizations to make initial decisions about the feasibility of exposing legacy system functionality as services within an SOA environment [6, 7]. The most common implementation, SMART-MP (Migration Pilot), focuses on identifying the scope for a pilot effort. It offers a realistic view of the potential challenges, constraints and trade-offs that an organization may encounter when migrating to SOA environments.

SMART-MP is the method that was described in earlier SMART references and it remains the most often used member of the SMART family [7]. SMART-MP was developed in response to a situation that is common in the US Department of Defense, where organizations are tasked to provide services for which other organizations will develop service consumers in the form of end-user applications, portals, or systems. SMART-MP has evolved significantly since its initial introduction including the refinement of its underlying conceptual model, refinement of the questionnaire that supports it, tool support, addition and refinement of tasks, and the application of lessons learned from past engagements. Applications of SMART-MP have led to the development of other members of the SMART family, including one that specifically addresses the situation where a single organization develops services and service consumers, as well as the infrastructure, which is more common in industry. However, given that services typically come from legacy systems, elements of SMART-MP are highly applicable in this situation as well.

The rest of this paper is organized as follows. Section 2 presents the SMART-MP process. Section 3 presents the results of a SMART-MP engagement. Section 4 outlines other members of the SMART family. Section 5 presents a summary and conclusions. Section 6 presents related work.

## **2 Service Migration and Reuse Technique – Migration Planning (SMART-MP)**

SMART-MP is an approach for making decisions on the migration of legacy systems to SOA environments. To make effective decisions, relevant and non-relevant legacy components need to be identified and decisions need to be made using a “hands-on”, contextual analysis. Estimates of cost and risk, as well as the confidence of estimates, are required for each legacy component. It analyzes the viability of reusing legacy components as the basis for services by answering these questions:

- Does it make sense to migrate the legacy system to services?
- What services make sense to develop?
- What components can be used to implement these services?
- What changes to components are needed to accomplish the migration?
- What would be an ideal pilot project?
- What migration strategies are most appropriate?
- What are the preliminary estimates of cost and risk?



## 2.1 Four Elements of SMART-MP

SMART-MP consists of four elements:

1. **SMART-MP Process.** A systematic process that gathers information about the legacy components, the candidate services and the target SOA environment; analyzes the gap between the current and future state; and develops an initial migration strategy.
2. **Service Migration Interview Guide (SMIG).** The SMIG is the codification of risks and issues associated to migration to SOA environments that guides the discussions during the initial SMART process activities. It contains more than 60 categories of questions that gather information about the migration context, the legacy components, the candidate services, and the target SOA environment. Answers to these questions will help determine the degree of difficulty and level of effort required to migrate legacy components into services, as well as potential migration issues. The use of this instrument assures broad coverage and consistent analysis of difficulty, risk, and cost issues. An example of the SMIG questions for *Describe Target SOA Environment*, as well as some of the potential migration issues related to these questions, is provided in Table 1.
3. **SMART Tool.** Using the SMIG as a framework, the tool automates data collection, relates answers to questions to potential risks to mitigation strategies to produce a draft migration strategy and migration issues list, and consolidates data from multiple engagements for trend analysis.

**Table 1.** Sample SMIG Questions and Potential Migration Issues

Discussion Topic	Related Questions	Potential Migration Issues
SOA Environment Characteristics	<ul style="list-style-type: none"> <li>• What is the status of the target SOA environment?</li> <li>• What are the major components of the SOA infrastructure?</li> <li>• Does the target SOA environment provide infrastructure services (i.e., communication, discovery, security, data storage)?</li> <li>• What is the communication model?</li> <li>• What constraints does the target SOA environment impose on services?</li> <li>• Does the legacy system have any behavior that would be incompatible with the target SOA environment?</li> <li>• Once developed, where will services execute?</li> </ul>	<ul style="list-style-type: none"> <li>• Target SOA environment undefined</li> <li>• Redundancy/conflicts between infrastructure services and legacy code</li> <li>• Lack of tools to support legacy code migration to target infrastructure</li> <li>• Compliance with constraints requires major effort.</li> <li>• Architectural mismatch</li> <li>• No thought given to service deployment and execution</li> </ul>
Support	<ul style="list-style-type: none"> <li>• Do you have to provide automated test scripts for the services and make them publicly available?</li> <li>• How will service consumers report problems and provide feedback?</li> <li>• How will service consumers be informed of potential changes in service interfaces and downtime due to upgrades or problems?</li> </ul>	<ul style="list-style-type: none"> <li>• Underestimation of effort to provide service consumer support</li> <li>• Lack of awareness of support requirements</li> </ul>

4. **Artifact Templates.** Templates for output products are created as part of the process. There are a variety of templates, including Migration Issues List, Business Process-Service Mapping, Service Table, Component Table, Service-Component Alternatives Table and Migration Strategy.

## 2.2 The SMART-MP Process

The SMART-MP process has six activities, and one major decision point, as illustrated in Fig. 1. The activities are iterative—data gathered in one activity may provide questions that require revisiting an earlier activity for additional information.

### **Establish Context**

*Establish Context* has the goal of understanding the business and technical context for migration, including a high level understanding of stakeholder goals, the business context, candidate services, and legacy systems.

### **Migration Feasibility Decision Point**

After the *Establish Context* activity, there is an explicit decision point to determine if the legacy system is a good candidate for migration to services. If the legacy system is not a good candidate, it will save time and money to simply stop at this point.

### **Define Candidate Services**

The goal of this activity is to select a small number of services (usually 3 to 4), from the initial list of candidate services that had been identified as part of *Establish Context*. Good candidate services are ones that perform concrete functions, that have clear inputs and outputs, and that can be reused across a variety of potential service consumers. These candidate services are now specified more completely to include a definition of service inputs and outputs, and quality of service (QoS) requirements.

### **Describe Existing Capability**

The goal of this activity is to gather information about the legacy system components that contain the functionality to meet the needs of the selected services. Technical personnel are questioned about system aspects such as: descriptive data about legacy components, architecture views, design paradigms, system quality, and change history.

### **Describe Target SOA Environment**

This activity gathers information about the target SOA environment for the selected services including:

- Major components of the SOA environment
- Impact of specific technologies and standards used in the environment
- Guidelines for service implementation
- State of target environment
- Interaction patterns between services and the environment
- QoS expectations and execution environment for services

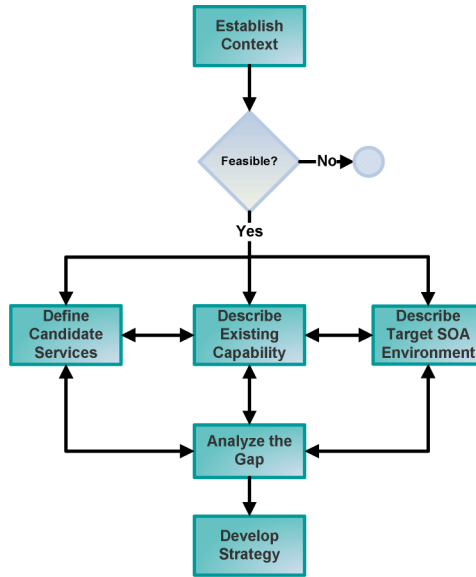


Fig. 1. The Generic SMART Process

### Analyze the Gap

This activity provides preliminary estimates of the effort, risk and cost to expose functionality from the candidate legacy components as services, given the candidate service requirements and target SOA characteristics. The discussion of the changes that are necessary for each component is used as the input to calculate these preliminary estimates. In some cases, additional analysis methods such as evaluation of code quality using code analysis tools or architecture reconstruction may be needed [8, 9].

The Service-Component Alternatives artifact is created during this activity to illustrate the potential sources for functionality to satisfy service requirements.

### Develop Strategy

The information gathered in the previous activities generates migration issues that need to be addressed by the migration strategy, which for SMART-MP includes the selection and setup of an initial pilot project. This information also provides the basis for estimates of cost, effort and risk of migration, which will place constraints on the migration strategy. This activity develops a Migration Strategy that may include:

- Feasibility, risk and options for proceeding with the migration effort
- Identification of a pilot project to migrate a simple service (or set of services)
- Order in which to create additional services
- Guidelines for identification and creation of services
- Specific migration paths to follow
- Needs for additional information or training

### 3 Application of SMART-MP to a Mission Status System

The following is a summary of the application of SMART-MP to a DoD Mission Status System (MSS). Each sub-section corresponds to a step of the SMART-MP process.

#### 3.1 Establish Context

A DoD organization has been tasked with developing services that can be used by multiple mission planning and execution applications.

The Mission Status System (MSS) targeted for migration is a system for comparison of a planned mission against current mission state to determine if corrective actions should be taken. The system obtains plan data and situational awareness data from a Planning System (PS) that belongs to the same organization. PS currently runs on the same machine as MSS and there is tight coupling between the two systems.

From a business perspective, a long term goal has been the migration of MSS to services. From a technical perspective, the driver is to make the developed services available to all planning and execution systems.

A standard Web Services environment has been selected as the target SOA environment, but it is not clear that this will be the future environment for the developed services—it will most likely be a DoD proprietary SOA infrastructure. However, by performing and executing this pilot, valuable insight on the migration process will be gained, and the overall process, as well as at least a significant part of the analysis will be able to be carried forward.

The short-term goal for the migration is to demonstrate the feasibility of one MSS component as a service to be used by one mission planning and execution system within four months. The long-term goal is to migrate the full system to services in two years.

The following set of candidate services was developed:

- AvailablePlans: Provides list of available plans that are being reasoned about.
- TrackedTasksPerPlan: Provides list of tasks that are being tracked for a certain plan.
- TaskStatus: Provides the status for a given task in a given plan.
- SetTaskAlert: Alerts when a given task in a given plan satisfies a certain condition.

These services were selected because their functionality is generic enough that it can be used by other known mission planning and execution systems.

#### 3.2 Migration Feasibility Decision Point

Based on the data obtained at this stage, a decision was made to continue with the rest of the SMART analysis. This was based on the following factors:

- the availability of stakeholders from the service provider and a service consumer
- a good understanding of MSS
- the request-response nature of the identified services
- a reasonable initial mapping of services to components

### 3.3 Define Candidate Services

The list of services identified in the previous step was considered reasonable for analysis. Inputs and outputs were next identified in detail for each of these services.

Migration issues identified during this activity included:

- The SetTaskAlert service requires the handling of events in service-oriented environments which is something that has been recently introduced in SOA 2.0 [10]. The implementation of SetTaskAlert would require either the service provider or the infrastructure to store the address of the service consumer so that it knows who to alert, and requires the service consumer to be set up such that it can receive alerts.
- It is unclear how the alert mechanism is going to be implemented.
- The complexity of the data representation of alert conditions is high.

### 3.4 Describe Existing Capability

MSS is written in C++, C# and Managed C++ in a Visual Studio 2005 development environment. It runs on a Windows XP platform. The size of the full system is approximately 13,000 lines of code. The amount of code considered for migration depends on the scope of the migration effort, although most of the code is being targeted for migration in the future. Code documentation was rated between Fair and Good by its developers.

Several architecture views were presented that were useful for understanding the system—high-level context diagram, component-and-connector view, module view, and runtime view.

### 3.5 Describe Target SOA Environment

As mentioned earlier, the target SOA environment for the migration is standard Web Services. It was decided to use an existing setup based on Microsoft IIS and ASP.NET.

### 3.6 Analyze the Gap

During this activity, the developers were asked to describe the details of the changes that would have to be made to the code given the service requirements, the service inputs and outputs, as well as the characteristics and components of the target SOA environment. They were then asked to provide an estimate of the effort required to make these changes. No code analysis or architecture reconstruction was necessary because the original developers were involved in the process, their input was credible, and the architecture documentation and knowledge of the application were acceptable.

### 3.7 Develop Strategy

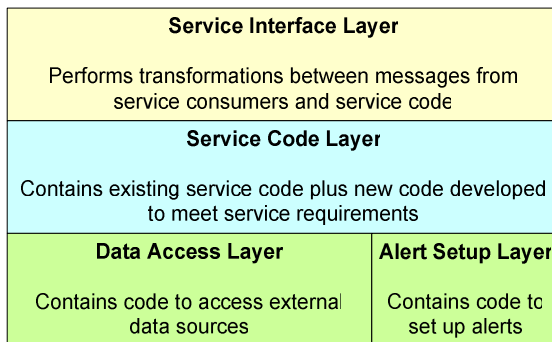
Given the identified migration issues and preliminary estimates of cost and risk, the following migration strategy was developed.

1. Define scope of initial migration for short-term feasibility demonstration. During the *Analyze the Gap* activity with developers of both the legacy system and the service consumer system, different options for migration for the short-term feasibility experiments were presented, as shown in Table 2.

**Table 2.** Alternatives for Short-Term Feasibility Demonstration

Migration Alternative	Effort (person-weeks)
1. Implement SetTaskAlert service using a Query Language package developed for use in another system (Option 2)	24
2. Implement SetTaskAlert service using functionality in the legacy system (Option 1)	20
3. Do not implement the SetTaskAlert service	11
4. Do not implement the SetTaskAlert service and do not separate out from PS	7

2. Define scope of subsequent iterations.
3. Finalize service inputs and outputs. Concretely define the service inputs and outputs from the Service Table need to be concretely in WSDL documents.
4. Gather information about the publish-subscribe component to be used as the mechanism for alert capability.
5. Create a service reference architecture. A service reference architecture to be followed by all services provides a framework for service development, the reusability of common service operations, and the isolation of service code from changes due to the differences between short-term and long-term goals for MSS. An example of a service reference architecture is shown in Figure 2.
6. Adjust Estimates. After making final decisions on scope, inputs, outputs and requirements are further defined, and the estimates are adjusted.



**Fig. 2.** Service Reference Architecture for MSS Services

7. Create MSS Services using the Service Reference Architecture.
8. Document Lessons Learned.

### 3.8 Highlights of Results

The organization implemented alternative 4 from the list of recommended alternatives in Table 2. This option included developing the following services: AvailablePlans, TrackedTasksPerPlan, and TaskStatus. This option also did not implement the Set-TaskAlert service because the event processing technology was still immature, migration issues from the application of SMART-MP highlighted a set of risks that were not justifiable to take at this point, and the organization had a short schedule on which to complete an initial demonstration.

The three services were implemented successfully on an experimental SOA infrastructure using the selected Web Services infrastructure in less than the estimated time. The SMART-MP engagement had estimated that these services could be developed in 7 weeks. The services actually took about 5 weeks to develop. The quicker than anticipated schedule can be attributed to the strong familiarity of the developers with the original code, the fact that the high risk items were deferred, and to the fact that the analysis from the SMART-MP process was more detailed than they had anticipated, and as a result it served as a blueprint for the development of the services.

The services may be re-hosted on another target SOA infrastructure in the future. Because the organization isolated the service code layer, data access layer and service interface layer, as recommended in the migration strategy, potential future re-hosting to a different SOA infrastructure may be accomplished without starting all over again.

## 4 Evolution of the SMART Family

SMART has been applied in four different organizations across six projects. In dealing with actual SMART engagements, we found that while SMART-MP provides support for helping an organization plan an initial SOA strategy and pilot implementation, there was a range of other types of needs that organizations have in getting started with SOA.

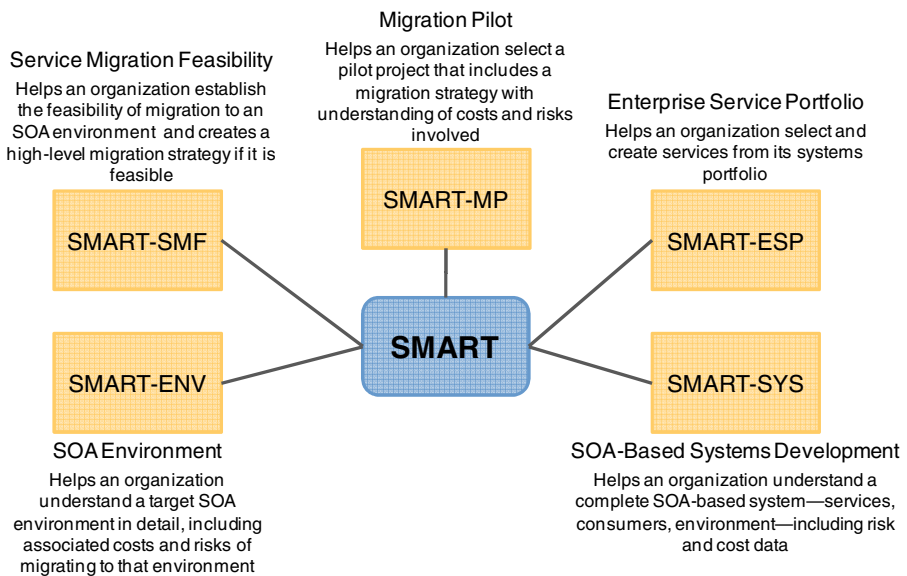
As a result, we began to identify different members of the SMART Family to help organizations that are dealing with different sets of issues. To address these broader issues we tailored SMART by extending the SMIG interview questions while maintaining the same basic structured process. The SMART Family is the formalization of this tailoring into a family of SOA techniques that practitioners can selectively employ depending on how far along they are in their SOA implementation. All members of the SMART Family follow the same generic process illustrated in Figure 1. What varies between members is the emphasis placed on each of the steps, the set of questions extracted from the overall SMIG, and the outputs of each step. The SMART family is illustrated in Figure 3.

