Lecture Notes in Computer Science 7084

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Services Science

Subline of Lectures Notes in Computer Science

Gerti Kappel Zakaria Maamar
Hamid R. Motahari-Nezhad (Eds.)

Service-Oriented Computing

9th International Conference, ICSOC 2011
Paphos, Cyprus, December 5-8, 2011
Proceedings

Springer

Volume Editors

Gerti Kappel
Vienna University of Technology
Institute of Software Technology
and Interactive Systems
Favoritenstraße 9-11/188
1040 Vienna, Austria
E-mail: gerti@big.tuwien.ac.at

Zakaria Maamar
Zayed University
College of Information Technology
P.O. Box 19282, Dubai, UAE
E-mail: zakaria.maamar@zu.ac.ae

Hamid R. Motahari-Nezhad
HP Labs - Services Research Lab
1501 Page Mill Road
Palo Alto, CA 94304, USA
E-mail: hamid-reza.motahari-nezhad@hp.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25534-2 e-ISBN 978-3-642-25535-9
DOI 10.1007/978-3-642-25535-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011941148

CR Subject Classification (1998): D.2, C.2, H.4, H.3, H.5, J.1, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Welcome to the 9th International Conference on Service-Oriented Computing (ICSOC 2011), held in Paphos, Cyprus, December 5–8, 2011. These proceedings contain high-quality research papers, both long and short, that showcase the latest developments in the ever-growing field of service-oriented computing.

Since the first meeting in 2003, ICSOC has become the premier forum for academics and industry researchers and practitioners to report and share groundbreaking works in service-oriented computing. ICSOC 2011 aimed at examining the research opportunities being offered by the possible blend of service-oriented computing with cloud computing. *Service-oriented and cloud computing* was thus the main theme of ICSOC 2011. Questions like how does service-oriented computing support the transition to cloud-based solutions, and how does it support infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) were discussed during the conference.

This year's call for research papers attracted a total of 184 full submissions from 35 countries spanning all continents of the world. All submitted papers were reviewed in detail by members of the Program Committee, which was composed of experts in the field of service-oriented and cloud computing from 21 countries. Based on their reviews, 30 submissions were accepted as full papers, giving an acceptance rate of 16%. Additionally, 24 submissions were accepted as short papers. The conference program was complemented by outstanding keynotes, industry paper presentations, demonstrations, and a panel discussion as well as a PhD Symposium and a collection of workshops.

We would like to express our gratitude to all the institutions and sponsors that supported ICSOC 2011, namely, the Computer Science Department of the University of Cyprus, Hewlett Packard, IBM Research, Salesforce, Springer, and ServTech. These proceedings and this conference would not have been possible without the expertise and dedication of the members of the Program Committee. We are also indebted to the General Chairs of ICSOC 2011 (Mohand-Said Hacid, Winfried Lamersdorf, and George Papadopoulos), to the Chairs of the different tracks (Anis Charfi, Sven Graupner, Yücel Karabulut, Sam Guinea, Florian Rosenberg, Youakim Badr, Francisco Curbera, Michael Q. Sheng, Cesare Pautasso, Sonia Ben Mokhtar, Leandro Krug Wives, Ivan Bedini, Yacine Atif, Rainer Unland, Christos Mettouris, and Dieter Mayrhofer), to the external reviewers, to the local organizers, and last but not least to the members of

the conference Steering Committee. All of them helped to make ICSOC 2011 a success. Finally, special thanks go to all the researchers and students who contributed with their work and participated in the conference. We hope that you find the papers in these proceedings as stimulating as we did.

September 2011

Gerti Kappel
Zakaria Maamar
Hamid R. Motahari-Nezhad

Organization

Honorary General Chair

Mohand-Said Hacid	University of Lyon, France

General Chairs

Winfried Lamersdorf	University of Hamburg, Germany
George Papadopoulos	University of Cyprus, Cyprus

Program Chairs

Gerti Kappel	Vienna University of Technology, Austria
Zakaria Maamar	Zayed University, UAE
Hamid R. Motahari-Nezhad	HP Labs, USA

Workshop Chairs

Mohamed Jmaiel	University of Sfax, Tunisia
George Pallis	University of Cyprus, Cyprus

Industry Chairs

Anis Charfi	SAP, Germany
Sven Graupner	HP Labs, USA
Yücel Karabulut	SAP, USA

Demonstration Chairs

Sam Guinea	Politecnico di Milano, Italy
Florian Rosenberg	IBM Research, USA

Panel Chairs

Youakim Badr	The Pennsylvania State University, USA
Francisco Curbera	IBM T.J. Watson Research Center, USA

PhD Symposium Chairs

Michael Q. Sheng	Adelaide University, Australia
Cesare Pautasso	University of Lugano, Switzerland
Sonia Ben Mokhtar	University College London, UK

Publicity Chairs

Leandro Krug Wives	UFRGS, Brazil
Ivan Bedini	Alcatel-Lucent Bell Labs, Ireland
Yacine Atif	UAE University, UAE
Rainer Unland	University of Duisburg-Essen, Germany

Organizing Committee

Christos Mettouris	University of Cyprus, Cyprus

Publication Chair

Dieter Mayrhofer	Vienna University of Technology, Austria

Program Committee

Marco Aiello	University of Groningen, The Netherlands
Rama Akkiraju	IBM T.J. Watson Research Center, USA
Álvaro Arenas	Instituto de Empresa Business School, Spain
Ebrahim Bagheri	Athabasca University, Canada
Luciano Baresi	Politecnico di Milano, Italy
Claudio Bartolini	HP Labs, USA
Samik Basu	Iowa State University, USA
Sujoy Basu	HP Labs, USA
Boualem Benatallah	University of New South Wales, Australia
Salima Benbernou	Unversité Paris Descartes, France
Antonia Bertolino	ISTI-CNR, Italy
Walter Binder	University of Lugano, Switzerland
Athman Bouguettaya	RMIT University, Australia
Christoph Bussler	Xtime Inc., USA
Manuel Carro	Universidad Politécnica de Madrid, Spain
Shiping Chen	CSIRO ICT, Australia