**SMART-MP** (Migration Pilot) is the original member of the SMART family that has been described earlier in this report. It identifies a pilot project that will help shape a

migration strategy for an organization, along with an understanding of cost and risk involved moving forward.

**SMART-SMF** (Service Migration Feasibility) is targeted at organizations that are new to SOA and are probably not ready for a pilot project. The end goal is to determine if it makes sense for the organization to adopt SOA, to understand migration options, and to start putting together a migration strategy that may include the execution of other members of the SMART Family.

**SMART-ESP** (Enterprise Service Portfolio) is targeted at organizations that want to take a look across all their legacy systems to identify potential services. The end goal is the creation of an enterprise service portfolio and the mapping of these services to legacy systems.



**Fig. 3.** SMART Family

**SMART-ENV** (Environment) is targeted at organizations that have identified a target SOA environment (or have been mandated a target SOA environment) but do not understand the implications of migrating to this environment. The end goal is the characterization of the target SOA environment, including preliminary costs and risks of migrating to that environment.

**SMART-SYS** (System) is targeted at organizations tasked with the development of a complete service-oriented system that potentially includes the identification and creation of services, the development or acquisition of a SOA infrastructure, and the development of service consumers. The end goal is a superset of the goals of the previous SMART family members.



## 5 Conclusions and Next Steps

SOA offers significant potential for leveraging investments in legacy systems by providing a modern interface to existing capabilities, as well as exposing legacy functionality to a greater number of users. In migrating legacy components to services, there is a need for detailed analysis to determine the feasibility of exposing legacy functionality as services. This analysis has to include the identification of needs of the target SOA environment, a clear distinction between the needs that can be satisfied by the legacy system and those that cannot be satisfied, and a systematic analysis of changes that need to be made to fit into the target SOA environment.

SMART-MP analyzes the viability of reusing legacy components as the basis for services by answering these questions:

- Does it make sense to migrate the legacy system to services?
- What services make sense to develop?
- What components can be used to implement these services?
- What changes to components are needed to accomplish the migration?
- What would be an ideal pilot project?
- What migration strategies are most appropriate?
- What are the preliminary estimates of cost and risk?

SMART-MP provides an approach to collect data that will enable answering these questions. This paper has presented the evolved SMART process as well as the results of its application to a specific real world experience. The paper outlines new members of the SMART family that have been developed as a result of specific engagements. Details and experiences with the application of these family members will be presented in future work.

## 6 Related Work

There is a large amount of work related to migration of systems to SOA environments. SMART falls in the category of techniques that take a strategic approach to SOA adoption, in which business goals and drivers are taken into consideration when making migration decisions to a SOA environment. Other related work includes:

- Ziemann et. al. agree that “rather than being of a purely technical nature, the challenges in this area are related to business engineering: How can a sub-functionality be identified as a potential service, or how can business process models be derived from a legacy system.” They propose a business-driven legacy-to-SOA approach based on enterprise modeling that considers both the business and legacy system aspects [11].
- IBM has a method called Service Oriented Modeling and Analysis (SOMA) that focuses on full system development but has some portions that address legacy reuse: “*SOMA facilitates integration with techniques for analyzing legacy applications, custom and packaged, to identify, specify and realize services for use in a service-oriented architecture. It breaks out the business functions of each existing application, identifying candidate services that can be used to realize business*

*goals under the new architecture. It also identifies potentially problematic areas and highlights areas where new services need to be developed or sourced from an external provider."* [12].

- Cetin et. al. propose a mashup-based approach for migration of legacy software to pervasive service-oriented computing platforms [5]. The interesting aspect about this work is the inclusion of presentation services, which is not typical. The approach is a combination of top-down, starting from business requirements, and bottom-up, looking at legacy code. Business requirements are mapped to services and integrated through a mashup server, which then eliminates the need for developing specific applications to access the services.

There is also work related specifically to the identification of services in legacy code. For example, in the context of Web Services, Aversano et. al. propose an approach that combines information retrieval tracing with structural matching of the target WSDL with existing methods [13]. It performs library schema extraction and then feature extraction to build a WSDL document from the legacy code. Then, it compares the generated WSDL document with the target WSDL document using structural matching. Also in the context of Web Services, Sneed proposes an approach that consists of salvaging the legacy code, wrapping the salvaged code and making the code available as a web service [14]. In the salvaging step he proposes a technique for extracting services based on identifying business rules that produce a desired result.

## References

1. Chung, S., Young, P., Nelson, J.: Service-Oriented Software Reengineering: Bertie3 as Web Services. In: Proceedings of the 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA, July 11-15, 2005. IEEE Computer Society, Los Alamitos (2005)
2. Polmann, M., Schonefeld, M.: An Evolutionary Integration Approach using Dynamic CORBA in a Typical Banking Environment. In: The Case Studies Workshop of the Sixth European Conference on Software Maintenance and Reengineering, Budapest, Hungary, March 11-13 (2002)
3. Radha, V., Gulati, V., Thapar, R.: Evolution of Web Services Approach in SFMS – A Case Study. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2004), San Diego, CA, USA, July 6-9 (2004)
4. Zhang, J., Chung, J., Chang, C.: Migration to Web Services Oriented Architecture – A Case Study. In: Proceedings of the 2004 ACM Symposium of Applied Computing, Nicosia, Cyprus, March 14 -17, 2004. ACM Press, New York (2004)
5. Morisio, M., Ezran, M., Tully, C.: Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering 28(4), 340–357 (2002)
6. Lewis, G., Morris, E., Smith, D., Simanta, S.: SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment, CMU/SEI-2008-TN-008, Software Engineering Institute (May 2008)
7. Lewis, G., Morris, E., Smith, D.: Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture. In: Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR 2006), Bari, Italy, March, 22 (2006)

8. Kazman, R., O'Brien, L., Verhoef, C.: Architecture Reconstruction Guidelines, 2nd edn., CMU/SEI-2002-TR-034, Software Engineering Institute (November 2003)
9. O'Brien, L., Stoermer, C., Verhoef, C.: Software Architecture Reconstruction: Practice Needs and Current Approaches, CMU/SEI-2002-TR-024, Software Engineering Institute (August 2002)
10. Violino, B.: "How To Plan for SOA 2.0", Baseline [online magazine], (March 8, 2007) (cited April 27, 2007),  
<http://www.baselinemag.com/article2/0,1540,2102088,00.asp>
11. Ziemann, J., Leyking, K., Kahl, T., Werth, D.: SOA Development Based on Enterprise Models and Existing IT Systems. In: Cunningham, P. (ed.) Exploiting the Knowledge Economy: Issues, Applications and Case Studies. IOS Press, Amsterdam (2006)
12. IBM Business Consulting Services. IBM Service-Oriented Modeling and Architecture (2005),  
<http://www-935.ibm.com/services/us/gbs/bus/pdf/g510-5060-ibm-service-oriented-modeling-arch.pdf>
13. Aversano, L., Di Penta, M., Palumbo, C.: Identifying Services from Legacy Code: An Integrated Approach. Presentation at the Working Session on Maintenance and Evolution of SOA-Based Systems (MESOA 2007). In: 23rd IEEE International Conference on Software Maintenance (ICSM 2007), Paris, France, October 4 (2007)
14. Sneed, H.: Integrating Legacy Software into a Service Oriented Architecture. In: Proceedings of the 10th European Conference on Software Maintenance (CSMR 2006), March 22-24, 2006. IEEE Computer Society Press, Bari (2006)

# Discovering and Deriving Service Variants from Business Process Specifications<sup>\*</sup>

Karthikeyan Ponnalagu and Nanjangud C. Narendra

IBM India Research Lab, Bangalore, India  
{karthikeyan.ponnalagu,narendra}@in.ibm.com

**Abstract.** Software service organizations typically develop custom solutions from scratch in each project engagement. This is not a scalable proposition, since it depends too heavily on labor alone. Rather, creating and reusing software “assets” is more scalable and profitable. One prevalent approach to building software solutions is to use service-oriented architecture (SOA) to compose software services to support business processes. In this context, the key to reusing assets is to discover (from existing assets in a portfolio) or derive service variants to meet the requirements of a stated business process specification. To that end, this paper presents our Variation-Oriented Service Design (VOSD) algorithm for the same. Via IBM’s Rational Software Architect modeling tool, we also demonstrate the practical usefulness of our algorithm via a prototype implementation in the insurance domain.

**Keywords:** Service-oriented Architecture, Business Process, Reuse.

## 1 Introduction and Motivation

Software service organizations developing custom business solutions are being faced with the increased need to effectively reuse existing assets and thereby enhance profitability. The emergence of service oriented architecture (SOA) [5], with its emphasis on loose coupling and dynamic binding of services, promises to enable more effective reuse by developing business processes as loosely-coupled collections of services modeled as reusable assets. However, one major obstacle against the realization of this vision is the cost involved in modeling and manipulating service variants in order to meet varied business process specifications. Currently this practice is carried out manually, making it cumbersome and error-prone. To that end, in this paper we present an algorithm called Variation Oriented Service Design (VOSD), which automatically discovers (from reusable assets in a portfolio) and/or derives (in case the reusable asset is not available) service variants from a portfolio of existing services to meet a stated business process specification. Our approach leverages from our earlier Variation-Oriented Engineering methodology (VOE) [3], which is an end-to-end approach spanning business processes to their SOA implementation to formally model and develop these variants, so that the reuse of solutions with variants can be facilitated.

---

<sup>\*</sup> Thanks to Dipayan Gangopadhyay, Biplav Srivastava and Islam Elgedawy for their feedback.

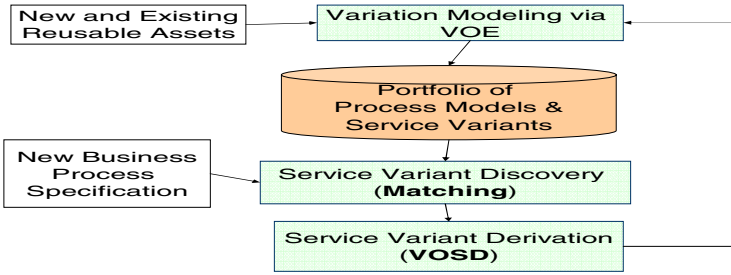


Fig. 1. Integrated Service Variant Discovery and Derivation Approach

This paper is an extension of an earlier paper [7], which introduced our basic VOSD algorithm. That paper, however, only focused on deriving service variants from stated business process specifications; in this paper, we extend the algorithm in [7] to the problem of selecting the appropriate (variant of) services in a portfolio, said variants being stored in a manner consistent with VOE principles. Service variant derivation is resorted to only if the appropriate variant is not available. We also demonstrate that this extension is non-trivial, and show how the meta-modeling approach in VOE is able to assist in automatic service variant discovery. Our overall approach is illustrated in Figure 1. Throughout the paper, we illustrate our algorithm on a simple yet realistic running example (drawn from a real-life project engagement) in the insurance domain. We also demonstrate our algorithm on a prototype implementation using IBM’s Rational Software Architect (RSA) modeling tool, thereby demonstrating its practical usefulness (please note, however, that our algorithm is independent of the modeling tool used).

This paper is organized as follows. We will first describe our running example. We then describe some preliminary concepts on which our integrated approach is based. We then describe our integrated VOSD discovery + derivation algorithm. We will then illustrate our algorithm on the running example via the RSA tool (however, our algorithm is not dependent on RSA, and can be implemented on any UML-based modeling tool). The paper will finally conclude with discussions on related & future work.

## 2 Running Example

Let us assume a basic insurance claims process solution, Sol1, which has been implemented for a customer, as in Figure 2. The net output of the execution of this process is an evaluation of the applicant’s claim; if the claim is accepted, the output would also include information on the payment to be made to the applicant. For this paper, we focus on the **Verify Claim** sub-process; in this sub-process, the *DetermineLiability* and *PotentialFraudCheck* services are first executed in parallel, and send their results to Claim Investigation service. A final review of the verified claim is then implemented by *FinalReview* service.

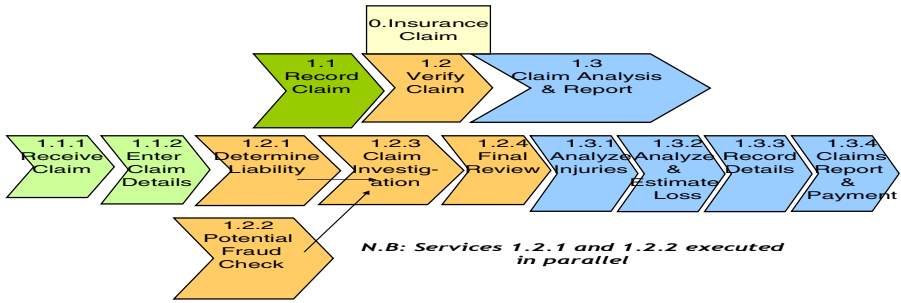


Fig. 2. Basic Insurance Claims Process – Solution Sol1

Let us assume that the same solution should now be tailored for a second customer to produce Sol2, with the following changed requirements:

- Improve cycle time for **Verify Claim** sub-process – for a new class of “high priority” applicants not previously served
- Improve fraud checking – a new and improved fraud checking module to be incorporated, given the fact these are now high-value claims

Based on these inputs, the following variations to Sol1 are anticipated:

- *DetermineLiability & PotentialFraudCheck* services to be outsourced for improved speed
- Also, *PotentialFraudCheck* service should be modified to take into account extent of liability – this would eventually involve changes in its business logic
- New *Liability+FraudChecks* service to be added

Let us assume that these variations would result in solution Sol2, as shown in Figure 3.

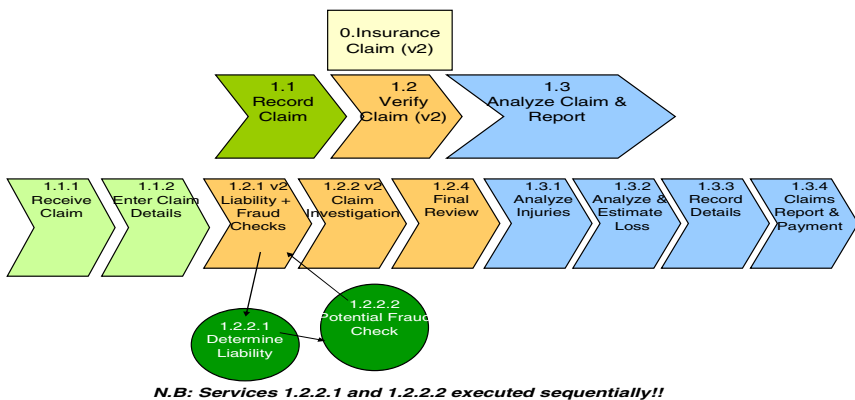


Fig. 3. Modified Insurance Claims Process – Solution Sol2

The original inputs and outputs for the services to be modified in Sol1 are as in Figure 4.

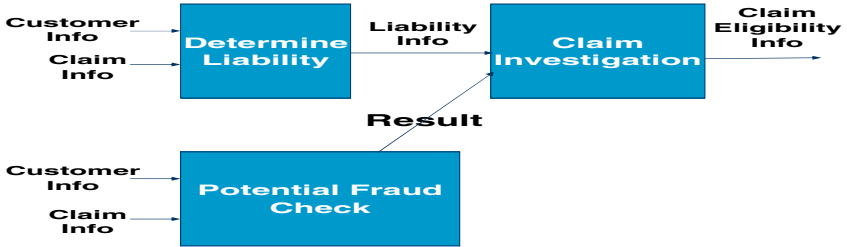


Fig. 4. Original Services from Sol1

The modifications for Sol2 are shown in Figure 5.

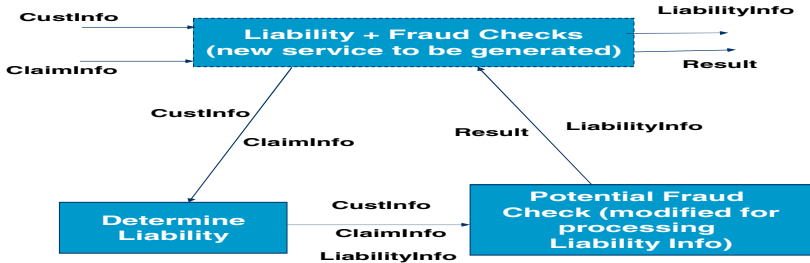


Fig. 5. Modified Services for Sol2

Upon examination of Figures 4 and 5, the following question arises: how do we decide that the *DetermineLiability* and *PotentialFraudCheck* services need to be sequentialized while they were originally executing in parallel? Also, how do we determine what the inputs and outputs of *Liability+FraudChecks* service should be? Current practice would be to determine the answers to these questions based on a combination of visual inspection (perhaps based on modeling the business processes in a process modeling tool) and manual calculation.

From a software engineering perspective, however, such an approach is cumbersome and time-consuming, and hence does not scale to large business processes. Additionally, such a calculation is in general non-trivial. For example, if a service  $S_k$  were to be sequentially inserted between  $S_i$  and  $S_j$ , this may involve the introduction of new control and data dependencies among  $S_i$ ,  $S_j$  and  $S_k$ . The question that then arises, is how should the functionality of  $S_i$  and  $S_j$  be modified? That is, some outputs of  $S_i$  may need to be redirected to  $S_j$ ; also, some inputs of  $S_j$  may have to be obtained from  $S_k$ . The introduction of  $S_k$  may also create new outputs, which would have to be redirected to  $S_j$ , thereby necessitating an enhancement of the functionality of  $S_j$ , i.e., the source code of the service  $S_j$  will need to be modified. This can be observed from

the example of inserting the *Liability+FraudChecks* service between *DetermineLiability* and *PotentialFraudCheck* services. Now, when the number of such business process-level changes grows, determining all the service-level code changes becomes a major design exercise that can only be eased and speeded up via automation. Hence the need for an algorithm such as VOSD.

### 3 Preliminaries

#### 3.1 Meta Model-Based Representation of Variations

We leverage the metamodel introduced in [3] that allows one to separately model the static and variable parts of any software component (service or business process) for our VOSD algorithm. This metamodel consists of the following parts, and is depicted in Figure 6:

- *Variation Points* - these are the points in the component where variations can be introduced. They are in turn of two types. *Implementation variation points* are the points in the component where the implementations of certain methods can be modified, without affecting the externally observable behavior of the component. Whereas, *specification variation points* are the points at the interface of the component which can be modified. This may necessitate changes to the internal implementation of the component, which are specified via implementation variation points. Specification variations could therefore involve adding new input/output data, and/or removing input/output data to the component.
- *Variation Features* – these further refine variation points, by specifying the action semantics of the variation and its specific applicability. The same variation point can admit more than one variation feature, and one variation feature can be applied to many variation points.

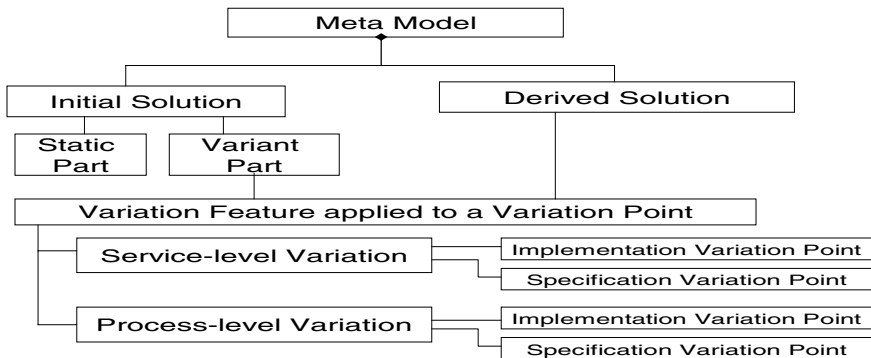


Fig. 6. VOE Meta-Model



This meta model will be further specialized to instantiate conceptual models for modeling service-level and business process-level variations. These conceptual models can then be treated as design templates from which actual variation-oriented design can be accomplished. Hence for our running example, an example of a variation point would be a method in *DetermineLiability* service for calculating insurance liability. A variation feature would be an action to replace that method by a different method. The actual service variant would be the modified *DetermineLiability* service.<sup>1</sup>

### 3.2 Modeling Business Process and Service Level Variations

Based on the metamodel from [3], we broadly categorize variations in a business process-based solution into two cases. *Service-level Variations* are variations at the level of individual services. They can in turn be classified into two sub-cases.

- Service implementation (ServImpl) variations model changes only to the internal service implementations, without requiring changes to their interfaces, and this is realized via implementation variation points. An example would be to modify the internal fraud checking method in *PotentialFraudCheck* service.
- Service Interface (ServIntf) variations model changes to the interfaces of the services, which will also require implementation changes – this is realized via specification variation points. An example would be to add the outputs *custInfo* and *claimInfo* to *DetermineLiability* service. This would also require concomitant ServImpl variations, as follows:
  - Input data received by the service – this could arise due to the following triggers. First, a change in the output data sent by a previously executed service, which is to be consumed by the service in question; second, a change in the input data needed by a service to be executed later - this data may have to be transmitted by the service in question, perhaps after modification.
  - Output data sent by the service - this would be a trigger for modifications to the services to be executed next, i.e., those that are dependent on the service in question.

*Process-level variations* are variations in the application flow of the business process. These are realized via combinations of ServImpl and ServIntf variations, and can be classified as follows:

- AddSeqSvc: a service  $S_j$  is added between  $S_i$  and  $S_k$ . If this does not cause any modifications to the inputs of  $S_k$  and outputs of  $S_i$ , then the output methods of  $S_i$  and input methods of  $S_k$  have to be redirected towards  $S_j$  – this can be realized as ServImpl variation, since this will involve modifying the input and output methods for the services  $S_i$  and  $S_k$ . However, if modifications are required, then this will be realized as a ServIntf variation on  $S_i$  and/or  $S_k$ , and needs to be modeled as such. An example is to add a suitably modified *PotentialFraudCheck* service between *DetermineLiability* service & *Liability+FraudChecks* service.

---

<sup>1</sup> Please note that our approach resembles inheritance-based variations from the object-oriented (OO) domain only in the implementation variation points; otherwise, the well-known “open to extension - closed to modification” principle prevalent in the OO domain does not apply for specification variation points.

- **DeleteSvc:** service  $S_{i+1}$  (predecessor is  $S_i$  and successor is  $S_{i+2}$ ) is deleted. If this does not cause any modifications to the outputs of  $S_i$  or the inputs of  $S_{i+2}$ , then the output methods of  $S_i$  and input methods of  $S_{i+2}$  need to be redirected towards each other. However, if modifications are required, this will require **ServIntf** variation on  $S_i$  and  $S_{i+2}$ , and needs to be modeled as such.
- **AddParSvc:** Add service in parallel – a service  $S_j$  is added between  $S_i$  and  $S_k$  in parallel – this would require **ServIntf** variations to  $S_i$  and  $S_k$ . If this does not cause any modifications to the inputs of  $S_k$  and outputs of  $S_i$ , then additional methods would need to be added to each service to accommodate the new service  $S_j$  – this would be an **ServImpl** variation. However, if modifications are required, then this will be realized as **ServIntf** variations on  $S_i$  and  $S_k$ , and needs to be modeled as such.
- **AddFlow:** Add dependency between two services – akin to adding an edge in the business process – this will be a **ServIntf** variation, requiring interface changes to the services.
- **DelFlow:** Delete dependency between two services – akin to deleting an edge in the business process – this is similar to **AddFlow**, in that it would require a **ServIntf** variation.

## 4 Integrated VOSD Algorithm

Before we describe our algorithm, we first informally introduce a process and its associated services, before giving a formal definition. Briefly, a process is defined as consisting of four parts: (a) a set of associated service models, (b) data dependencies between the services based on the execution of preceding services (also called *produced data dependencies*), (c) data dependencies between the services based on the input model of preceding services (also called *received data dependencies*), and (d) control flow dependencies that provides the choreography of the services. Please note that this definition is quite realistic from a business perspective. Most business analysts in software organizations (who are not expected to possess programming expertise) would typically represent a business process as a collection of services with their respective inputs and outputs, and then augment it with control flow and data dependencies among pairs of services that do not share the predecessor/successor relationship. Therefore, such business analysts would also find it easy to distinguish between received and produced data dependencies, which is crucial for our algorithm (as we will see later in this section).

### 4.1 Formal Process and Service Model

We define a **process**  $P = \{S, E, D, C\}$ , wherein

$S = \{S_1.. S_n\}$  is the set of services that participate in  $P$

$S \subset U$ , where  $U$  is the total portfolio of services (and their associated variations) for that particular domain of  $P$ .

$E = \forall ij \{E_{ij}\}, \{ \text{iff } S_i \xrightarrow{e_{ij}} S_j = \text{true} \}$ , is the set of all *produced data dependencies* of service  $S_j$  with  $S_i$ , where  $i \neq j$

$E_{ij}$ , with the value of true, lets us know that the data produced by the execution of  $S_i$ , irrespective of  $S_i$ 's input is part of the data  $d_{ij}$  that needs to be passed from  $S_i$  to  $S_j$ .

$D = \forall ij \{D_{ij}\}, \{ \text{iff } S_i \xrightarrow{d_{ij}} S_j = \text{true} \}$ , is the set of all *received data dependencies* of service  $S_j$  on  $S_i$ , where  $i \neq j$ . Here Service  $S_i$  needs to pass its input data *without modifications* to Service  $S_j$ . This is represented by  $D_{ij}$  with the value of true.

The distinction between produced and received data dependencies is *absolutely crucial*, since this will help in the case of removal of  $S_i$  from the process flow  $P$  or sequential inclusion of a new Service  $S_x$  between  $S_i$  and  $S_j$  (i.e., DelSvc- and AddSeqSvc-type variations – see Section 3.2), as we will illustrate later in this Section with the help of the running example of Section 2.

$C = \forall ij \{C_{ij}\}, \{ \text{iff } S_i \xrightarrow{c_{ij}} S_j = \text{true} \}$ , where  $i \neq j$ , is the set of *control flow dependencies* between  $S_i$  and  $S_j$ , where  $C_{ij}$  is either true or false, based on whether  $S_i$  controls the execution of  $S_j$ , i.e., iff  $S_i$  precedes  $S_j$  in terms of control flow.

Only with the value of  $C_{ij}$  being true, can we know that a service  $S_i$  precedes the Service  $S_j$  in the business process. Only with this information, can we expect Service  $S_i$  to pass on the received data  $D_j$  for Service  $S_j$  from Service  $S_j$ 's predecessor services. On the other hand,  $C_{ij}$  having a value of false means  $S_j$  would execute in parallel with respect to  $S_j$  and  $S_i$  is not expected to carry any additional data for  $S_j$ .

We also define a **service**  $S$  via its inputs and output sets respectively, i.e.

$S = \{D_{in}, D_{out}\}$ , where  $\{D_{in}\} = \text{Set of input Data required for invoking } S$ , and  $\{D_{out}\} = \text{Set of Output Data expected after invoking } S$ .

To illustrate via our running example, from Figure 2, let Start Service =  $S_0$ , which is the initial service that provides all the inputs for subsequent services; let *DetermineLiability* =  $S_1$ , *PotentialFraudCheck* =  $S_2$ , *ClaimInvestigation* =  $S_3$ , *Liability+FraudChecks* =  $S_4$ . The service definitions for our running example are listed here below.

$S_0 = \{D_{in} = D_{out} = \{\text{CustInfo, ClaimInfo}\}\}$   
 $S_1 = \{D_{in} = \{\text{CustInfo, ClaimInfo}\}, D_{out} = \{\text{LiabilityInfo}\}\}$   
 $S_2 = \{D_{in} = \{\text{CustInfo, ClaimInfo}\}, D_{out} = \{\text{Result}\}\}$   
 $S_3 = \{D_{in} = \{\text{LiabilityInfo, Result}\}, D_{out} = \{\text{LiabilityInfo, Result}\}\}$

Let us consider Figure 4 (for original process) and Figure 5 (for variant process) and only those services highlighted in those two diagrams for illustration. Let us refer to the processes, respectively, as  $P_{old}$  and  $P_{new}$ . Hence the process in Figure 4 can be formally modeled as:

$$P_{old} = \{S, E, D, C\}$$

Where  $S = \{S0, S1, S2, S3\}$  //used to derive new services or contains variants

$$E = \{e_{13}, e_{23}\}, \text{ where } e_{13} = e_{23} = \text{true}$$

$$D = \{d_{01}, d_{02}\}, \text{ where } d_{01} = d_{02} = \{\text{Custinfo, Claiminfo}\}$$

$$C = \{c_{01}, c_{02}, c_{13}, c_{23}\}$$

$$c_{01} = c_{02} = c_{13} = c_{23} = \text{true}$$

With respect to D in the original process (i.e., received data dependencies), Services S1 and S2 are dependent on S0 via received data and control dependencies (i.e., S1 and S2 can execute only upon successful execution of S0). Similarly Service S3 is dependent on both S1 and S2 via received data, produced data and control dependencies.

Now let us look at the process depicted in Figure 5, i.e., the variant process. It can be represented as below.

$$P_{new} = \{S, E, D, C\}$$

Here,  $S = \{S0, S1, S2, S3, S4\}$  // we need to derive or discover these

$$E = \{e_{12}, e_{24}\} \text{ Where } e_{12}, e_{24} = \text{true}$$

$$D = \{d_{04}, d_{41}, d_{12}, d_{24}, d_{43}\}$$

$$d_{04} = d_{41} = \{\text{custinfo, claiminfo}\}$$

$$d_{12} = \{\text{custinfo, Claiminfo, liabilityinfo}\}$$

$$d_{24} = d_{43} = \{\text{liabilityinfo, result}\}$$

Similarly  $C = \{c_{04}, c_{41}, c_{12}, c_{24}, c_{43}\}$

Where  $c_{04}, c_{41}, c_{12}, c_{24}, c_{43} = \text{true}$

## 4.2 Algorithm Description

We assume the following inputs to our VOSD algorithm: New Process Model = NP, Functionally closer, existing process Model from portfolio = OP, Associated set of services for OP from the portfolio = S, set of process tasks mapped from NP to services in S =  $S_{mod}$ , set of process tasks *not* mapped from NP =  $S_{new}$ , and finally, variation models for all the services in the portfolio (modeled as per VOE principles from Section 3.2).

Before we present the algorithm, however, we first describe a function called **Matching** which implements service variant discovery from existing service variants in the portfolio. This function will be invoked by the VOSD algorithm as its first step. The inputs to **Matching** are: (i) the selected task in NP for which the appropriate service variant needs to be determined, (ii) a selected service variant S' in OP (which belongs to the set  $S_{mod}$ ) which needs to be matched against (i). The **Matching** function returns one of the following outputs: either (a) FALSE, i.e., no match occurs; or (b) TRUE, along with the appropriate (variation point, variation feature) pair of the selected service variant.

**BEGIN Matching**

```

Variation Model  $V_m \leftarrow$  new VariationModel ( $S'$ );
//Lists all variation points and variation features applicable to  $S'$ 
For each Variation Point
{
    For each Variation Feature applicable at that Variation Point
    {
         $V_m.getVariant(VariationPoint)$ ;
//Choose the variation point corresponding to the service variant.
        If (No such variation point exists)
        {
            return ("FALSE");
            exit(-1);
        }
         $V_m.getVariantFeature(VariationPoint, Variant)$ ;
// Check whether there is an actual service variant available in the portfolio of reusable
assets. If so, this function returns the actual variation feature, or exits with failure.
        If (No such variant is available)
        {
            return ("FALSE");
            exit(-1);
        }
    }
}
return (ServiceVariant, VariationPoint, VariationFeature);
//Finally returns the service variant along with its variation point and variation feature
END Matching

```

**Algorithm: VOSD**

1. Pick a task from NP. Verify and compare the input and outputs for it and its corresponding identified Service in  $S_{mod}$
2. If **Matching**, go to Step 1 for next task. If this is the Last Task then Exit. Otherwise, go to Step 3.
3. Get the service Variation Model. Find the associated Variation Points and Variants for that Service based on the conflict.
  - a. Ensure the service is declared Variant (VP exists)
  - b. Ensure the associated Operation is declared Variant (VP exists).
  - c. If neither a nor b occurs, exit with message ("Variation Failed"). If no Variation model available (this will be the case for new services in  $S_{new}$ ), go to step 5.
4. Select the Variant from the selected list in step 3, that obeys the following properties.
  - a. it addresses the change identified from NP
  - b. it matches on the listed Variation Features

If Variant exists, return ("Variant Available"); else go to step 5
5. Invoke **DeriveServices** //This is the service variant derivation step

At the end of execution of this algorithm, either we will get an implemented service variant from the portfolio or a skeleton variant related to an existing implementation through the algorithm **DeriveServices**.

**Algorithm: DeriveServices**

(i) Get  $P_{new}, P_{old}$ ,

Service  $S1[] = P_{new}.getS()$ ;

// Abstract Services , links to S2.

Service  $S2[] = P_{old}.getS()$ ;

// Returns Actual Services

Service  $S_{new}[] = S2[] - S1[]$

// Services to be created for  $P_{new}$

Service  $S_{old}[] = S1[] \cap S2[]$  // Services to be modified

(ii) **createNewServices**( $P_{new}, S_{new}$ )

(iii) **deriveServiceVariants**( $P_{new}, P_{old}$ )

(iv) **InstantiateServices**( $S_{new}[]$ )

We will elaborate steps (ii), (iii) and (iv) of the **DeriveServices** algorithm here. **CreateNewServices**() first creates all the new services that are required in the new process. By creation of services, we mean a consolidated list of references with respect to other service's inputs and outputs. This approach is required, as at this point of time, we expect many of the services to go through a variation phase which will impact their input/output interface models to suit the new process. Hence once all the variations are identified for all the services, the input/output model can be used for actual instantiation of the corresponding services, which we will see later in this section. The **CreateNewServices**() algorithm is given below.