Lawrence Chung	University of Texas at Dallas, USA
Emmanuel Coquery	Université Claude Bernard Lyon 1, France
Francisco Curbera	IBM T.J. Watson Research Center, USA
Vincenzo D'Andrea	University of Trento, Italy
Florian Daniel	University of Trento, Italy
Flavio De Paoli	Università di Milano-Bicocca, Italy
Frédéric Desprez	INRIA, France
Khalil Drira	LAAS-CNRS, France
Marlon Dumas	University of Tartu, Estonia
Schahram Dustdar	Vienna University of Technology, Austria
Gregor Engels	University of Paderborn, Germany
Abdelkarim Erradi	Qatar University, Qatar
Rik Eshuis	Eindhoven University of Technology, The Netherlands
Noura Faci	Université Claude Bernard Lyon 1, France
Andreas Friesen	SAP Research, Germany
Hiroaki Fukuda	Keio University, Japan
Dragan Gašević	Athabasca University, Canada
Carlo Ghezzi	Politecnico di Milano, Italy
Paolo Giorgini	University of Trento, Italy
Sven Graupner	HP Labs, USA
Paul Grefen	Eindhoven University of Technology, The Netherlands
Hakim Hacid	Alcatel-Lucent Bell Labs, France
Mohand-Said Hacid	Université Claude Bernard Lyon 1, France
Peng Han	Chongqing Academy of Science and Technology, China
Jos van Hillegersberg	University of Twente, The Netherlands
Valérie Issarny	INRIA Paris-Rocquencourt, France
Hans-Arno Jacobsen	University of Toronto, Canada
Rania Y. Khalaf	IBM T.J. Watson Research Center, USA
Markus Kirchberg	HP Labs/National. University of Singapore, Singapore
Woralak Kongdenfha	Naresuan University, Thailand
Gerald Kotonya	Lancaster University, UK
Jeffrey T. Kreulen	IBM Almaden Research Center, USA
Patricia Lago	VU University Amsterdam, The Netherlands
Francesco Lelli	ERISS Tilburg, The Netherlands
Frank Leymann	University of Stuttgart, Germany
Jun Li	HP Labs, USA
Fu-ren Lin	National Tsing Hua University, China
Lin Liu	National Tsing Hua University, China
Xumin Liu	Rochester Institute of Technology, USA
Heiko Ludwig	IBM Almaden Research Center, USA

Paul P. Maglio	IBM Almaden Research Center, USA
Zaki Malik	Wayne State University, USA
Wathiq Mansoor	American University in Dubai, UAE
Michael Maximilien	IBM Almaden Research Center, USA
Massimo Mecella	Sapienza Università di Roma, Italy
Luis Miguel Vaquero	HP Labs, UK
Michaël Mrissa	Université Claude Bernard Lyon 1, France
Nanjangud C. Narendra	IBM Research India, India
Surya Nepal	CSIRO, Australia
Olga Ormandjieva	Concordia University, Canada
Guadalupe Ortiz	University of Cádiz, Spain
Helen Paik	University of New South Wales, Australia
Christian Pérez	INRIA, France
Radha Krishna Pisipati	INFOSYS Technologies Ltd., India
Julien Ponge	INSA Lyon, France
Frank Puhlmann	inubit AG, Germany
Mu Qiao	The Pennsylvania State University, USA
Robin Qiu	The Pennsylvania State University, USA
Manfred Reichert	University of Ulm, Germany
Wolfgang Reisig	Humboldt-Universität zu Berlin, Germany
Colette Rolland	Université Paris 1 Panthéon Sorbonne, France
Florian Rosenberg	IBM Research, USA
Gustavo Rossi	Universidad Nacional de La Plata, Argentina
Antonio Ruiz-Cortés	University of Seville, Spain
S. Masoud Sadjadi	Florida International University, USA
Régis Saint-Paul	Create-Net, Italy
Jakka Sairamesh	360Fresh Inc., USA
Ignacio Silva-Lepe	IBM Research, USA
Munindar P. Singh	North Carolina State University, USA
George Spanoudakis	City University London, UK
Bryan Stephenson	HP Labs, USA
Eleni Stroulia	University of Alberta, Canada
Jianwen Su	UC Santa Barbara, USA
Stefan Tai	Karlsruhe Institute of Technology (KIT), Germany
Wei Tan	IBM T.J. Watson Research Center, USA
Zahir Tari	RMIT University, Australia
Beatriz Toledo	UNICAMP, Brazil
Farouk Toumani	Blaise Pascal University, France
Peter Tröger	University of Potsdam, Germany
Srikumar Venugopal	University of New South Wales, Australia
Changzhou Wang	Boeing, USA
Yan Wang	Macquarie University, Australia
Bruno Wassermann	University College London, UK

Ingo Weber	University of New South Wales, Australia
Mathias Weske	University of Potsdam, Germany
Karsten Wolf	University of Rostock, Germany
Lai Xu	Bournemouth University, UK
Ramin Yahyapour	TU Dortmund, Germany
Zheng Yan	Aalto University, Finland/Xidian University, China
Jian Yang	Macquarie University, Australia
Konstantinos Zachos	City University London, UK
Hossein Zadeh	RMIT, Australia
Alex Zhang	HP Labs, USA
Weiliang Zhao	Macquarie University, Australia
Andrea Zisman	City University London, UK

External Reviewers

Rahul Akolkar	Cristian Klein	Sergio Segura
Mohsen Asadi	Bardia Mohabbati	Aleksander Slominski
Marko Bošković	Jonathan Munson	Paul de Vrieze
Pablo Fernández	José Antonio Parejo	
Sinem Güven	Marcus Roy	

Table of Contents

Research Papers – Long

Business Process Modeling

Quality of Service 1

Formal Methods

XaaS Computing

Service Discovery

Service Science and Management

Service Security and Trust

Service Monitoring

Service Composition

Quality of Service 2

Research Papers – Short

Business Process Modeling

XaaS Computing

Quality of Service

Service Runtime Infrastructures

Service Migration and Adoption

Service Composition

Service Applications

Computing Degree of Parallelism for BPMN Processes*

Yutian Sun and Jianwen Su

Department of Computer Science,
University of California, Santa Barbara
{sun,su}@cs.ucsb.edu

Abstract. For sequential processes and workflows (i.e., pipelined tasks), each enactment (process instance) only has one task being performed at each time instant. When a process allows tasks to be performed in parallel, an enactment may have a number of tasks being performed concurrently and this number may change in time. We define the "degree of parallelism" of a process as the maximum number of tasks to be performed concurrently during an execution of the process. This paper initiates a study on computing degree of parallelism for three classes of BPMN processes, which are defined based on the use of BPMN gateways. For each class, an algorithm for computing degree of parallelism is presented. In particular, the algorithms for "homogeneous" and acyclic "choice-less" processes (respectively) have polynomial time complexity, while the algorithm for "asynchronous" processes runs in exponential time.

1 Introduction

There has been an increasing interest in developing techniques for supporting business processes in research communities (e.g., recent conferences/workshops including BPM, COOPIS, ICSOC, ...). A business process is an assembly of tasks (performed by human or systems) to accomplish a business goal such as handling a loan application, approving a permit or treating a patient. The emergence of data management tools in the early 1980's brought the concept of workflow systems to assist execution of business processes in an ad hoc manner. IT innovations in the last decade have been exerting a growing pressure to increase automation in design, operation, and management of business processes. Recent research in this area focused on modeling approaches (e.g., [12,3,21,22,10,1]), verifying properties of business processes and workflow (e.g., [19,20,7]), etc. In this paper, we study the problem of computing the maximum number of tasks that are to be performed in parallel, which can provide useful information to execution planning for processes or workflow [14,24].

Performing business tasks requires resources [16,17] including data, software systems, devices, and in particular human. Resource planning is essential in business process (and workflow) execution management. For processes (workflow) with sequentially arranged tasks (i.e., pipelined tasks), each process instance has at most one task to be performed at one time; the amount of resources needed can be roughly determined

* Work supported in part by NSF grant IIS-0812578 and a grant from IBM.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 1–15, 2011.

by the process initiation rate, the number of tasks in the process, and the amount of work needed for each task. In this paper we focus on calculating the number of tasks that are performed simultaneously, this information provides a needed input to resource estimation.

When a process allows tasks to be performed in parallel, an enactment (process instance) may have a number of tasks to be performed concurrently and this number may change in time. We introduce a new concept "degree of parallelism" as the maximum number of tasks to be performed concurrently during a process execution, i.e., the peak demand on tasks to be performed. This paper initiates a study on computing degree of parallelism for business processes specified in BPMN [4].

Degree of parallelism is a worst case metric for business processes and can provide useful guidance to process modeling and execution planning. For example, some processes may have unbounded degrees, i.e., their peak use exceeds any fixed number. It is quite likely that such processes are results of modeling mistakes. More importantly, the peak time information on tasks could help in planning the needed resources (including human) for the execution of defined business processes.

Technically, this paper defines a formal model for processes (or workflows). The core building blocks in the model are adopted and/or generalized from BPMN constructs; in addition, our model also incorporates an expected duration for each task. The semantics resembles the Petri nets based semantics presented in [8]. The modeling of task durations makes the model closer to real world processes, e.g., healthcare treatment protocols (processes). We formally define the notion of degree of parallelism on this model, and present the following new technical results on three new subclasses of processes based on different combinations of BPMN gateways:

1. For "homogeneous" processes (that use only one type of gateways) a polynomial time algorithm is developed that computes the degree of parallelism.
2. For acyclic "choice-less" processes that does not allow choice gateways nor cycles, we present a polynomial time algorithm for processes in this subclass (the time complexity is linear in the sum of all task durations in the process).
3. We also consider asynchronous processes that use only two types of BPMN gateways: exclusive-merge and parallel-split. By mapping such processes to "process graphs", we show an algorithm to compute the degree of parallelism. The complexity of the algorithm is exponential time in general, but quadratic if the process contains at most one cycle. The general case solution answers an open problem in [13], and the quadratic result improves the cubic time result in [13].

This paper is organized as follows. The formal model and the key notion of degree of parallelism for processes are presented in Section 2. Sections 3, 4, and 5 focus on homogeneous, acyclic choice-less, and asynchronous processes, respectively. Related work is discussed in Section 6, and conclusions are included in Section 7.

2 A Formal Model for Processes

In this section, we introduce a formal model for BPMN processes (or workflows), the key notions include "process", "(pre-)snapshot", "derivation", and "reduction".

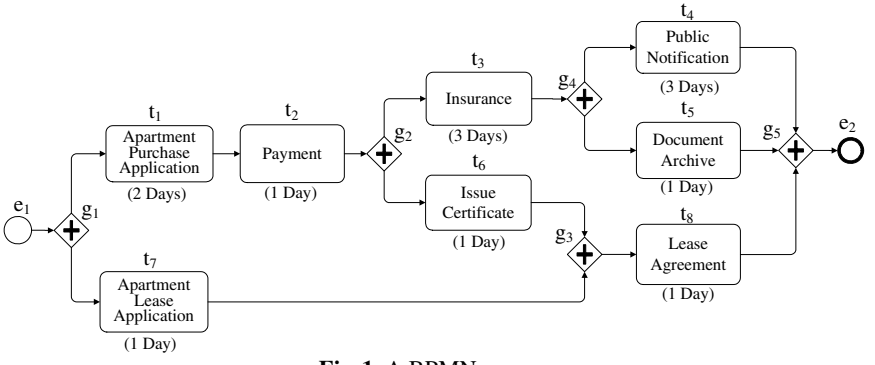


Fig. 1. A BPMN process

The semantics resembles the Petri net semantics for BPMN are presented in [8]. We also define the central notion of "degree of parallelism" used in this paper.

In BPMN [4], a process is modeled as a graph whose nodes and edges are of different types. In this paper, we focus on one type of edges corresponding to "sequence flow" in BPMN, and three types of nodes: "event", "task", and "gateway".

We consider two classes of *events* in BPMN: *start* and *end* events. A start (event) node initializes a process by sending out a "flow front" (or an active point of execution) to the next node. When all flow fronts reach their end (event) nodes, the process ends. A *task* (node) represents an atomic unit of work.

A *gateway* node in a process alters the current execution path (e.g., by choosing an alternative path or proceeding on all paths). There are four kinds of frequently used gateways in BPMN: (exclusive-)*choice*, (exclusive-)*merge*, (parallel-)*split* and (parallel-)*join*. Choice and merge gateways allow a flow front in a process to follow one of several alternatives (choice) or choose only one flow front from possibly several incoming edges to continue (merge). Split and join gateways, on the other hand, forward a flow front to every outgoing edge for parallel execution (split) or synchronize flow fronts from all incoming edges and combine them into one (join).