**createNewServices**(Process  $P_{new}$ , Service[]  $S_{new}$ )

//Get Each  $S_{new}[i]$  for  $i = 1 \dots n$ ,

// where n is the total no. of services

Service  $S = S_{new}[i].getNext()$  //

$S = CreateService(S, P_{new})$

**CreateService**( $S, P_{new}$ ) //Creates new services

Service  $Ssource[] = P_{new}.getE().getAllSource(S)$ ;

//returns all services, whose execution, Service S depends on

for all Services  $Ssource$   $i = 1..n$ .

If( $Ssource[i] \notin S_{old}[]$ ) continue;

**AddProducedinputs**( $S_{new}[i], Ssource[i]$ )

Service  $Ssource[] = P_{new}.getD().getAllSource(S)$

for all Services  $Ssource$   $i = 1..n$  {

    If ( $Ssource[i] \notin S_{old}[]$ ) continue;

**AddReceivedinputs**( $S_{new}[i], Ssource[i]$ );

Now let us discuss the step **DeriveServiceVariants**, in which all the existing services of old process that are required in the new process are first verified for need of variation and appropriately their input/output model is defined accordingly. For this also, we consider the values of E and D of the process  $P_{new}$ . One thing worth mentioning is, that an iteration on E, will affect only the dependent services, while an iteration on D can affect both the services corresponding to the variable  $D_{ij}$ . This step is elaborated here.

```

Int n = Enew.getSize(); Int j = Dnew.getSize();
For (int i = 0; i<n;i++)
{
    Service S = Enew[i].getSource();
    Service t = Enew[i].getTarget();
    // Each value of Enew[i], represents a class that supports methods that returns the source
    and target services
    If (Snew.contains(t)) continue; //Already created
    Else // ModifyService
    AddProducedInputs(t,s);
    // Add data that are produced by service s to the input list of service t
}
For (int i = 0; i<j;i++)
{
    Service S =Dnew[i].getSource();
    Service t = Dnew[i].getTarget();
    If (Snew.contains(t)) continue; //Already created
    Else // ModifyService
    AddReceivedInputs(t,s);
    // Add data that are received as Inputs by s to the input list of t and output list of d
}
}

```

The final step of the algorithm is InstantiateServices(), which actually instantiates the services of the new process P<sub>new</sub>. This is illustrated below, along with two methods for adding received and produced inputs as per E and D dependencies, respectively.

```

addreceivedinputs(t,s)
{
    s.addoutputs(s.getinputs());
    // This method in the source service Structure s - adds its input data into its Output Model.
    t.addinputs(s.getinputs(t));
    // This method gets only those inputs that the target service t does not have
    // for example S4 just wants liabilityinfo from S2 and not Custinfo,cliaminfo
}
addproducedinputs(t,s)
{
    t.addinputs(s.getoutputs(s));
    // add only those data that are produced by s; for example liabilityinfo produced by S1 is
    added but not custinfo and claiminfo.
}
InstantiateServices(Service Snew[])
{
    // Here we actually derive service variants based on input output list consolidated with above steps
    int n = s2.getSize();
    for (int k = 0; k <n; k++)
    {
        create(S2[k]);
        // This method retrieves the Input and Output Model for each of the Service
        S2[k] and instantiates the references based on other services
    }
}

```

## 5 Illustration on the Running Example

Referring to our running example, this Section describes a prototype implementation to the illustration of our algorithm. The prototype was implemented as a transformation plugin in IBM's Rational Software Architect v7.0.

Recall that we have declared earlier the following for the associated services for Verify claim: Start Service = S0, *DetermineLiability* = S1, *PotentialFraudCheck* = S2, *ClaimInvestigation* = S3, *Liability+FraudChecks* = S4. Let us first start with the illustration on Service S4. From step 1,3 and 5 in **VOSD** and from step (i) in **DeriveServices**, we know that  $S_{new} = \{S4\}$ . As we mentioned in Section 4, we can define S4 in terms of ( $D_{in}$ ,  $D_{out}$ )

With respect to Step (ii) in **DeriveServices** algorithm, this needs to be created based on the execution/data dependencies:  $E = \{e_{12}, e_{24}\}$  Where  $e_{12}, e_{24} = \text{true}$ .  $D = \{d_{04}, d_{41}, d_{12}, d_{24}, d_{43}\}$ .  $d_{04} = d_{41} = \{\text{custinfo}, \text{claiminfo}\}$ .  $d_{12} = \{\text{custinfo}, \text{Claiminfo}, \text{liabilityinfo}\}$ .  $d_{24} = d_{43} = \{\text{liabilityinfo}, \text{result}\}$ . Hence for S4 as target, the Set E of  $P_{new}$  provides S2, whose execution provides a part of the inputs for S4. The Set D provides two sets of received inputs from Services S0 and S2, which at this point are just existing services. Hence  $D_{in}$  of S4 = (**Received Input of S0, Received Input of S2, Processed data from S2**)

As can be seen here, we do not actually store the values, but only the references of the above variables, as S0 and S2 are still to be modified for the new process flow  $P_{new}$  and the context of S4s creation is for  $P_{new}$  and not for  $P_{old}$ . But as one completes the algorithm and instantiates S4, it will contain the following:  $D_{in}$  of S4 = {**Custinfo, Claiminfo, Liability Info, Result**} that is instantiated from the above equation. Similarly, one can derive the following:  $D_{out}$  of S4 = {**Custinfo, Claiminfo, Liability Info, Result**}. The first two data derived from  $d_{41}$  and last two data derived from  $d_{43}$ . With this definition,  $P_{old}$  consists of the following implemented services.

S0 = {Min, Mout, Mproc}, where Min= Mout = {SetCustinfo(), SetClaimInfo()}, and Mproc = null

S1 = {Min, Mout, Mproc}, where Min = {SetCustinfo(), setClaimInfo()}, Mout = {getLiabilityInfo()}, and Mproc = {processLiabilityInf()}

S2 = {Min, Mout, Mproc}, where Min = {SetCustInfo(), SetClaimInfo()}, Mout = {getResult()}, and Mproc = {fraudcheck()}

S3 = {Min, Mout, Mproc}, where Min = {setliabilityInfo(), SetResult()}, Mout = {getLiabilityInfo(), GetResult()}, and Mproc = {investigateClaim()}

The available Variant Model for the *LiabilityInfo* Service in  $P_{old}$  in the prototype implementation is depicted in Figure 7. From step 2 of the **VOSD** algorithm, Service S2 is declared as a variant. From Step 3 of the algorithm, we can see from Figure 7 that the available variation feature matches with the expected output data model that supports creation of two new methods, with the names of the operations and their associated parameter values confirming the matching. Now, in the case of Service S3, from step 3 of the **VOSD** algorithm and this variation model, we could see that there is no variation feature available. (In practical cases, there is possibility that the available variation features even if any, may not match the expected variant required for



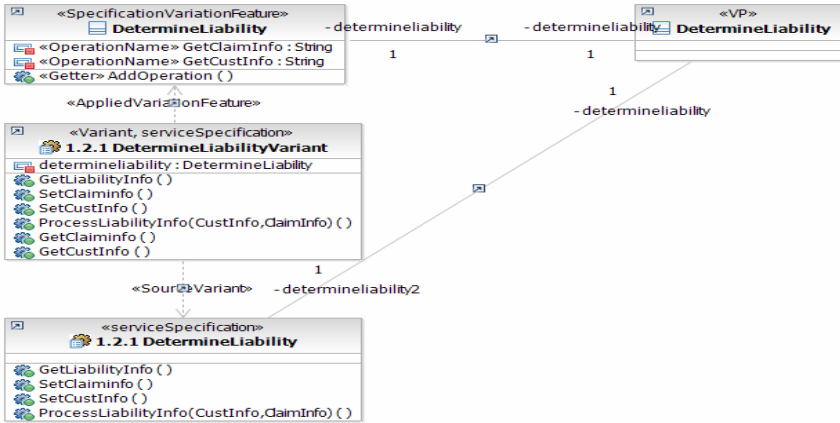


Fig. 7. Variant Model for  $P_{old}$

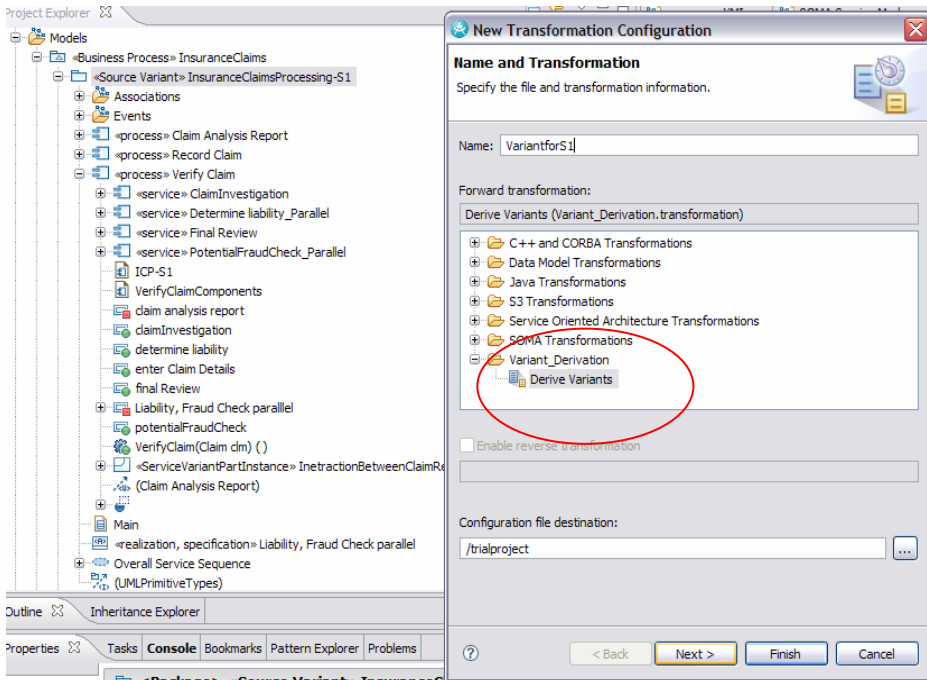


Fig. 8. Transformation Plugin

$P_{new}$ ) Hence from step 4 of the algorithm, we invoke the **DeriveServices** algorithm and derive the new variant for  $S3$ .

Our transformation plugin is depicted in Figure 8, where the user can invoke the service derivation transformation, and subsequently provide the process model  $P_{new}$  as the input to the transformation.

We summarize the services for  $P_{new}$ , below, with variations listed in ***bold-italic*** font:

$S_0 = \{Min, Mout, Mproc\}$ ; where  $Min = Mout = \{SetCustinfo(), SetClaimInfo()\}$  and  $Mproc = null$   
 $S_1 = \{Min, Mout, Mproc\}$ , where  $Min = \{SetCustinfo(). setClaimInfo()\}$ ,  $Mout = \{getLiabilityInfo(), ***getCustinfo()***, ***GetClaiminfo***\}$ , and  $Mproc = \{processLiabilityInf()\}$   
 $S_2 = \{Min, Mout, Mproc\}$ , where  $Min = \{SetCustInfo(), SetClaimInfo(), ***setLiabilityInfo()***\}$ ,  $Mout = \{***getLiabilityInfo()***, getResult()\}$ , and  $Mproc = \{***fraudcheck()***\}$   
 $S_3 = \{Min, Mout, Mproc\}$ , where  $Min = \{setliabilityInfo(), SetResult()\}$ ,  $Mout = \{getLiabilityInfo(), GetResult()\}$ , and  $Mproc = \{investigateClaim()\}$

Figure 9 depicts the output of our integrated VOSD algorithm, displaying  $P_{old}$  and  $P_{new}$ , with  $P_{new}$  (circled in red) named as a variant of  $P_{old}$ .

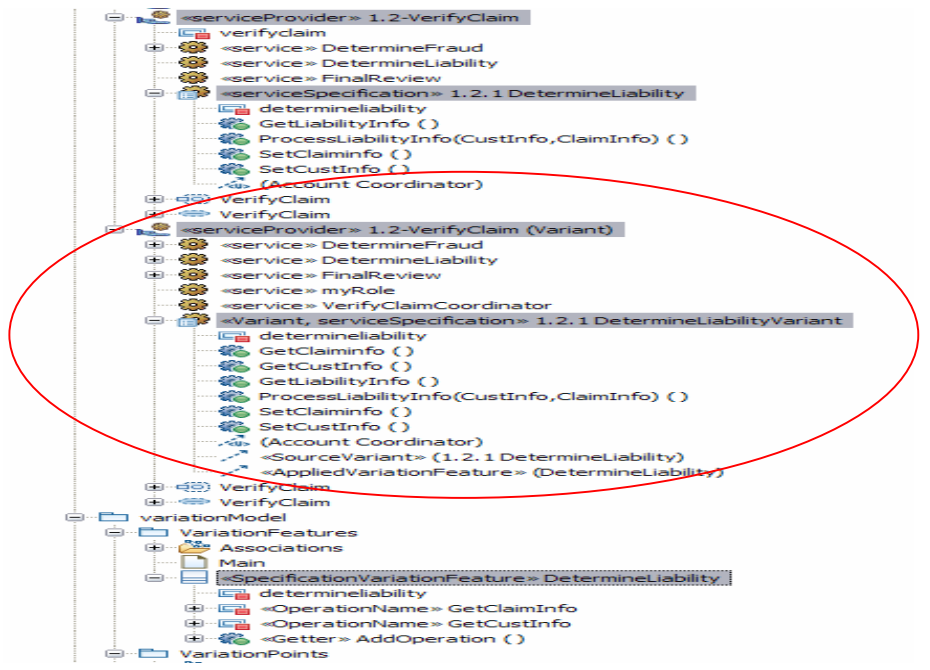


Fig. 9. Output of Integrated VOSD Algorithm

## 6 Related Work

The importance of a systematic approach towards service identification from business process specifications has been highlighted in [8], which bolsters the need for our VOSD algorithm. One of the early formalizations of the variation-oriented analysis and design (VOAD) principle can be found in [1]. Our paper incorporates and extends

these ideas by applying it to the specific case of business process-based solutions and deriving associated service variations. Another relevant paper is [4], which presents a design method called Grammar-Oriented Object Design (GOOD), which provides a means for the business analyst to declaratively specify the requirements of a dynamically reconfigurable software architecture driven by business-process architectures. Another approach similar to ours is presented in [2]; however, that paper primarily concentrates on detailing various ServImpl-type variations expressed via the object-oriented concept of inheritance (i.e., interfaces are considered to be invariant). In contrast, our approach uses SOA principles to provide a far greater range of variability, as already explained earlier in Section 3. In a similar vein, IBM's Model-driven Business Transformation (MDBT) approach [6] proposes a model-driven approach, based on OMG's MDA (<http://www.omg.org/mda/>) approach, for transforming business requirements into IT solutions. Our approach is similar, but we focus on transforming business processes into their constituent service specifications.

## 7 Conclusions and Future Work

In this paper, we have addressed the crucial problem of rapidly comparing an extended template business process (defined for a different context; defined for different customer requirements) with an existing business process that is already implemented as a choreography of collection of services; thereby enhancing reuse of these services by appropriate modification and thereby gaining flexibility with minimal additional effort. As part of this effort, we have described our integrated algorithm for discovering (from a portfolio of reusable assets) and deriving service variants for stated business process specifications, which we have called Variation-Oriented Service Design (VOSD). Via IBM's RSA Modeling Tool, we have also demonstrated our algorithm on a realistic running example in the insurance domain. Our prototype implementation proves the practical usefulness of our algorithm; hence our primary future work is to implement our algorithm along with our industry partners on real-life customer engagements.

## References

- [1] Arsanjani, A., Zedan, H., Alpigini, J.: Externalizing Component Manners to Achieve Greater Maintainability through a Highly Reconfigurable Architectural Style. In: Proceedings of International Conference on Software Maintenance (ICSM) 2002. IEEE Computer Society, Los Alamitos (2002)
- [2] Schneiders, Puhlmann, F.: Variability Mechanisms in E-Business Process Families. In: Proceedings of Business Information Systems, BIS 2006 (2006)
- [3] Narendra, N.C., Ponnalagu, K., Srivastava, B., Banavar, G.S.: Variation-Oriented Engineering (VOE): Enhancing Reusability of SOA-based Solutions. In: Proceedings of SCC 2008. IEEE Computer Society, Los Alamitos (to appear, 2008)
- [4] Arsanjani, A.: Empowering the Business Analyst for On Demand Computing. IBM Systems Journal 44(1) (2005)
- [5] Singh, M.P., Huhns, M.N.: Service Oriented Computing, 1st edn. Wiley-VCH Publishers, Chichester (2004)

- [6] Kumaran, S.: Model-driven Enterprise. In: Proceedings of the Global EAI (Enterprise Application Integration) Summit, pp. 166–180 (2004)
- [7] Ponnalagu, K.: Deriving service variants from business process specifications. In: Proceedings of ACM Compute (2008), [http://portal.acm.org/ft\\_gateway.cfm?id=1341776&type=pdf&coll=&dl=GUIDE&CFID=25839879&CFTOKEN=86101169](http://portal.acm.org/ft_gateway.cfm?id=1341776&type=pdf&coll=&dl=GUIDE&CFID=25839879&CFTOKEN=86101169)
- [8] Hubbers, J.-W., Ligthart, A., Terlouw, L.: Ten Ways to Identify Services. SOA Magazine (accessed May 21, 2008), <http://www.soamag.com/I13/1207-1.asp>

# Market Overview of Enterprise Mashup Tools

Volker Hoyer<sup>1,2</sup> and Marco Fischer<sup>1</sup>

<sup>1</sup> SAP Research CEC St. Gallen, 9000 St. Gallen, Switzerland

<sup>2</sup> University of St. Gallen, 9000 St. Gallen, Switzerland  
{volker.hoyer,marco.fischer}@sap.com

**Abstract.** A new paradigm, known as Enterprise Mashups, has been gain momentum during the last years. By empowering actual business end-users to create and adapt individual enterprise applications, Enterprise Mashups implicate a shift concerning a collaborative software development and consumption process. Upcoming Mashup tools prove the growing relevance of this paradigm in the industry, both in the consumer and enterprise-oriented market. However, a market overview of the different tools is still missing. In this paper, we introduce a classification of Mashup tools and evaluate selected tools of the different clusters according to the perspectives general information, functionality and usability. Finally, we classify more than 30 tools in the designed classification model and present the observed market trends in context of Enterprise Mashups.

**Keywords:** Enterprise Mashups, Internet of Services (IoS), Market Overview, Classification.

## 1 Introduction and Motivation

The networked information economy in the 21th century is characterized by a common-based peer production which represents a new model of economic production. According to Yochai Benkler who coined the term, "it refers to production systems that depend on individual action that is self-explained and decentralized rather than hierarchically assigned" [1]. Thereby, the creative energy of large number of people ("Wisdom of Crowds") is used to react flexible on continuous dynamic changes of the business environment. In the software development and consumption process, you can observe early signs of this common-based peer production as well. Known as Enterprise Mashups, the actual business end-users are empowered to adapt their individual business workplaces according to their individual and heterogeneous needs [2]. Driven by the consumer-oriented market, the Mashups paradigm is picked up in the enterprise context during the last two years.

Market research companies like Gartner [3], Forester [4] or the Economic Intelligence Unit [5] forecast a growing relevance of this paradigm in the next years. However, an overview of the upcoming Enterprise Mashup market regarding the provided functionality is missing. This article is devoted to a systematic analysis of the Mashup market to provide an overview of the current status.

The reminder of this paper is structured as follows: Chapter two deals with a clear definition of the Enterprise Mashup paradigm. In chapter three, we present a model to classify Mashup tools according to the dimension functionality (editor and catalogue) and target tools group (consumer and enterprise). By means of eight case studies, chapter four analyses existing tools for the identified clusters. In addition, the developed classification model is applied. Finally, chapter five closes this article with a brief summary and an outlook of identified trends in the Enterprise Mashup market.

## 2 Terms and Research Approach

### 2.1 Enterprise Mashups - Definition and Characteristics

In literature, the exact definition of Enterprise Mashups or in general Mashups is open to debate. To build our research on a scientific foundation, we have investigated existing definitions by an in-depth literature analysis. Besides references from scientific articles published in international journals or conference proceedings, we also consider the current situation in the industry which has driven the paradigm. Regarding from a technical ([6], [7], [8], [9], [10]), business ([11], [12], [13]), application ([14], [15]), consulting ([16], [17]), software vendor ([18], [19], [20]) and community (Wikipedia [21]) perspective, we analyse 16 definitions.

First of all, there exists no general difference between definitions used in the scientific community and in the industrial world which has driven the terminology so far. The authors of the scientific papers define the term Mashup by their own and don't quote a primary reference. In almost all definitions, the resource composition style is named as the central characteristic of Mashups. They only differ from which resources are mashed. Besides content or data ([9], [11], [12], [13], [15], [19], [20], [21]) according to the Web 2.0 data-centric approach, traditional application functionalities are also mentioned as Web-based resources ([18], [14], [30]). The history of the term itself is referred by many authors to the music hip hop style. Starting in the early 1970s, hip hop artists began mixing and matching beats from various sources, and then layering their own rhythmic vocals on top. This new art form proved highly popular with young people, and now constitutes one of the industry's most lucrative genres.

After a wide dissemination of the first consumer-oriented Mashups, the paradigm is also picked up in the context of enterprise applications. Even if software vendors [19] and some scientific authors introduced the term Enterprise Mashups [7], there exists no general difference to the original Mashup definition. Only the relation to existing concepts, in particular the Service-Oriented Architecture and the Web 2.0 philosophy are mentioned additionally. At the interface of this two converging principles, Enterprise Mashups put a visual face to the heavyweight Service-Oriented Architecture [18], [6], [7], [8] and can be interpreted as an evolution of SOA [18]. Therewith, they represent one technology to build "situational applications" [18] within hours to react flexible on changing business environments and follow an end-user centric approach [6].

Summing up and based on the literature review, we take the following summary as foundation for this work: "An Enterprise Mashups is a Web-based resource that combines existing resources, be it content, data or application functionality, from more than one resource in enterprise environments by empowering the actual end-users to create and adapt individual information centric and situational applications."

## 2.2 Lightweight Resource Composition

In context of Enterprise Mashups a clear and wide established terminology is missing both in the industry as well as in the academic world. In the following section we present the significant components and terms used in the discussion of Enterprise Mashups. Figure 1 depicts the two different lightweight composition styles (wiring and piping) and classifies them in an Enterprise Mashup Stack [2] that is based on three layers: Resources, Widgets and Mashups.

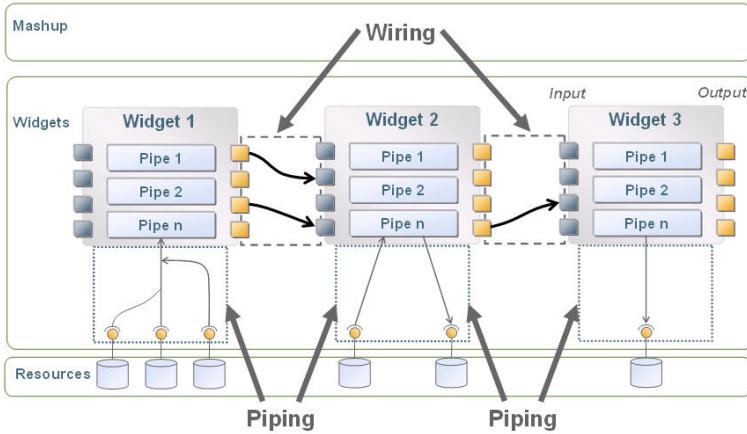


Fig. 1. Lightweight Composition with Enterprise Mashups (Piping vs. Wiring) [2]

**Resources.** The lowest layer contains the actual Web resources, be it content, data or application functionality. They represent the core building blocks of Enterprise Mashups and are the differentiator of the resource-centric paradigm. According to the lightweight Representational State Transfer (REST) architecture style [22], each Web-based resource can be addressed by a Universal Resource Identifier (URI). The resources itself are sourced via a well-defined public interface, the so called Application Programming Interfaces (APIs). They encapsulate the actual implementation from the specification and allow the loosely coupling of existing Web-based resources.

**Widgets.** Widgets represent application domain functions or information specific functions and put a visual face on the underlying resources. They are sourced

via public APIs, and are responsible for providing graphical, simple and efficient user interaction mechanisms that abstract from the technical description (functional and non-functional) of the Web-based resources. The term used is widget, but some vendors are also using the wording gadget or badget.

**Mashup.** By assembling and composing a collection of widgets stored in a catalogue or repository, end-users are able to define the behavior of the actual application according to their individual needs. By aggregation and linking content of different resources in a visual and intuitive way, the end-users are empowered to create their own workspace which fits best to solve their heterogeneous business problems. No skills in programming concepts are required.

The central driver of Enterprise Mashups is the lightweight resource composition style by reusing building blocks in different contexts [2]. As depicted in Figure 1, the composition takes place both on the resource layer (piping) and on the widget (wiring) layer. In reference to the UNIX shell pipeline concept, the *piping* composition integrates a number of heterogeneous Web-based resources defining composed processing data chains/ graphs concatenating successive resources. The output of each process is direct input to the next one. Aggregation, transformation, filter and sort functions adapt, mix and manipulate the content, data and application functionality of the Web-based resources. On the widget layer, the actual end-user is able to *wire* existing widgets together by interconnecting visually their input and output parameters.

### 2.3 Research Approach

This section is devoted to the presentation of the research approach applied. The goal of the research is to provide an overview about Enterprise Mashup tools. Based on an intensive analysis of more than 30 Mashup tools, we design a Mashup classification model to classify the heterogeneous tools. Thereby the analysis is done by means of three main criteria. First, a general information part consists of vendor information and about the tool itself (e.g., Mashup type and addressed target group). Second, the Mashup tools are analysed concerning their functionality (catalogue, mass collaboration and lightweight composition style [2]). Third, an analysis of the usability gives an answer how user-friendly a Mashup tool is. The developed classification model is applied by classify different Mashup tools. In addition, case studies for each cluster of the model describe the functionality of existing tools. Summering up, identified and observed market trends are presented.

## 3 Classification Model

After presenting the underlying research approach, this section presents a designed Mashup classification model. By means of an intensive analysis of different tools we identify two main differentiators: functionality/ property and target group.



### 3.1 Functionality/ Property

Due to the innovative characteristic of Mashup tools, the provided functionalities cover a wide area reaching from catalogue to more visually editor functions.

**Catalogue.** A catalogue is a collection or library of existing resources and widgets dependent on which layer (resource or widget) you operate. By describing the resources by the resource provider or also by the actual end-user following the Web 2.0 philosophy, the user is enabled to search for relevant resources according to his individual and heterogeneous need. In addition to these organizational tasks, catalogues have also to mediate between the different resources types. As mentioned before, the power of the Enterprise Mashup paradigm is based on the integration and easily connection of different resource types might it be Web pages (HTML), Web Services (WSDL), databases or enterprise applications. In general, a catalogue is sub-divided into repository and adapter:

- An **adapter** integrates existing resources in Mashup environments by mediate between different resource types both on syntactic and semantic level. To connect the resources, the adapter maps to a common protocol and internal format. For example, adapters scrape information from unstructured Web pages expressed in HTML or integrate complex Web Services with several input and output parameters into the Mashup environment by reducing the parameters.
- A **repository** organizes the growing number of resources and widgets in the Internet of Services. First of all, it comprises a registry in order to make them all retrievable via one single point of access. A user is so enabled to browse the existing widgets and resources. Also the efficient management and governance (e.g., service level agreement to ensure quality-of-service interaction) is part of the structural organization of a Mashup repository. The ad-hoc lightweight composition requires also a user-driven organization following the Web 2.0 philosophy. In that sense, end-users are enabled to rate, describe or recommend services.

**Editor.** An editor allows creating, modifying and aggregating of individual software applications by connecting resources retrievable from the catalogue. To address end-users characterized with no or only limited programming skills, the editor abstracts from the underlying programming interfaces through intuitive visual design environments following a "programming with the mouse" principle. The creating of ad-hoc enterprise applications is done within minutes or even hours instead of days or months. In contrast to traditional development environments focusing on actual programmers, Mashup editors don't separate between the design and runtime phase. These two phases are converging. According to the layer concept of the Enterprise Mashup Stack presented in chapter two, Mashup editors focus either on a presentation layer or on a transformation/ aggregation layer:

- **Transformation/ Aggregation.** combine data and content according to the lightweight resource composition style (piping) by reusing building blocks in different contexts. Information is sourced from several resources comparable with traditional EAI tools.

- **Presentation layer** tools present content from disparate sources together in a unified view and run the composition. In that sense, the presentation tools focus by default explicitly on end-users and use the transformed and aggregated resources.

### 3.2 Target Group

In addition to the actual functionality of Mashup tools, we have to distinguish from the addressed target group. Driven by the consumer oriented market in the last years tools focus more and more on enterprise requirements which are different to the consumer requirements.

**Consumer Mashups.** A consumer Mashup tool is mainly aimed at individuals to easily create Mashups for private use, e.g. personalized browser page. The consumer Mashup is perhaps the best know type of Mashups. Consumer Mashups combine data elements from multiple sources, hiding this behind a simple unified graphical interface. Instead of opening several Web pages to view, for example, the weather forecast, the news and your private emails, the consumer is able to create an individual start page pulling the information from different sources.

**Enterprise Mashups.** They combine existing resources, be it content, data or application functionality, from more than one source in enterprise environments. In contrast to consumer Mashups, enterprise environments implicate additional requirements like security, quality or availability. In additional, Enterprise Mashups focus on integrating existing back-end systems. So, Enterprise Mashups have enormous potential to allow more rapid and much less expensive development of applications by emphasizing assembly over development, economies of scale by enabling high levels of reuse, and the consequent ability to rapidly get software solutions with the right data in the right place at the right time.

## 4 Market Overview Mashup Tools

In chapter four, we provide a market overview of existing Mashup tools by means of the presented classification model. Several vendors provide more than one feature; in this case we classify tools according to one capability in the case studies.

### 4.1 Case Studies: Consumer Market

**Adapter.** Dapper entered the Mashup market in 2005. Dapper's core business is to provide public Web-based Software as a Service (SaaS) tool for generating feeds from any Web page (e.g., RSS, XML, JSON, etc.). Dapper Factory<sup>1</sup> lets user extract and structure data from around the Web, and then create services based around this structured information. Dapper is aimed at consumer and expand their ability to integrate Web content into their personal live. Additionally,

---

<sup>1</sup> <http://www.dapper.net/>



Fig. 2. Netvibes Ecosystem

Dapper provides a Web community, where user can access and share Dapper delivered feeds. Therefore is Dapper also a repository, but the main functionality is the Web-based wizard. For now, Dapper is a free and entirely Web-based service though they intend on providing revenue generators in the near future with an enterprise license.

**Repository.** Netvibes<sup>2</sup> is one of the pioneers of personalized browser pages (presentation layer), and includes a huge repository of predefined resources. Netvibes let you assemble widgets, feeds, social networks, email, videos and blogs on one fully-customizable page. Netvibes introduced a beta version of the same name in 2006 and is much like iGoogle, and Microsoft Live, but has much less advertising and much more focus on the widgets, which are stored in the repository. Furthermore Netvibes Ecosystem<sup>3</sup> is a collection of user submitted modules or widgets. Widgets can be tagged, rated or commented and are findable through categories (e.g. business, news, education and so on) or browser search.

**Transformation/ Aggregation.** Microsoft develops, manufactures, licenses, and supports a wide range of software products for computing device. Microsoft decided to launch their first Mashup product in 2005, and recognized early, that there could be a potential in the market. Microsoft provides with Popfly<sup>4</sup> a public 3D Web-based consumer tool with a Mashup Creator to combine and aggregate widgets with the lightweight resource composition style (piping). Popfly has a rich user interface based on the Silverlight technology. Behind is a social network called Popfly Space for sharing, rating, and commenting user contributed Mashups. For this reason, Popfly is an editor as well as a repository of existing resources.