Example 2.1. Fig. 1 shows an example BPMN process, which combines purchasing an apartment and putting it out for lease (in China where leasing arrangements need an approval from the city office for real estate management). The process begins from a start event (e_1) and immediately forks into two paths by a split gateway g_1. The upper and lower paths represent the purchase and lease sub-processes respectively. The applicant files the purchase (t_1) and lease (t_7) applications. The expected duration of each task is shown below the task node in the figure, e.g., t_1 would take 2 days while t_7 only 1 day. After paying a purchasing transaction fee (t_2), the applicant obtains the certificate (t_6). Together with the leasing application, the applicant can finish a lease contract with the tenant (t_8). For the other branch, the applicant also needs to spend 3 days on getting an insurance policy (t_3). Once this is done, the housing office will make a public notification (t_4) for additional 3 days as required by law before archiving all the documents (t_5). Finally, all flow fronts will synchronize at the join gateway g_5; the process will end when reaching end event e_2. ∎

In our formal model, a "process" is a graph with edges corresponding to control-flow transitions and nodes representing start/end events, tasks, or gateways. Similarly, a gateway node alters the execution path (choosing an alternative path or following parallel paths). Our model includes two kinds of gateways that are more general than BPMN gateways: *exclusive* (denoted as ⊗) and *parallel* (denoted as ⊕). A ⊗-gateway essentially combines a merge gateway and a choice gateway, and a ⊕-gateway is a join gateway followed by a split gateway. Specifically, a ⊗-gateway node passes an incoming flow front on an incoming edge immediately to one of the outgoing edges to continue the flow front. A ⊕-gateway node, on the other hand, waits until one flow front from each incoming edge arrives, merges them into one flow front, and then split it again to pass a flow front to each of its outgoing edges. Note that when the number of incoming/outgoing edges is 1, ⊗- and ⊕-gateways degenerate to BPMN gateways.

Our model associates a duration to each node to indicate the typical length for the node to complete. Without loss of generality, the duration of each gateway or event node is always 0 (it takes no time to complete). A task node takes some time (> 0) to perform before finishing. In Fig. 1, durations are shown in parentheses below task nodes.

For the technical development, we use *indeg* and *outdeg* to denote the number of incoming edges and outgoing edges of a node respectively. Let **T** be the set consisting of the following types: ○ (start), ● (end), □ (task), ⊗ (exclusive gateway), and ⊕ (parallel gateway). Let $\mathbb{N}$ be the set of natural numbers.

Definition: A *process* (*with durations*) is a tuple $P = (V, s, F, E, \tau, \delta)$, where

- V is a (finite) set of *nodes*,
- $s \in V$ and $F \subseteq (V - \{s\})$ are the *start* node and a set of *end* nodes (resp.),
- $\tau : V \to \mathbf{T}$ is a mapping that assigns each node in V a type such that $\tau(s) =$ "○", $\tau(v) =$ "●" for each $v \in F$, and $\tau(v)$ is not "○" nor "●" for each $v \in V - F - \{s\}$,
- $\delta : V \to \mathbb{N}$ is a mapping that assigns each node a *duration* such that for each $v \in V$, $\delta(v) > 0$ iff $\tau(v) =$ "□" (v is a task node), and
- $E \subseteq (V - F) \times (V - \{s\})$ is a set of *transitions* satisfying all conditions listed below:
  1. For the start node s, $outdeg(s) = 1$ and $indeg(s) = 0$,
  2. For each end node $v \in F$, $indeg(v) = 1$ and $outdeg(v) = 0$, and
  3. For each task node v, $indeg(v) = outdeg(v) = 1$.

Given a process $P = (V, s, F, E, \tau, \delta)$, a *cycle* (*of size* $n \in \mathbb{N}$) is a sequence $v_1, v_2, ..., v_n$ such that for each $i \in [1..n]$, v_i is a node in V, and $(v_i, v_{(i \bmod n)+1}) \in E$. A process is *acyclic* if it contains no cycles.

The graph shown in Fig. 1 can also be viewed as a process in our model, where e_1 and e_2 are the start and end nodes (resp.), g_i's ($1 \leqslant i \leqslant 5$) are ⊕-gateway nodes, and t_i's ($1 \leqslant i \leqslant 8$) are task nodes with non-0 durations.

In general, a process can be nondeterministic and/or have tasks performing in parallel. For example, if a flow front goes into a ⊗-gateway, the gateway can choose nondeterministically an outgoing edge to route the flow front. Also, a process can spawn several flow fronts during the execution due to ⊕-gateway nodes. The goal of this paper is to compute the maximum number of tasks that may run in parallel.

In order to define the notions precisely for algorithm development, we need to provide a semantics for processes. We introduce a pair of notions "pre-snapshots" and "snapshots" below that are used to formulate the semantics.

In the remainder of this section, let $P = (V, s, F, E, \tau, \delta)$ be some process. A *flow front* is a triple (u, v, n), where $(u, v) \in E$ is an edge (transition) in P and n is a (possibly negative) integer such that $n \leqslant \delta(u)$. Intuitively, a positive number n denotes the remaining time needed to complete the node u or "time-to-live" for u. When $n \leqslant 0$, u is completed and the flow front is ready to move forward through v in the process. Since the duraton of a non-task node is always 0, a flow front (u, v, n) originating at a non-task node u can proceed unless v is a $\oplus$-gateway node.

A *pre-snapshot* of the process P is a multiset of flow fronts. Note that duplicates are allowed in a pre-snapshot. The singleton multiset $\{(s, u, 0)\}$ is an *initial* pre-snapshot where s is the start node. Given a pre-snapshot S of P, a node v in P is *ready* (*to activate*) in S if one of the following holds:

- v is an end/task/$\otimes$-gateway node and a flow front (u, v, n) is in S for some $n \leqslant 0$, or
- v is a $\oplus$-gateway node and for each incoming edge (u, v) into v, (u, v, n) is a flow front in S for some $n \leqslant 0$.

Example 2.2. Consider the process shown in Fig. 1. The triples $(e_1, g_1, 0)$, $(t_1, t_2, 2)$ are flow fronts of the process, $(e_1, g_1, 1)$ is not a flow front since the duration of e_1 is $0 < 1$, nor is $(t_1, t_3, 2)$ since (t_1, t_3) is not an edge in the process. The following are pre-snapshots: $\{(e_1, g_1, 0)\}$, $\{(g_1, t_1, -2)\}$, $\{(t_1, t_2, 2), (t_7, g_3, 1)\}$, and also $\{(t_1, t_2, 2), (t_2, g_2, -1), (t_7, g_3, 1)\}$. In the pre-snapshot $\{(t_1, t_2, 2), (t_2, g_2, -1), (t_7, g_3, 1)\}$, task t_2 has completed, tasks t_1 and t_7 are still running in parallel, and node g_2 is ready. ∎

If a node v is ready in a pre-snapshot S, we can proceed a (or more) flow front(s) to the node v to *derive* a new pre-snapshot S' as follows.

- If v is an end node and (u, v, n) is in S where $n \leqslant 0$, then $S' = S - \{(u, v, n)\}$.
- If v is a task or $\otimes$-gateway node and (u, v, n) is in S where $n \leqslant 0$, then $S' = (S - \{(u, v, n)\}) \cup \{(v, w, \delta(v))\}$ where $\delta(v)$ is the duration of v and (v, w) is an edge leaving v in P. (When v is $\otimes$-gateway, w is nondeterministically selected.)
- If v is a $\oplus$-gateway node with all incoming edges from $u_1, ..., u_\ell$ and for each $i \in [1..\ell]$, (u_i, v, n_i) is in S where $n_i \leqslant 0$, then $S' = (S - \{(u_i, v, n_i) \mid 1 \leqslant i \leqslant \ell\}) \cup \{(v, w_i, \delta(v)) \mid (v, w_i)$ is an outgoing edge of v in $P\}$.

Example 2.3. Consider the process in Fig. 1. Since e_1 is the start node, the initial pre-snapshot is $\{(e_1, g_1, 0)\}$. Clearly, g_1 is ready and we can derive the pre-snapshot $\{(g_1, t_1, 0), (g_1, t_7, 0)\}$ since g_1 is a $\oplus$-gateway. Now both t_1 and t_7 become ready. We may derive the pre-snapshots $\{(t_1, t_2, 2), (g_1, t_7, 0)\}$, and then $\{(t_1, t_2, 2), (t_7, g_3, 1)\}$. At this point, no nodes are ready. ∎

We call a pre-snapshot of process P in which no nodes are ready a *snapshot*. In Example 2.3, $\{(t_1, t_2, 2), (t_7, g_3, 1)\}$ is a snapshot. In general, a pre-snapshot can always derive in a finite number of steps into a snapshot. We call the procedure of a pre-snapshot eventually deriving a snapshot a *reduction*.

From a snapshot, derivations cannot be made since no nodes are ready. At this time we can advance process operations by one time unit. Technically, let S be a snapshot and S' a pre-snapshot. S *task-derives* S' if $S' = \{(u, v, n - 1) \mid (u, v, n) \in S\}$.

Note that if a flow front has a positive time-to-live the time is decremented by 1, if the time-to-live is zero or negative, the resulting time may be negative. While this may be a useful information for measuring performance, we do not use the negative amounts in this paper. Also, as the task-derivation indicates, the scheduling algorithm for process tasks is an eager one—it performs the task immediately when the task becomes ready. It is interesting to examine alternative scheduling policies and explore their impact on, e.g., the degree of parallelism. But this is beyond the scope of the present paper.

Example 2.4. Continuing with Example 2.3, the snapshot $\{(t_1, t_2, 2), (t_7, g_3, 1)\}$ task-derives the pre-snapshot $\{(t_1, t_2, 1), (t_7, g_3, 0)\}$. The latter indicates that t_7 is completed but g_3 is not ready since it is a $\oplus$-gateway and the other incoming edge does not have a flow front. Therefore, $\{(t_1, t_2, 1), (t_7, g_3, 0)\}$ is also a snapshot, which further task-derives $\{(t_1, t_2, 0), (t_7, g_3, -1)\}$. Now task t_2 becomes ready. ∎

Definition: Let $n \in \mathbb{N}$ and $P = (V, s, F, E, \tau, \delta)$ be a process. An *enactment* of length n of P is a sequence $p = S_1 S_2 ... S_n$ such that for each $i \in [1..n]$, S_i is a snapshot, S_1 is a reduction from the initial pre-snapshot and for each $i \in [2..n]$, S_i is obtained from S_{i-1} by first applying task-derivation on S_{i-1} followed by a reduction. The enactment p is *complete* if $S_n = \emptyset$. The *semantics* of a process P is a set of all complete enactments.

Let S be a snapshot, the *active cardinality* of S, denoted as $|S|_{\text{active}}$, is the cardinality of the multiset $\{(u, v, n) \mid (u, v, n) \in S \text{ and } n \geqslant 1\}$. The active cardinality of S indicates the number of (currently) running tasks at the time of the snapshot.

Definition: The *degree (of parallelism)* of a process P, denoted as $\text{DP}(P)$, is the maximum active cardinality of a snapshot in some enactment of P.

The degree of process P reflects how much parallelism the execution of P allows, i.e., the maximum number of (active) flow fronts that can appear during the execution of P. Suppose the total amount of "work" in P is fixed. The greater $\text{DP}(P)$ is, the more resources operations of P will need. On the other hand, the availability of these resources will mean the total time to complete an enactment is shorter. This, however, does not mean the throughput of the business managing process P is automatically higher. To achieve operational efficiency under resource limitation, it may be possible to plan tasks in P in a way to lower the degree of parallelism while maintaining the throughput. The study on the degree of parallelism is an initial step towards understanding the issue of resource needs and constraints on tasks as specified in a process.

3 Homogeneous Processes

In this section, we focus on a subclass of processes, called "homogeneous processes", and present a polynomial time algorithm to compute the degree of parallelism.

A process is *homogeneous* if its gateway nodes only use one kind of gateway, either $\otimes$-gateway or $\oplus$-gateway, but not both. There are two flavors of homogeneous processes. A *parallel*-(or $\oplus$-)homogeneous process uses only $\oplus$-gateway while a *choice*-(or $\otimes$-)homogeneous process uses only $\otimes$-gateway.

Lemma 3.1. The degree of parallelism for each ⊗-homogeneous process is always 1.

From the semantics, it is easy to see that derivation and task-derivation from a pre-snapshot will not increase the cardinality, since at most one outgoing edge (transition) can be invoked for each node. Since the initial pre-snapshot only contains one element, the cardinality of each snapshot of each arbitrary ⊗-homogeneous process is always one, which bounds the degree of parallelism. The proof can be done by an induction.

Obviously, Lemma 3.1 fails for ⊕-homogeneous processes. In the remainder of this section, we only focus on the calculation of the degree of parallelism of ⊕-homogeneous processes. The process in Fig. 1 is a ⊕-homogeneous process.

Given a process P, a node v is *reachable* in P, if there exists an enactment $S_1 S_2 ... S_k$, such that v is ready either in the snapshot S_k, or in a pre-snapshot that can be derived from S_{k-1} and reduced to S_k.

Lemma 3.2. If every node in a ⊕-homogeneous process P is reachable, P is acyclic.