**Presentation.** Google Inc. offers with iGoogle<sup>5</sup> a presentation tool for the Mashup market. Google's consumer platform aims to centralize all personal information in a personalized browser page. iGoogle includes the capability to add RSS feeds and Google gadget, similar to those available on Google desktop. Some of the themes are animated depending on weather conditions or the time

<sup>2</sup> <http://www.netvibes.com/>

<sup>3</sup> <http://eco.netvibes.com/>

<sup>4</sup> <http://www.popfly.com/>

<sup>5</sup> <http://www.google.com/ig>

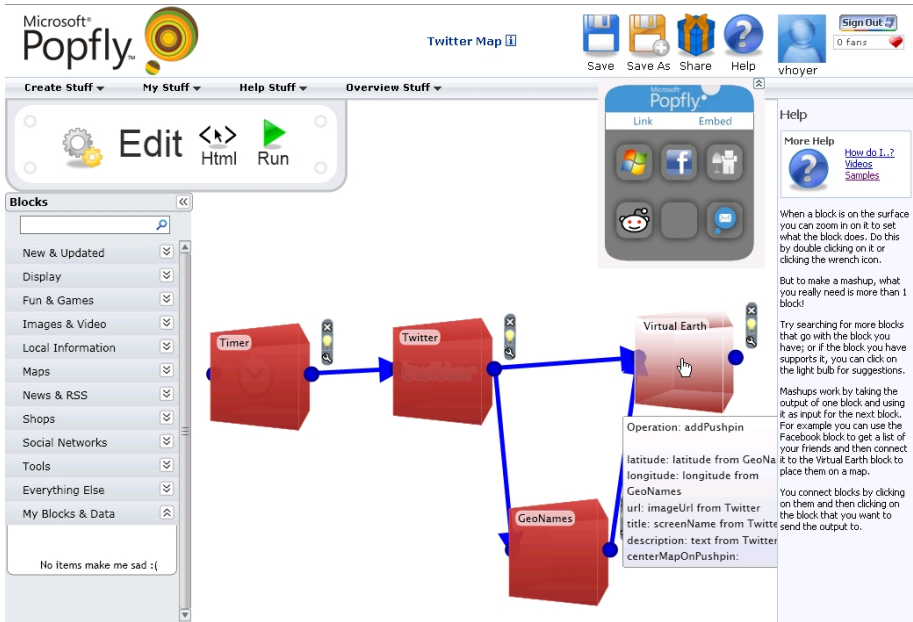


Fig. 3. Microsoft Popfly

in different areas. Furthermore the Google Gadgets API is public and allows anyone to develop a iGoogle gadget. Since May 2007, Google integrates a Gadget Maker, where users can create a special gadget that does not require the use of the Gadgets API. Thus iGoogle is a combination between a repository and front-end tool, but the main capability is to present content from different sources.

## 4.2 Case Studies: Enterprise Market

**Adapter.** Kapow Technologies is a standing vendor in the Mashup market, which already launched its product in 2001. Therefore, Kapow has been in production at many large consumer sites and enterprises for years. Kapow provides both an open community known as OpenKapow<sup>6</sup> as well as a desktop-based Mashup adapter called Kapow Mashup Server<sup>7</sup>. This commercial adapter focuses on information access, augmentation and scraping off Web-based information. Kapow uses robots to convert unstructured data form various sources into information feeds. Kapow's key feature is to turn Web pages into data sources with a wide range of data outputs (e.g., XML, HTML, CSV, and JSON).

**Repository.** IBM is a multinational computer technology and consulting corporation which manufactures and sells computer hardware and software, and offers

<sup>6</sup> <http://openkapow.com/Default.aspx>

<sup>7</sup> <http://www.kapowtech.com/products.html>



Fig. 4. IBM Mashup Hub

infrastructure services, hosting services in areas ranging from mainframe computers to nanotechnology. IBM launched with the Mashup Starter Kit its first Mashup Product in 2006. IBM Mashup Starter Kit consists of two technologies, and one of them is IBM’s Mashup Hub<sup>8</sup>. Mashup Hub is primary a catalogue of feeds and widgets, which can be input for another IBM product QEDWiki. The repository allows to tag and rate resources and the aid of guided process flows for ease of use. IBM’s Mashup Hub is also a Web-based editor to create feeds from different sources, (e.g., XML, SQL queries, spreadsheet) and supports therefore feed generation for enterprise data sources, which finally stored in the repository.

**Transformation/ Aggregation.** Yahoo provides a wide array of Internet services that cater to most online activities. They entered 2006 with its Web application product Yahoo! Pipes<sup>9</sup> in the Mashup market. Yahoo! Pipes provides a graphical user interface for building applications that aggregate Web feeds as depicted in Figure 5, Web pages, and other services, creating Web-based applications from various sources, and publishing those applications. Development is based on dragging resources from a toolbox and dropping them in work area, specifying data input, interconnecting gadgets through "pipes" and specifying data output format. At the moment Pipes has limitations for enterprise use, but it can handle simple business needs by now.

**Presentation.** The company JackBe was founded in 2002 as an AJAX widget company to later find itself targeting the IT enterprise market. JackBe launched

<sup>8</sup> <http://services.alphaworks.ibm.com/mashuphub/>

<sup>9</sup> <http://pipes.yahoo.com/pipes/>

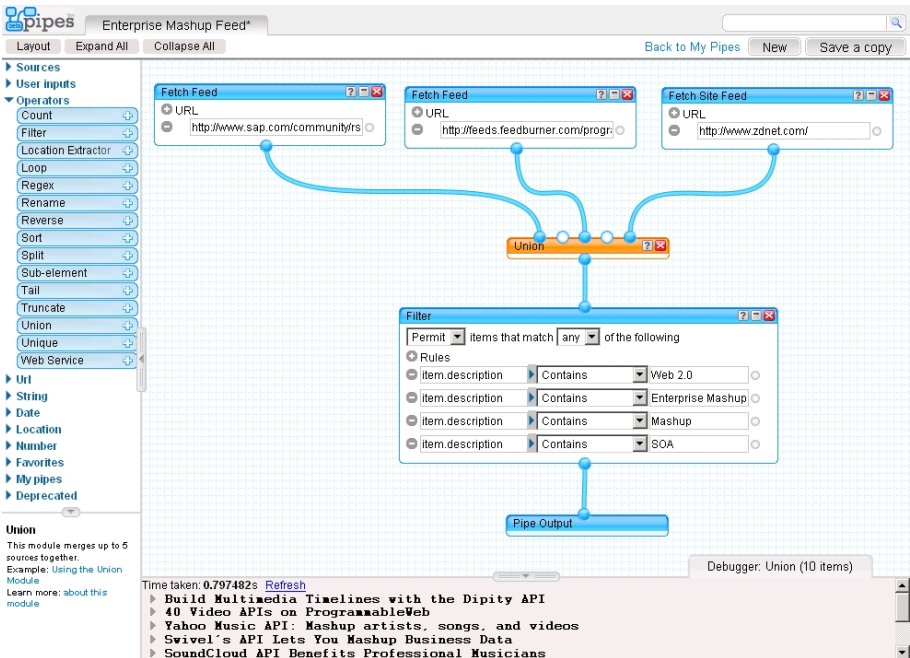


Fig. 5. Yahoo Pipes

with Presto a series of Enterprise Mashup solutions, consist of four components. In this case we cater to JackBe's front-end tool Presto Edge Enterprise Mashup Server<sup>[10]</sup>. Presto Edge allows publishing of Web services and provides collaboration and execution on the presentation layer. Sources, e.g. from another JackBe tool can easily piece together to a new application. JackBe also offers predefined solutions in security, administration and management capabilities.

### 4.3 Market Overview

After presenting selected Mashups tools for the different clusters according to the designed classification model, we classify more than 30 Mashup tools, shown in Figure 6. Several vendors provide more than one capability according to the classification model and therefore these vendors are classified in multiple clusters. For example, IBM QEDWiki is mentioned as an editor and a catalogue.

The market overview identifies a huge amount of vendors, which cavort in the Mashup space. Tools and services like Pageflakes, iGoogle aimed at consumer and non-technical users to create and publish their own Mashups. Consumer can easily generate own personalized browser pages with news, feeds and different gadgets. But Mashups seem to be not only a bauble for consumer, more and more enterprise vendors enter the market from the application integration, information

<sup>10</sup> <http://www.jackbe.com/products/edge.php>



Fig. 6. Classification of Mashup Tools

integration, rich Internet application and enterprise portal markets. In the last two and a half years enterprises like Microsoft, IBM and Serena placed new products into the Mashup market. But this is just the beginning, because the market is still in a state flux.

Additionally the trend of extremely individualized worker [23] means that enterprises also must observe public Mashup technologies to measure their potential for enterprise use. Interest in Mashup tools is also affecting the horizontal portal market and multiple portal vendors already support Mashups to a limited degree [24]. Furthermore CRM and ERP software enterprises, which are not mentioned in the figure above, integrate Mashup solutions in their existing enterprise software (e.g., vtiger, Salesforce.com, or Oracle). Thus, non Mashup specialists come into the market and Mashup tools getting closer to mainstream business use - they are moving into the enterprise as mentioned before. Companies have significant challenges to integrate information from various resources and Mashups can be an answer of this problem. They recognize the theoretical potential behind Mashup tools and plan to use them in the next years anymore [5]. In addition to positives of the market, enterprises should be aware, that Enterprise Mashup tools still in its infancy and aspects like security, administration, new and necessary IT strategies are not in detail discussed or implemented by now.

## 5 Conclusion

The opportunity of remixing internal and external resources more rapid and with much less expensive development into entirely new applications has captured the software industry. Niche players, visioniers and challengers provide a wide range of different Mashup tools and platforms to grip market share. This market overview analyses several tools and offers an overview of the existing market with the assistance of a Mashup classification model. The analysis shows that Enterprise Mashups still in flux and the market will be in move over the next years. New enterprises will enter the market, because Mashup tools affect the whole software industry. A new report from Forrester Research predicts that Mashups will be coming to the enterprise in a big way with a USD 700 Mio market by 2013 [4]. Additionally the way workers view their workplace is changing. New employees have different skills and expectations, because they are grow up with IT and know how to customize and individualize almost everything [25].

Nevertheless, there are still issues and lacks in existing tools, like a missing screenflow design, semantic aspects and the covering of enterprise typical requirements like security or reliability. In frame of the EU funded project FAST [26]<sup>11</sup>, we are currently developing a new visual environment following a user-centric approach that will facilitate the development of complex widgets required in business environments. Besides these technical challenges, the project focuses on the creation of business relevant widgets. As identified in the evaluation of the different Mashup tools, the actual content encapsulated by widgets is still missing. In future, a growing number of business widgets will be a critical success factor for a wide dissemination of the Mashup paradigm in corporate environments.

**Acknowledgments.** This paper has been created closely to research activities during the EU-funded project FAST (INFSO-ICT-216048) [26].

## References

1. Benkler, Y.: *The Wealth of Networks. How Social Production Transforms Markets and Freedom.* Yale University Press, New Haven (2006)
2. Hoyer, V., Stanoevska-Slabeva, K., Janner, T., Schroth, C.: Enterprise Mashups: Design Principles towards the Long Tail for User Needs. In: *IEEE International Conference on Services Computing (SCC)*, vol. 2, pp. 601–602 (2008)
3. Bradley, A., Gootzit, D.: *Who's Who in Enterprise Mashup Technologies.* Gartner Research (2007)
4. Young, G., Daley, E., Gualtieri, M., Lo, H., Ashour, M.: *The Mashup Opportunity.* Forrester (2008)
5. *The Economist Intelligence Unit: Serious Business - Web 2.0 goes Corporate.* The Economist Intelligence Unit (2007)
6. Liu, X., Hui, Y., Sun, W., Liang, H.: Towards service composition based on mashup. In: *Proceedings of the IEEE International Conference on Service Computing (SCC 2007)*, pp. 332–339 (2007)

<sup>11</sup> <http://fast.morfeo-project.eu/>



7. Janner, T., Canas, V., Hierro, J., Licano, D., Reyers, M., Schroth, C., Soriano, J., Hoyer, V.: Enterprise Mashups: Putting a face on next generation global SOA. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, Springer, Heidelberg (2007)
8. Soriano, J., Lizcano, D., Canas, M., Reyes, M., Hierro, J.: Foster Innovation in a Mashup-oriented Enterprise 2.0 Collaboration Environment. *System and Information Sciences Notes* 1(1), 62–68 (2007)
9. Kultathuramaiyer, N.: Mashups: Emerging application development paradigm for a digital journal. *Journal of Universal Computer Science* 13(4), 531–542 (2007)
10. Daniel, F., Matera, M., Yu, J., Benatallah, B., Saint-Paul, R., Casati, F.: Understanding UI Integration. A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing* 11(3), 59–66 (2007)
11. Dearstyne, B.: Blogs, mashups, wikis oh my. *Information Management Journal* 41(4), 24–33 (2007)
12. O'Brien, D., Fitzgerald, B.: Mashups, remixes and copyright law. *Internet Law Bulletin* 9(2), 17–19 (2007)
13. Gerber, R.: Mixing it up on the web: Legal issues arising from the internet mashup. *Intellectual Property and Technology Law Journal* 18(8), 11–14 (2007)
14. Miller, C.: A beast in the field: The google maps mashup at *gis/2*. *Cartographica - The International Journal for Geographic Information and Geovisualization* 41(3), 187–199 (2007)
15. Cho, A.: An introduction of mashups for health librarians. *Journal of the Canadian Health Libraries Association* 28(1), 19–22 (2007)
16. The Economist: Mashing the web. *The Economist - Special Section* 376(8444), 4 (2005)
17. Hof, R.: Mix, match, and mutate. *Business Week Magazine* (2005)
18. Watt, S.: Mashups - the evolution of the soa, part 2: Situational applications and the mashup ecosystem (2007), <http://www.ibm.com/developerworks/webservices/library/ws-soa-mashups2/>
19. Clarkin, L., Holmes, J.: Enterprise mashups. *The Architecture Journal* 13 (2007)
20. Salesforce: Mashups: The what and why (2007), <http://wiki.apexdevnet.com/index.php/>
21. Wikipedia: Mashups (2008), <http://en.wikipedia.org/>
22. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. Thesis (2000)
23. Morello, D., Burton, B.: Future Worker 2015. Extreme Individualization. In: *Garnter Symposium ITxpo, Orlando* (2005)
24. Gootzit, D., Phifer, G., Valdes, R.: Magic Quadrant for horizontal Portal Products. *Garnter Research* (2007)
25. Cherbakov, L., Bravery, A., Goodman, B., Pandya, A., Bagget, J.: Changing the corporate it development model: Tapping the power of grassrots computing. *IBM System Journals* 46(4) (2007)
26. FAST: EU Project FAST (INFSO-ICT-216048) (2008), <http://fast.morfeo-project.eu/>

## Appendix: Evaluation Matrix

Evaluation Criteria	Dapper Factory	iGoogle	IBM Mashup Hub	JackBe Presto Edge	Kapow Mashup Server	Microsoft Popfly	Netvibes	Yahoo Pipes
<b>Product Information</b>								
<i>Mashup Type</i>								
Editor		x		x		x	x	x
Catalogue	x	x	x		x	x	x	x
<i>Target Group</i>								
Consumer	x	x				x	x	x
Enterprise			x	x	x			x
<b>Functionality</b>								
<i>Mashup Catalogue</i>								
Number of resources <sup>a</sup>	2.000	n.a.	150	n.a.	6.000	150	172.000	1.000
Number of resource types <sup>b</sup>	7	4	6	5	7	3	3	4
Query and search capabilities	x	x	x	x	x	x	x	x
Creation of widgets	x	x		x	x	x	x	x
External catalogues			x					
<i>Mass Collaboration</i>								
Tagging	x	x	x	n.a.	x	x	x	x
Rating		x	x	n.a.	x		x	x
FAQ	x	x	x	x	x	x	x	x
Blog	x	x		x	x	x	x	x
Forum	x	x	x	x	x	x	x	x
<i>Lightweight Composition</i>								
Screenflow design								
Visual Wiring				x		x		
Visual Piping				x		x		x
<b>Usability</b>								
<i>End-User Interface</i>								
Ease of use	x	x	x			x	x	x
Drag-and-Drop		x		x		x		x
Guided process flows	x	x	x	x	x	x		x
Performance	<2 sec	<2 sec	<1 sec	<1 sec	<1 sec	<2 sec	<1 sec	<2 sec
<i>Help</i>								
Demos	x	x	x	x	x	x	x	x
Tutorials		x		x	x	x	x	x

<sup>a</sup> Estimated number

<sup>b</sup> Resource types can be: Widgets, ATOM, CSV, data base, HTML, JSON, RSS, WSDL, XML, etc.

# Siena: From PowerPoint to Web App in 5 Minutes

David Cohn<sup>1</sup>, Pankaj Dhoolia<sup>2</sup>, Fenno Heath III<sup>1</sup>, Florian Pinel<sup>1</sup>, and John Vergo<sup>1</sup>

<sup>1</sup> IBM T.J. Watson Research Center  
{dcohn, theath, pinel, jvergo}@us.ibm.com

<sup>2</sup> IBM India Research Lab  
pdhoolia@in.ibm.com

**Abstract.** Siena lets users design web applications using commonly available PowerPoint as the modeling/development tool. From PowerPoint, users can model business artifacts and processes, transform applications to a standard representation and then immediately deploy and execute these composite applications on a model execution engine.

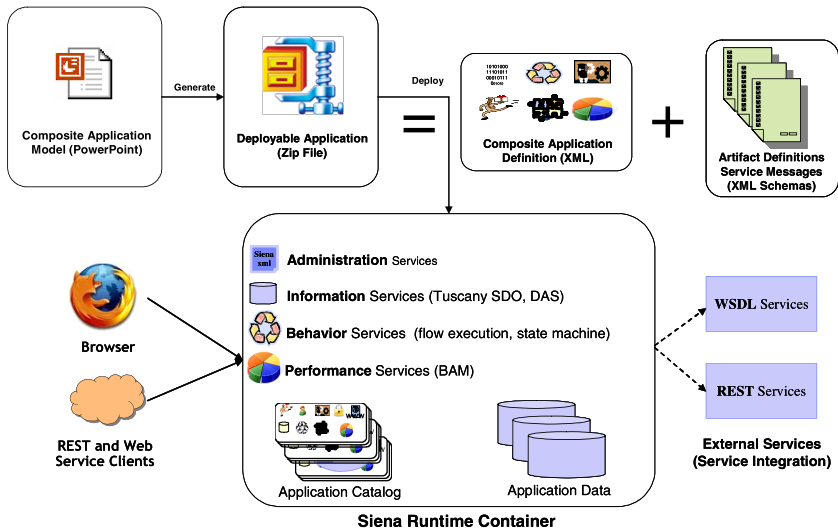


Fig. 1. Siena Architecture

## Characteristics and Architecture

By contrast with traditional Business Process Management (BPM) tools, Siena provides a flexible method for developing BPM applications that let innovators focus on business transformation and adapt to changes rapidly:

- A radically simplified modeling tool based on PowerPoint empowers any business user to model and manage the entire lifecycle of BPM applications.
- A standard, human-readable XML representation facilitates the persistence and sharing of BPM application definitions.

- The Siena Runtime Container, a web 2.0 model execution engine running on Apache Tomcat, allows for rapid solution development and deployment.

Fig. 1 describes the overall Siena architecture. Clients communicate with deployed BPM applications using a browser and REST or Web Services. As part of provisioning these services, the Siena runtime does the following:

- Manage applications (Administration Services).
- Persist, access and manage business artifacts (Information Services).
- Execute business flows and state machines (Behavior Services).
- Monitor business performance (Performance Services).
- Consume services from other applications (External Services).

Fig. 2 and Fig. 3 examine the end-to-end transformation of an artifact lifecycle. Users start by drawing a state diagram for their business artifact in PowerPoint. The deployed application automatically offers persistence and transition services.

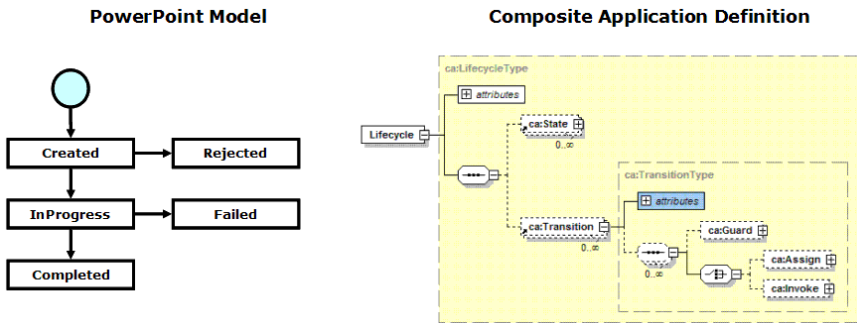


Fig. 2. Artifact Lifecycle in PowerPoint and in Composite Application XML Definition

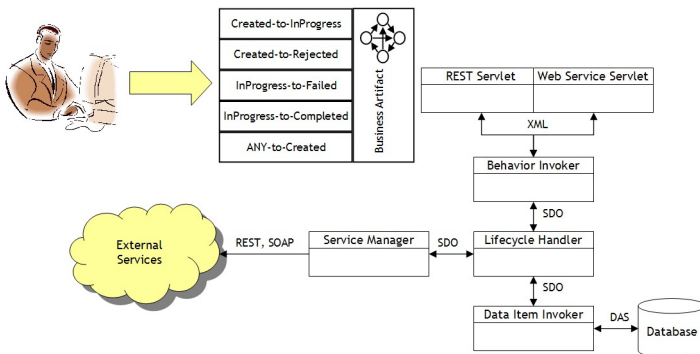


Fig. 3. Execution of Artifact Transition Services

The demonstration showcases the creation of business artifacts and component services, the invocation of external services, and the deployment and testing of the resulting composite applications.

# Exploration of Discovered Process Views in Process Spaceship

Hamid R. Motahari Nezhad<sup>1</sup>, Boualem Benatalah<sup>1</sup>, Fabio Casati<sup>2</sup>,  
Regis Saint-Paul<sup>3</sup>, Periklis Andriostos<sup>2</sup>, and Adnene Guabtui<sup>1</sup>

<sup>1</sup> Comp. Sci. and Eng., The University of New South Wales, Australia

<sup>2</sup> Dept of Inform. and Comm. Tech., The University of Trento, Italy

<sup>3</sup> CREATE-NET International Research Center, Trento Italy

## 1 Introduction

Business processes are important for streamlining the operations of public and private enterprises. Over the last decade, capabilities arising from advances in online technologies, especially Service Oriented Architectures (SOA), enabled enterprises to increase productivity, simplify automation, and extend the execution of business processes to various systems in the enterprise. While business process management systems, which allow for modeling, analysis, and management of business processes, are relatively successful, currently, they only cover a fraction of business processes in the enterprise. One challenge in modern enterprises is that information about business process execution is maintained over multiple heterogeneous systems (e.g., email systems, ERP, document management systems, etc), and rarely there exists a central workflow log, where all process execution information can be found. The next challenge is that the traditional one-view-fits-all fashion of process definition does not scale, as different users may have their own perspective of the business process execution in the enterprise. In such environments, not only one but a space of processes can be defined corresponding to the perspectives of different users or systems involved in the process.

We define *process views* as an abstract representation of a process, from the perspective of a user or a system, in terms of tasks and their relationships, i.e., control and data flow, and properties such as participating roles, and execution times. Process views can be defined at various levels of abstractions (high level or detailed). Furthermore, we define a *process space* as the superimposition of a set of process views in the enterprise, at various levels of abstractions, over heterogeneous information systems containing process execution information.

We have developed *Process Spaceship* for the discovery of process views from process related data sources [2]. Unlike process discovery from workflow logs [3], where the *process instances* [1] are known and the problem is that of discovering the process definitions, in process spaces, the new challenge is that of *events correlation*, i.e., identifying which set of events in application logs are related to the same process instance. Events correlation in SOA can be done following various patterns, e.g., based on the content (data attributes) of events

---

<sup>1</sup> A process instances specifies a set of messages that are exchanged to fulfill a certain goal.

or their timestamp [1]. In principle, there are more than one way of correlating events into process instances, corresponding to various process views that can be defined in the enterprise. We define *correlation condition* as binary predicate over the attributes of event content (e.g., *orderID=orderID*) to specify if two events correspond to the same process instance. Correlation conditions could be atomic (defined over a pair of attributes) or composite (consists of several atomic conditions). We have proposed a set of algorithms and heuristics to discover interesting correlation conditions following a level-wise approach starting from atomic conditions followed by the discovery of composite conditions [2].

## 2 Demonstration Scenario

We demonstrate two features of *Process Spaceship*: (i) explorative process views discovery, and (ii) process space navigation. For the sake of working in the context of a concrete example, we use the logs of interactions of a set of Web services in a supply chain scenario as the input. The system has been developed in Java, and uses HP SOA Manager to monitor service interactions.

*Explorative discovery of process views.* The explorative discovery of process views in *Process Spaceship* starts with capturing information that the user might have about how events are correlated in the input data sources. This includes selection of attributes that are potentially used in correlation, and if known, the correlation pattern. Based on this information, a set of basic process views are discovered and presented to the user, among which she may select the ones that are of her interest. Depending on the input data, and the correlation pattern(s) that is(are) used for correlation, the basic views may be composed to form composite process views. These views contain larger processes, compared to the basic ones. This procedure can continue until a process view corresponding to the business process of the enterprise is discovered. However, only user-selected views are kept and used as a basis for discovering larger, more abstract views.

*Process Space Navigation.* The discovered process views (in an explorative manner or automatic way) are organized in a *process map*, where each process is represented by a node and is linked to other process views, which have *part-of* or *subsumed* relationship with it. We demonstrate how this organization facilitates navigation of process views, and also the use of various perspectives that *Process Spaceship* provides for end users.

## References

1. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)
2. Motahari, H., et al.: *Process Spaceship: Discovering process views in process spaces*. Technical Report UNSW-CSE-TR-0721, The University of New South Wales (2007)
3. van der Aalst, W., et al.: Workflow mining: a survey of issues and approaches. *DKE Journal* 47(2), 237–267 (2003)

# ROME4EU: A Web Service-Based Process-Aware System for Smart Devices\*

Daniele Battista<sup>2</sup>, Massimiliano de Leoni<sup>1</sup>, Alessio De Gaetanis<sup>2</sup>,  
Massimo Mecella<sup>1</sup>, Alessandro Pezzullo<sup>2</sup>, Alessandro Russo<sup>2</sup>,  
and Costantino Saponaro<sup>2</sup>

<sup>1</sup> Dipartimento di Informatica e Sistemistica  
SAPIENZA - Università di Roma, Rome, Italy  
{deleoni,mecella}@dis.uniroma1.it

<sup>2</sup> Faculty of Computer Engineering  
SAPIENZA - Università di Roma, Rome, Italy

Nowadays, process-aware information systems (PAISs) are widely used for the management of “administrative” processes characterized by clear and well-defined structure. Besides such scenarios, PAISs can be used also in mobile and pervasive scenarios, such as in coordinating operators during emergency situations [1]. In these pervasive settings, due to highly mobility, operators have to be equipped with small devices, such as PDAs, and to communicate through ad-hoc networks.

At the best of our knowledge, all of available PAISs allow currently to execute on smart devices only client applications, such as work-list handlers for accepting/refusing the assigned tasks. The engines at the heart of PAISs, which are in charge of assigning tasks to process participants, are still designed to be executed on standard desktop machines. Therefore, the current PAISs cannot really work in pervasive and highly mobile scenarios where the entire system has to be deployed on the spot and running on smart devices. The possibility of having a remote coordination center where a PAIS engine is running is not really feasible. Assigning tasks to team members and orchestrating the process execution remotely would require to exist an underlying infrastructure reliable and fast. That is not the case in these settings where the best case would be having a GPRS/UMTS connection. Furthermore, the Hurricane Katrina experience has taught us that if every team used such an infrastructure, it would be going to fall down or would become too slow.

In the light of this, we have developed a PAIS, namely ROME4EU (The Roman Orchestration Mobile Engine for Emergency Units) [2], whose engine resides on the MS Windows Mobile PDA of the team leader. Modern PDAs are becoming increasingly powerful and, hence, able to execute complex applications. Team Members are also equipped with PDAs and their work is coordinated by the PAIS running on the team leader PDA. That makes ROM4EU really applicable in pervasive scenarios.

In ROME4EU, process schemas are defined in the form of Activity Diagrams enriched for describing all the different aspects: definition of tasks in term of

---

\* This work is supported by the European Commission through the FP6-2005-IST-5-034749 project WORKPAD.

<sup>1</sup> <http://www.dis.uniroma1.it/pub/mecella/projects/ROME4EU/>

pre- and post-conditions, the control and data flow, as well as the assignment of tasks to appropriate members. Tasks are associated to a set of conditions to hold in order that they are assignable to participants. Conditions are defined on control and data flow (e.g., a previous task has to be finished, a variable needs to get assigned a specific range of values, etc.). Every task can be only assigned to a certain member that provides certain capabilities. We model that by binding each and every task to a set of capabilities. Moreover, every member declares to furnish certain capabilities. Considering the control and data flow, the ROME4EU engine assigns every task to a certain member providing all capabilities required. At client side, every member uses a task handler to be notified of tasks assignment and to start the proper application for their execution. The same application is also used to perform the log-in phase, where members specify the capabilities they can provide and to show the information coming from the ROME4EU engine about the process to carry on.

ROME4EU overtakes interesting challenges, such as how to concretely design and develop a PAIS running on smart devices connected in mobile networks, considering that this network class provides reduced communication bandwidth and low reliability. Furthermore, smart devices are battery operating and, thus, the engine has to deal with the issue of minimizing the power consumption. It is worthy mentioning that reduced screen sizes limit the amount of information which can be visualized at the same time.

ROME4EU follows an approach comparable to BPEL4People [2] where task-list handlers are exposed as web-service endpoints and seen from the BPEL-based engine viewpoint as mere services to be integrated. Unfortunately, at our knowledge, no BPEL4People implementation is targeted to smart devices so far.

**Technical Solutions.** ROME4EU is completely developed on the .NET Compact Framework. The interaction between the engine and clients is based on web-service invocations. Specifically, we used [3] and extended it to handle complex data types, required for exchanging process variables, and one-way invocations. The latter feature is quite important in (unreliable) Mobile Settings, where it is difficult and battery consuming to keep alive SOAP connections for long times. The engine is based on a porting of the BPEL engine *Sliver* [4] in MS .NET C#, which has been later extended to integrate the aforementioned WS middleware.

## References

1. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Salvatore, B., Vetere, G., Dustdar, S., Juszczak, L., Manzoor, A., Truong, H.: Pervasive Software Environments for Supporting Disaster Responses. *IEEE Internet Computing* 12, 26–37 (2008)
2. Kloppmann, M., Koenig, D., Leymann, F., Pfau, G., Richayzen, A., von Rigen, C., Schmidt, P., Trickovic, I.: WS-BPEL Extension of People - BPEL4People (July 2005)
3. Nicoloudis, N., Prastitha, D.: .NET Compact Framework Mobile Web Server Architecture (2003) Prompted on June 8th, 2008, <http://msdn2.microsoft.com/en-us/library/aa446537.aspx>
4. Hackmann, G., Haitjema, M., Gill, C., Roman, G.C.: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In: Dan, A., Lamersdorf, W. (eds.) *ICSOC 2006*. LNCS, vol. 4294, pp. 503–508. Springer, Heidelberg (2006)



# WS-Engineer 2008

## A Service Architecture, Behaviour and Deployment Verification Platform

Howard Foster

Department of Computing, Imperial College London,  
180 Queen's Gate, London SW7 2BZ, UK  
[howard.foster@imperial.ac.uk](mailto:howard.foster@imperial.ac.uk)  
<http://www.ws-engineer.net>

**Abstract.** In this demonstration we present the LTSA WS-Engineer Tool Suite. WS-Engineer<sup>1</sup> started as a formal service composition analysis tool for service orchestrations based upon the Labelled Transition Analyser (LTSA). Since its introduction in 2006, the tool suite has grown to consider several areas of service composition engineering, including architecture, behaviour and deployment. The tool is integrated into the Eclipse and IBM Rational Software Architect IDEs.

## 1 The WS-Engineer Approach

Our initial tool support [1] and expanding approach in service composition analysis, takes a 2-dimensional view of service composition analysis. In a first dimension it considers core service composition artifacts being; orchestrations (service processes), service interfaces, choreography (global partner policies for interactions) and resources (architecture dependent features of service compositions). From a second dimension it considers the analysis features of service compositions including; design, implementation, architecture configuration and deployment. Thus, service engineers can use the approach to safety check designs for service orchestrations and choreography, or alternatively the deployment of collaborating processes and their architecture configuration in service choreography, or any aspect against the other in the matrix. We believe such an approach provides a much richer coverage of service composition development, accessible to engineers such that they can analyse compositions from different viewpoints (depending on the context of analysis). An overall integrated service behaviour analysis approach is suggested in Figure 1. The core of the approach is transforming some design or implementation artifact to relevant and detailed models (Labelled Transition Sstems) for analysis (Model Generation). We consider analysis of service orchestrations and choreography given input as design specifications (e.g. MSCs, UML2 and xADL2 models), implementations (in the form of WS-BPEL and WS-CDL policies) and service component interfaces (in the

---

<sup>1</sup> Sponsored by the EU funded project SENSORIA (IST-2005-016004) and by IBM Eclipse Innovation Awards (2006/07).

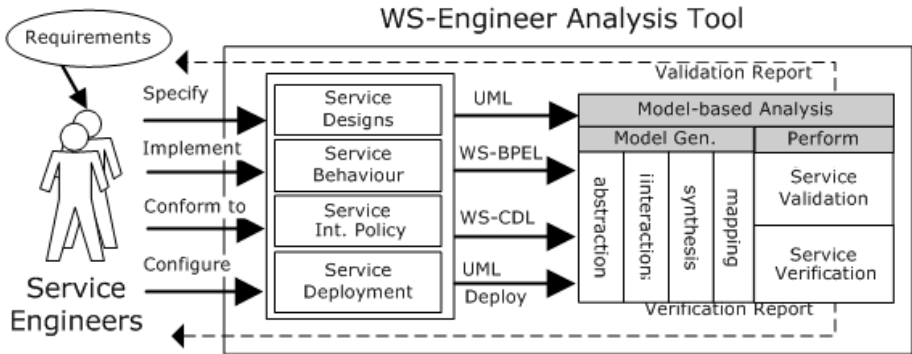


Fig. 1. WS-Engineer Approach to Service Composition Analysis

form of WSDL documents). Analysis in the approach provides features to compare each of these either as model validation (through animation) or verification through model property traces. Each feature considers behaviour analysis for a different element of service compositions which we aim to clearly demonstrate.