Proof: (Sketch) Let P be a ⊕-homogeneous process that contains a cycle C. Consider a sequence of pre-snapshots $S_1, ..., S_m$ such that (1) S_1 is initial, and for each $i \in [2..m]$, S_i is derived or task-derived from S_{i-1} in one step, (2) for some node v in C, v is ready in S_m, and (3) no other nodes in C that is ready in S_j for $j < m$. Since each node in P is reachable, it is possible to find a v and pre-snapshot sequence that satisfy (1)-(3). Clearly, v must have at least two incoming edges (one from the path and the other on the cycle) and thus a ⊕-gateway node. By the definition of derivition/task-derivition, some node on C must be ready in S_j for some $j < m$, a contradiction. ∎

Since a ⊕-homogeneous process is acyclic according to Lemma 3.2, each node will be added into a snapshot or pre-snapshot at most once. Thus, the degree of parallelism is finite and less than the total number of task nodes in the process.

Theorem 3.3. Given a ⊕-homogeneous process $P = (V, s, F, E, \tau, \delta)$, the degree of parallelism of P can be computed in $O(|V| \log |V|)$ time.

To establish Theorem 3.3, we develop an algorithm that simulates the execution of a process $P = (V, s, F, E, \tau, \delta)$ in computing its degree of parallelism. The simulation uses a priority queue to store all nodes that are currently running. When a node finishes, it is popped from the queue. Thus, the degree of the ⊕-homogeneous process is the maximum number of tasks that appear in the queue at some point during the simulation.

We use $[v, t]$ to denote an element in Q where t is the completion time for a node v. Entries of form $[v, t]$ in Q, are sorted according to the completion time t in the ascending order. The algorithm (Alg. 1) uses an array $RN(v)$ to record the number of remaining incoming nodes that haven't finished but are necessary for v to be performed.

Alg. 1 starts by placing $[s, 0]$ in Q which is analogous to the initial pre-snapshot. The array $RN(v)$ is initialized to $indeg(v)$ for each v. Every time an element $[v, t]$ is popped from Q indicates the completion of v at time t. (Since elements in Q are sorted by their end times, the one at the front of Q always has the earliest end time.) Once a node v finishes, the algorithm checks if there is an edge connecting from v to u, and if so $RN(u)$ is decremented by 1. If $RN(u)$ becomes 0, u starts to execute and therefore will

Algorithm 1. Compute Degree of a Parallel-Homogeneous Process

Input: A process $P = (V, s, F, E, \tau, \delta)$
Output: degree of parallelism $\text{DP}(P)$

```
 1: Initialize a priority queue Q to be empty;
 2: Q.enque([s, 0]);
 3: for each v ∈ V − {s} do
 4:     RN(v) := indeg(v);
 5: end for
 6: deg := 0;
 7: t_old := 0;
 8: while Q is not empty do
 9:     [v1, t1] := Q.deque();
10:     for each v2 ∈ {v | (v1, v) ∈ E} do
11:         RN(v2) := RN(v2) − 1;
12:         if RN(v2) = 0 then
13:             Q.enque([v2, t1 + δ(v2)]);
14:         end if
15:     end for
16:     if t_old ≠ t1 then
17:         #T := |{[v, t] | [v, t] ∈ Q ∧ v is a task node }|;
18:         deg :=max(deg, #T);
19:         t_old := t1;
20:     end if
21: end while
22: return DP(P) = deg;
```

Step	Q	Updated RN	$\#T$	DP	Step	Q	Updated RN	$\#T$	DP
1	$[e_1, 0]$		0	0	9	$[t_8, 5], [t_3, 6]$	$RN(t_8) = 0$	2	2
2	$[g_1, 0]$	$RN(g_1) = 0$	0	0	10	$[t_3, 6]$	$RN(g_5) = 2$	1	2
3	$[t_7, 1], [t_1, 2]$	$RN(t_7) = RN(t_1) = 0$	2	2	11	$[g_4, 6]$	$RN(g_4) = 0$	0	2
4	$[t_1, 2]$	$RN(g_3) = 1$	1	2	12	$[t_5, 7], [t_4, 9]$	$RN(t_5) = RN(t_4) = 0$	2	2
5	$[t_2, 3]$	$RN(t_2) = 0$	1	2	13	$[t_4, 9]$	$RN(g_5) = 1$	1	2
6	$[g_2, 3]$	$RN(g_2) = 0$	0	2	14	$[g_5, 9]$	$RN(g_5) = 0$	0	2
7	$[t_6, 4], [t_3, 6]$	$RN(t_6) = RN(t_3) = 0$	2	2	15	$[e_2, 9]$	$RN(e_2) = 2$	0	2
8	$[g_3, 4], [t_3, 6]$	$RN(g_3) = 0$	1	2	16	$\varnothing$		0	2

Fig. 2. Simulating the process in Example 2.1

be pushed into Q. During the simulation, let $\#T$ be the number of task nodes in Q. In all, the degree $\text{DP}(P)$ is the highest $\#T$ that appears during the entire simulation.

Fig. 2 illustrates the details of simulating the process in Example 2.1. It turns out that the degree of parallelism is 2 even though the process has 3 parallel branches (Fig. 1).

The complexity of Alg. 1 depends on the time to maintain the priority queue. Since the size of the queue can be at most $|V|$, the complexity of this algorithm is $O(|V| \log |V|)$.

4 Acyclic Choice-Less Processes

In this section, we introduce another subclass of BPMN processes, acyclic "choice-less processes" and focus on the computation of degree of parallelism for such processes. We present a polynomial time algorithm to compute the degree. Note that for acyclic processes, the degree is always finite.

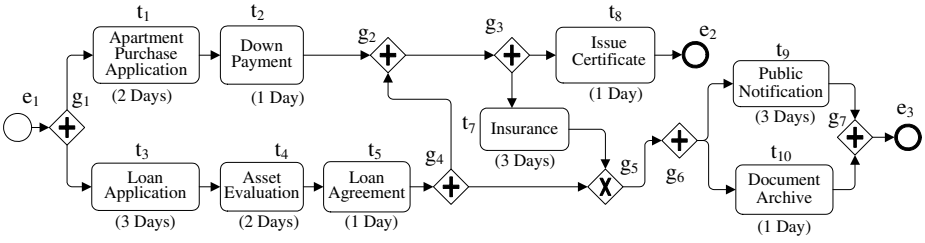


Fig. 3. An acyclic choice-less process

Definition: A process $P = (V, s, F, E, \tau, \delta)$ is *choice-less* if for each $v \in V$, $outdeg(v) = 1$ whenever $\tau(v) =$ "⊗".