## 2 The Demonstration

The demonstration illustrates several integrated key features of WS-Engineer, themed in a way that follows conceptual design, implementation and deployment of service compositions. The example is focused on a single case study yet with multiple properties (engineering concerns) covered through different techniques using formal analysis. To begin with the user specifies a high-level design (in UML sequence charts) and implementations in WS-BPEL. Verification shows any violations with a weak bi-simulation between the models of processes in the service composition. Corrective actions are illustrated. The user then proceeds to specify a service choreography specification (in WS-CDL) and performs a similar verification against design and implementation. Towards deployment of these artifacts, the user builds a service deployment diagram for architecture configuration (highlighting service host resource constraints) of the compositions, specifies the WS-BPEL implementations and performs a check on behaviour and resource usage. Similarly a set of violations are raised and corrective actions illustrated. All three verifications steps aim to provide greater assurance prior to deployment and runtime of service compositions. Additionally we will highlight some future work on dynamic service compositions using our concepts of Service Modes and a prototype dynamic service broker.

## References

1. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Ws-engineer:tool support for model-based engineering of web service compositions and choreography. In: IEEE International Conference on Software Engineering (ICSE 2006), Shanghai, China. IEEE, Los Alamitos (2006)

# MetaCDN: Harnessing Storage Clouds for High Performance Content Delivery

James Broberg<sup>1</sup> and Zahir Tari<sup>2</sup>

<sup>1</sup> Department of Computer Science and Software Engineering, The University of Melbourne, Australia

<sup>2</sup> Department of Computer Science and Information Technology, RMIT University, Australia

**Abstract.** Content Delivery Networks (CDNs) such as Akamai and Mirror Image place web server clusters in numerous geographical locations to improve the responsiveness and locality of the content it hosts for end-users. However, their services are priced out of reach for all but the largest enterprise customers. An alternative approach to content delivery could be achieved by harnessing existing infrastructure provided by ‘storage cloud’ providers, at a fraction of the cost. MetaCDN is a system that leverages several existing ‘storage clouds’, creating an integrated overlay network that provides a low cost, high performance content delivery network for content creators. MetaCDN intelligently places content onto one or many storage providers based on the quality of service, coverage and budget preferences of participants.

## The MetaCDN System

Numerous ‘storage cloud’ providers (or ‘Storage as a Service’) exist that can provide coverage in several continents, offering Service Level Agreement backed performance and uptime promises for their services. Customers are charged based on their utilisation of storage and transfer of content, which is typically in the order of cents per gigabyte. Whilst these emerging services have reduced the cost of content storage and delivery by several orders of magnitude, they can be difficult to use for non-developers, as each service is best utilised via unique web services or programmer API’s. Furthermore, a customer may need coverage in more locations than offered by a single provider. To overcome this, MetaCDN utilises numerous storage providers in order to create an overlay network that can be used as a high performance, reliable and geographically distributed CDN.

The MetaCDN system integrates with each storage provider via a *connector* that provides an abstraction to hide the complexity arising from the differing provider interfaces. An abstract class, *DefaultConnector*, is defined that prescribes the basic functionality that each provider could be expected to support, that must be implemented for all existing and future connectors. These include basic operations like creation, deletion and renaming of files and folders. If an operation is not supported on a particular service, then the connector for that service should throw a *FeatureNotSupportedException*.

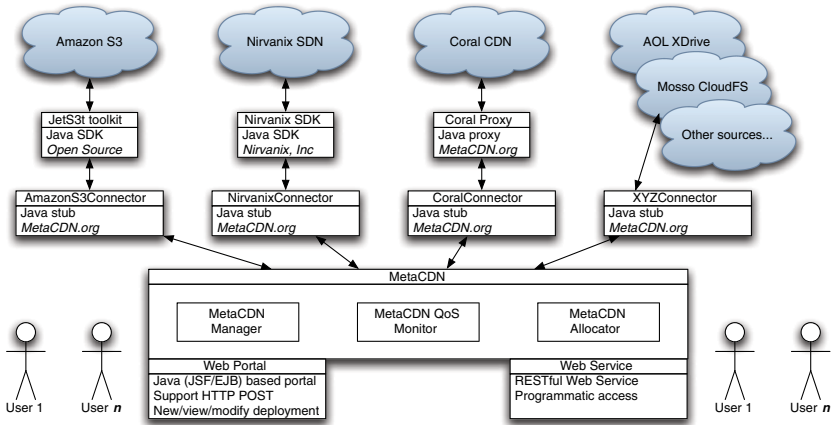


Fig. 1. MetaCDN

The service (depicted in Figure 1) is presented to end-users as a web portal for small or ad-hoc deployments (which is the focus of this paper) or as SOAP and RESTful Web Services (currently under development) for integration of customers with more complex and frequently changing content delivery needs. The web portal was developed using Java Enterprise and Java Server Faces (JSF) technologies, with a MySQL back-end to store user accounts, deployments, and the capabilities and pricing of service providers.

The MetaCDN system offers a number of functions via the web portal interface<sup>1</sup>, including: the creation of an account in the MetaCDN system, where a user registers their details, as well as credentials for any service providers they wish to utilise; manual deployment of content to geographical regions of the user's choice; deployment of file replicas to numerous geographically distributed locations based on a user's storage and transfer budget; viewing and modifying existing content deployment; and viewing the physical location of deployed content replicas as a Google Maps overlay.

A number of features are currently under active development, including: matching and deployment of file replicas to storage providers based on quality of service parameters like uptime, average throughput and average response time for end-users located in specific geographical areas; a single URL and namespace for uploaded files, with automatic client redirection to optimal replicas; deploying large files using Bittorrent as well as HTTP; and making all MetaCDN functionality available via SOAP and RESTful Web Services.

This work is supported by Australian Research Council (ARC) as part of the Discovery Grant 'Coordinated and Cooperative Load Sharing between Content Delivery Networks' (DP0881742, 2008-2010).

<sup>1</sup> A screencast of the web interface is available at <http://www.metacdn.org>

# Yowie: Information Extraction in a Service Enabled World

Marek Kowalkiewicz and Konrad Jünemann

SAP Research, 133 Mary Street, Brisbane, Australia  
{marek.kowalkiewicz,konrad.juenemann}@sap.com

**Abstract.** Service Oriented Computing is a potential enabler for popular applications of Named Entity Recognition and Information Extraction. In this demo we show an example of such an application and discuss how Service Oriented Architecture (SOA) makes the application fully flexible and easily extensible. The application brings SOA close to the end-user and gives possibilities hardly possible with other approaches.

**Keywords:** SOA applications, natural language processing.

## 1 Introduction

The domain of Natural Language Processing (NLP) has reached a state of maturity where commercial applications are possible and reliability of such applications becomes acceptable. Although some of the areas of NLP, for instance Information Extraction (IE) or Named Entity Recognition (NER), are particularly advanced [1,2], there still is no popular “killer application” that can demonstrate high potential of NLP. That holds for both enterprise and non-enterprise world.

There are some examples of NER applied in popular applications. However, these systems are not extensible by end-users in any way, and there is no possibility to (a) enable recognition of new types of entities or (b) provide users with other, alternative actions that can be performed with the recognized entities. As there are more and more information processing services available both in Internet and in enterprise software, SOA can provide a solution to this problem.

Ability to add new types of entity recognition on top of standard ones is especially important in enterprise context. Date, place name, or human name recognition, often provided by NLP systems, can be extended with features such as product name (code), customer name or contract number recognition, providing much more automation to the enterprise world. Ability to offer new services that process entities brings benefits to both enterprise and non-enterprise worlds.

Since relevance of actions and services, together with their rankings, varies based on numerous factors, high flexibility and ability to add and replace system components become useful. In this demonstration we showcase Yowie, a NER system built following SOA guidelines. Thanks to SOA, Yowie is fully extensible and can make use of external services.

## 2 Description of the System

Yowie provides a link between business productivity software and external services local desktop, enterprise systems and Internet vendors. The link that Yowie provides is based on the assumption that certain fragments of text documents contain enough information to create automatic links to data objects and services available in external systems.

Yowie consists of four main components: (1) a set of *plugins*, responsible for accessing local applications and communicating with Yowie, (2) a *core*, acting as a mediator between all other components, (3) a set of *extractors*, recognizing entities in documents, and (4) a set of *service wrappers* providing access to services and information related to extracted entities. The components are loosely coupled and each of them can be easily replaced. The main novel characteristics of Yowie include:

- Integration of NER and IE services.** Yowie can integrate various NER and IE services in one application following the SOA paradigm. Other existing approaches are either hard coded or do not integrate various services at all;
- NER and IE: entry points for service consumption.** NER and IE can be used for service selection. Yowie is an example of an application that can consume services offered by a service broker and choose the relevant ones;
- A potential for an SOA-enabled killer application.** By integrating NER, IE, and SOA in end-user applications, Yowie has a potential of becoming popular and being a visible example of SOA in end-user applications.

## 3 Functions and Features to be Demonstrated

During the demo we will showcase functionality of Yowie. During the presentation we will focus on one Yowie plugin, Yowie for MS Outlook. We are going to demonstrate the following.

- Ability to recognize various types of entities in processed documents using a simple email as an example.
- Ability to provide end users with related services (local, Web, and enterprise services). For each of the recognized entities we will show information and service access provided by Yowie.
- Ability to add or replace recognizers for new types of named entities. Yowie can be easily extended by adding new services that recognize named entities.
- Ability to add or replace offered services based on recognized entities. New services that use recognized entities can be easily added to the system.

## References

1. Maynard, D., Tablan, V., Ursu, C., Cunningham, H., Wilks, Y.: Named Entity Recognition from Diverse Text Types. Recent Advances in NLP (2001)
2. Nahm, U., Mooney, R.: Text Mining with IE. In: Proc. of the AAAI 2002 Spring Symposium on Mining Answers from Texts and Knowledge Bases (2002)

# Author Index

- Abramowicz, Witold 271  
Andristsos, Periklis 724  
Arbab, Farhad 70  
Ardagna, Danilo 599  
Arnold, William 162
- Balasubramaniam, Sriram 678  
Baligand, Fabien 483  
Bao, Liang 511  
Baresi, Luciano 614  
Barros, Alistair 331  
Bartolini, Cesare 524  
Basu, Samik 453  
Battista, Daniele 726  
Beauche, Sandrine 530  
Benatalah, Boualem 724  
Bertino, Elisa 116  
Bertolino, Antonia 524  
Binder, Walter 241  
Blau, Benjamin 517  
Broberg, James 730  
Budny, Peter 438
- Caporuscio, Mauro 195  
Carlson, Michael Pierre 317  
Carro, Manuel 302  
Casati, Fabio 724  
Chang, Henry 147  
Chang, Soo-Ho 180  
Chen, Ping 511  
Chen, Sheng 511  
Chen, Ying 649, 664  
Clayphan, Andrew John 225  
Cohn, David 722  
Comerio, Marco 607
- D'Andrea, Vincenzo 607  
Dasgupta, Gargi 54  
de Gaetanis, Alessio 726  
de Leoni, Massimiliano 726  
De Paoli, Flavio 607  
Decker, Gero 331  
Deridder, Dirk 592  
Dhoolia, Pankaj 722  
Di Francescomarino, Chiara 132
- Diament, Judah 422  
Duan, Zhenhua 505  
Dustdar, Schahram 607
- Eilam, Tamar 162  
Ezenwoye, Onyeka 54
- Faulkner, Stéphane 362  
Ferguson, Donald F. 4  
Ferrini, Rodolfo 116  
Fischer, Marco 708  
Fong, Liana 54  
Foster, Howard 558, 728  
Foster, Ian 3  
Franczyk, Bogdan 626
- Gangadharan, G.R. 607  
Ghidini, Chiara 132  
Gooneratne, Nalaka 585  
Gorton, Ian 225  
Govindharaj, Srihari 438  
Gowri Nanda, Mangala 378  
Guabtini, Adnene 724  
Guinea, Sam 614
- Han, Jun 5  
Haniewicz, Konstanty 271  
Harland, James 585  
He, Qiang 22  
Heath III, Fenno 722  
Herssens, Caroline 362  
Honavar, Vasant 453  
Hoyer, Volker 708  
Hu, Shengming 511  
Huhns, Michael N. 407
- Ishida, Toru 572  
Issarny, Valerie 195  
Iyengar, Arun 422
- Jalote, Pankaj 378  
Jin, Hai 22  
Joncheere, Niels 592  
Jonckers, Viviane 592  
Jünemann, Konrad 732  
Jureta, Ivan J. 362

- Kaczmarek, Monika 271  
 Kalantar, Michael 162  
 Kalayci, Selim 54  
 Konstantinou, Alexander V. 162  
 Kotonya, Gerald 468  
 Kowalkiewicz, Marek 732  
 Kraft, Frank Michael 331  
 Krishnaswamy, Shonali 620  
 Kumar, Apurva 565  
 Kwok, Thomas 633
- Lamparter, Steffen 517  
 Lazovik, Alexander 70  
 Ledoux, Thomas 483  
 Lei, Hui 147  
 Lewis, Grace A. 678  
 Leymann, Frank 100  
 Li, Lily 347  
 Li, Xin Hui 649  
 Li, Ying 649  
 Liang, Qianhui 407  
 Lin, Kwei-Jay 180  
 Ling, Sea 210  
 Lingenfelder, Christoph 147  
 Liu, Tian Cheng 649  
 Liu, Yan 225  
 Liu, Zhen 538  
 Lohmann, Niels 331  
 Lovera, Marco 599  
 Ludwig, André 626  
 Luthria, Haresh 256
- Mancioppi, Michele 302  
 Maraikar, Ziyang 70  
 Marchetti, Eda 524  
 Mateescu, Radu 84  
 Mayer, Philip 545  
 Mazzoleni, Pietro 286  
 Mecella, Massimo 726  
 Mikalsen, Thomas 422  
 Mohindra, Ajay 633  
 Morris, Ed 678  
 Moser, Simon 100  
 Mosincat, Adina 241  
 Motahari Nezhad, Hamid R. 724  
 Moun gla, Hassine 195  
 Mukherjee, Debdoot 378  
 Mukhija, Arun 558  
 Müller, Carlos 394
- Narendra, Nanjangud C. 691  
 Ngu, Anne H.H. 317  
 Nourine, Lhouari 38
- Paci, Federica 116  
 Panahi, Mark 180  
 Papazoglou, Mike P. 302  
 Paradkar, Amit 551  
 Pasquale, Liliana 614  
 Pezzullo, Alessandro 726  
 Phan, Tan 5  
 Pinel, Florian 722  
 Podorozhny, Rodion 317  
 Poizat, Pascal 84, 530  
 Polini, Andrea 524  
 Ponnalagu, Karthikeyan 691
- Rabhi, Fethi 256  
 Ragab Hassen, Ramy 38  
 Ranganathan, Anand 538  
 Raverdy, Pierre-Guillaume 195  
 Resinas, Manuel 394  
 Riabov, Anton 538  
 Rivierre, Nicolas 483  
 Robinson, Daniel 468  
 Rosenblum, David S. 558  
 Rospocher, Marco 132  
 Rouvellou, Isabelle 422  
 Ruiz-Cortés, Antonio 394  
 Russo, Alessandro 726
- Sadjadi, S. Masoud 54  
 Saint-Paul, Regis 724  
 Salaün, Gwen 84  
 Santhanam, Ganesh Ram 453  
 Saponaro, Costantino 726  
 Schneider, Jean-Guy 5  
 Schroeder, Andreas 545  
 Schwan, Karsten 438  
 Serafini, Luciano 132  
 Silva-Lepe, Ignacio 422  
 Simanta, Soumya 678  
 Sinha, Avik 551  
 Smith, Dennis 678  
 Speiser, Sebastian 517  
 Srivastava, Biplav 286  
 Steller, Luke 620  
 Subramanian, Revathi 422  
 Sun, Haiyang 579  
 Sun, Yuqing 116  
 Sundaresan, Neel 2



- Tai, Stefan 517  
 Tan, Min'an 225  
 Tanaka, Masahiro 572  
 Tanelli, Mara 599  
 Tari, Zahir 585, 730  
 Taylor, Kerry 347  
 Tonella, Paolo 132  
 Totok, Alexander A. 162  
 Toumani, Farouk 38  
 Truong, Hong-Linh 607  
  
 Uchitel, Sebastian 558  
  
 van den Heuvel, Willem-Jan 302  
 Van Der Straeten, Ragnhild 592  
 Vanhatalo, Jussi 100  
 Varela, Leonardo 180  
 Vergo, John 722  
 Viswanathan, Balaji 54  
 Völzer, Hagen 100  
 Vosshall, Peter 1  
  
 Wang, Hao 664  
 Wang, Xin 579  
 Wibisono, Waskitho 210  
 Wilson, Kirk 5  
  
 Yan, Jun 22  
 Yang, Bo 664  
 Yang, Jian 579  
 Yang, Yang 511  
 Yang, Yun 22  
  
 Zapletal, Marco 498  
 Zaslavsky, Arkady 210  
 Zeng, Liangzhao 147, 317  
 Zhang, Jing 180  
 Zhang, Li 599  
 Zhang, Man 505  
 Zhang, Xiang 511  
 Zhang, Yanchun 579  
 Zhang, Yue 180  
 Zyskowski, Dominik 271