Intuitively, a choice-less process contains no exclusive-decision gateway nodes. But it may contain exclusive-merge gateway (with one outgoing edge). These processes are used frequently in scientific workflows [2], where the focus is on computations that involve large amounts of datasets. Knowing the degree of parallelism of a scientific workflow would potentially help scheduling computations (i.e. tasks), especially for a cloud computation setting [11].

Example 4.1. Fig. 3 shows an example of acyclic choice-less process for purchasing an apartment with loan. The process begins with two branches: to apply for the apartment purchase (t_1) and pay the down payment (t_2), and to apply for the loan (t_3). After assessing the apartment (t_4), the bank decides to pay the rest of the balance (t_5) to complete the purchase (g_2). Once the loan is settled, the housing office will archive the documents (t_{10}) and make a public notification (t_9). Also, the office will give the certificate to the buyer (t_8) for the new ownership. After the customer purchases the insurance (t_7), the housing office will again archive the documents (t_{10}) and make a public notification (t_9). Note that t_{10} is invoked twice due to the presence of a ⊗-gateway (g_5). ∎

Theorem 4.2. Given an acyclic choice-less process $P = (V, s, F, E, \tau, \delta)$, the degree of parallelism of P can be computed in $O(|E| \log |V| + |E|L)$ time where L is the sum of durations of all task nodes in P.

In the remainder of this section, we discuss key ideas for proving Theorem 4.2. More details are provided in the online appendix [18].

The key idea to compute the degree of an acyclic choice-less process is to partition the process into smaller pieces, analyze the pieces, and then aggregate them together. We view each ⊕-gateway node as a pair of BPMN split and join gateways, all ⊗-gateway nodes are actually merge gateways due to the choice-less restriction. The following are the 3 main steps:

1. Decompose the process into segments according to join and merge gateway nodes.
2. For each segment, a list is computed to capture the parallelism information.
3. Combine all such lists and compute the degree for the input process.

In the ***first*** step, a process is chopped into segments. Each segment is separated by join and merge gateway nodes. To generate a segment, a depth-first search is used. We create

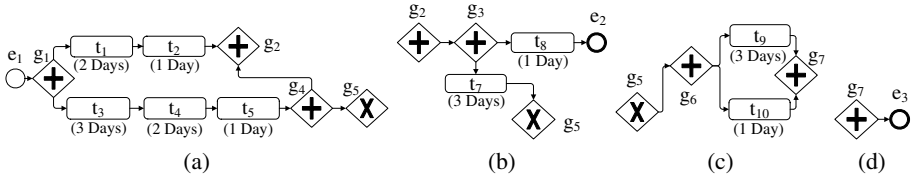


Fig. 4. Four segments of the process in Fig. 3

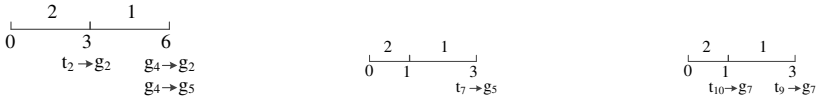


Fig. 5. Event point list

a new segment by traversing from the start node. when a join or merge gateway node is visited, it is marked as an *exit node* for the current segment, and starts a new segment. The node also plays the role of the *entry node* of the new segment. Since a join or merge gateway node has only one outgoing edge, each segment has only one entry node and may have several exit nodes.

Example 4.3. Fig. 4 shows all four segments of the process in Fig. 3. Fig. 4(a) has the entry node e_1 and two exit nodes g_2, g_5. Fig. 4(b) starts from g_2 and ends at e_2, g_5. Fig. 4(c) enters at g_5 and has one exit node g_7. Fig. 4(d) starts from g_7 and ends at e_3. ▮

In the ***second*** step, we compute the "parallelism" information of each segment, with a data structure *event point list*. An event point list contains two basic pieces of information: (1) the cardinality of the corresponding segment's enactment between two timestamps, and (2) the time the segment will reach its exit nodes and through which edge the segment will reach each exit node.

Example 4.4. Fig. 5 shows three event point lists generated according to the segments in Example 4.3. Fig. 5(a) corresponds to Fig. 4(a). From time 0 to 3, the degree is 2, then t_2 completes and invokes g_2. From time 3 to 6, only one flow front exists, and at timestamp 6, g_4 invokes g_2 and g_5. Fig. 5(b)(c) provide the similar event point lists corresponding to Fig. 4(b)(c), resp. The event point list of Fig. 4(d) is an empty list. ▮

Constructing event point lists is similar to the algorithm in Section 3. By mapping each entry node to an start node and each exit node to an end node, each segment is in fact a homogeneous process. Similar to Alg. 1, a priority queue can be used to simulate each segment. And the event point list can be derived according to $\#T$. When an exit node is popped out from the queue, this node, together with its incoming edge, will be recorded in the event point list.

In the remainder of this section, we may use term "event point list" and "segment" interchangeably to refer to the same object according to the context.

Once all event point lists are constructed, the ***third*** step combines the lists. Since the choice-less process is acyclic, a key observation is that all segments follow a topological order, i.e., a segment can only be invoked by its preceding segments.

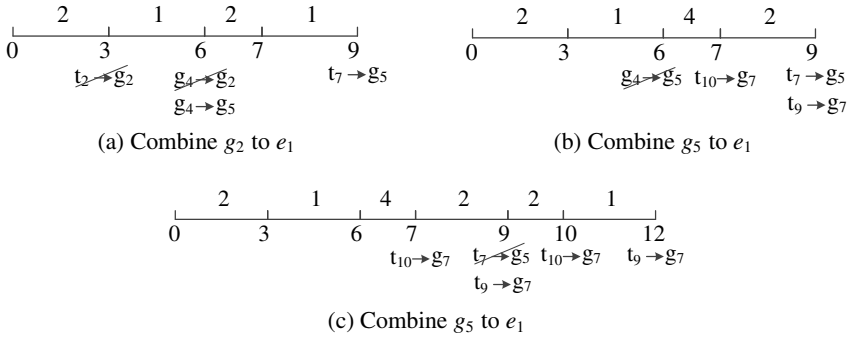


Fig. 6. Combination of event point lists

With the sorted segment sequence, we remove the second segment and combine its event point list into the first event point list. Note that this guarantees that the second segment can only be invoked by one segment, i.e., the first. This procedure repeats until only one event point list left in the end.

There are two types of event point lists to be combined. One starts from a join gateway (node) and the other from a merge gateway. The combination of these two types of event point lists to the first event point list need be handled differently.

If the second event point list's entry node is a join gateway, we first mark where this segment is invoked in the first event point list according to each different incoming edges from left to right. once all different incoming edges are marked, we combine the second event point list to the first one at the last timestamp where an edge is marked. Then we repeat the above steps until the second event point list cannot be combined any more. The reason to mark incoming edges is to simulate the synchronization property of join gateway. A join gateway can only continue once all its incoming edges are ready.

Example 4.5. The segment order for Example 4.4 is e_1, g_2, g_5, g_7. Now consider the second event point list that starts from join gateway g_2. Since g_2 has two incoming edges, (t_2, g_2) and (g_4, g_2), in the first event point list, we mark $t_2 \rightarrow g_2$ and $g_4 \rightarrow g_2$ and then combine the second event point list at time 6. Fig. 6(a) is the new event point list starts from e_1 and g_2 (segment) should be removed from the segment sequence. ∎

If the second event point list's entry node is a merge gateway, the combination is simpler. Since for each incoming edge of this kind of node, once a flow front arrives, the node immediately routes it to its outgoing edge. Thus when scanning the first event point list, once at some timestamp, the second segment is invoked, we can simply do the combination.

Example 4.6. After merging g_2 to e_1 in Example 4.5, the segment sequence is e_1, g_5, g_7. Now the second event point list starts from the merge gateway g_5. In the first event point list (Fig. 6(a)), there are two places that call g_5. Hence, two combinations are needed. Fig. 6(b) and (c) show the first and second combination respectively.

Now the only event point list left is the one with entry node g_7. Since g_7 leads an empty event point list, the final event point list is the same as the one in Fig. 6(c). ∎

The algorithm details are provided in the online appendix [18].

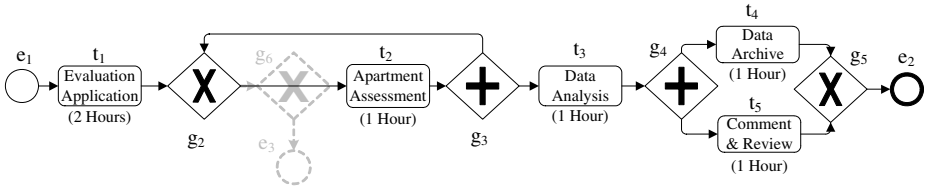


Fig. 7. Pre-sell permit approval process

5 Asynchronous Processes

In this section, we introduce the third subclass of BPMN processes, called "asynchronous processes", and present an algorithm to compute their degrees. Intuitively, an asynchronous process only includes split and merge gateways, i.e., it cannot do synchronization nor choices. It turns out that computing degree of parallelism for such processes is rather intricate, the time complexity of the algorithm is exponential.

Definition: A process $P = (V, s, F, E, \tau, \delta)$ is *asynchronous* if for each node $v \in V$, $outdeg(v) = 1$ whenever $\tau(v) =$ "$\otimes$" and $indeg(v) = 1$ whenever $\tau(v) =$ "$\oplus$".

From the definition, an asynchronous process includes only gateways nodes that are split gateway or merge gateway.

Example 5.1. Fig. 7 shows a process for apartment evaluation. If a developer is building apartments and plans to sell them, she needs a "pre-sell" permit from the city housing office. The office checks if the apartments are in good quality. An apartment quality evaluation process will start when an application (t_1) is received. Then the office staff will assess each apartment. If there is no more apartment to check, the process will end at g_6 and exit to e_3. Otherwise, evaluation modes to the next apartment (t_2). Once an apartment is assessed, the data will be send to the housing office asynchronously for analysis (t_3). After that, comment will be drawn (t_5) and data will be archived (t_4). ∎

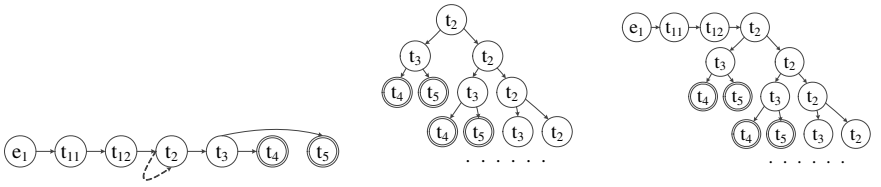
Technically, the process in the above example is not asynchronous due to the decision gateway (g_6). In order to simplify the analysis, we hide it from the process, link an edge directly from g_2 to t_2, and remove e_3 as well.

In the technical development, we use simplified graphs for asynchronous processes.

Definition: A (*process*) *graph* is a tuple (V, E, s, F) where V is a set of nodes containing the initial node s and a set F of final nodes, and $E \subseteq (V-F)\times V$ is a set of edges.

A *path* of size n of an process graph $G = (V, E, s, F)$ is a sequence of nodes $v_1 v_2 ... v_n$, where for each $i \in [1..n]$, $v_i \in V$, $v_1 = s$, and for each $i \in [1..(n-1)]$, $(v_i, v_{i+1}) \in E$. A path denotes one possible execution of the given process graph. However, in order to take all the possible executions into consideration, we pursue all paths in parallel. Let $D_n(G)$ denote the number of distinct paths of G with length n. We define the *degree* of G to be the max $D_n(G)$ for all $n \in \mathbb{N}$.

Lemma 5.2. Each asynchronous process P can be translated into a process graph G, such that the degree of P is the same as the degree of G.



(a) Process graph (b) Expanded paths from t_2 (c) Expanded paths from e_1

Fig. 8. Process graph and its expansion

Fig. 8(a) shows the process graph translated from the process in Fig. 7. We now can focus on process graphs and compute degree of an asynchronous process by computing degree of its corresponding graph.

A process graph G is said to be *bounded* if the degree of G is finite. The key results of the section are now stated below.

Theorem 5.3. Let P be an asynchronous process whose process graph has n nodes and m edges. Boundedness of degree of P can be decided in $O(m+n)$ time; if the degree is bounded, the degree can be computed in exponential time, and in $O(mn)$ time if P is acyclic or contains only one cycle.

Theorem 5.3 follows from the following two lemmas (Lemmas 5.4 and 5.5).

Lemma 5.4. (1) The degree of a process graph is bounded iff it does not contain two distinct cycles such that one connects to the other. (2) Given a process graph $G = (V, E, s, F)$, its boundedness can be determined in $O(|V| + |E|)$ time.

Lemma 5.4 is a slight variant of a result in [13] (the models are slightly different). Furthermore, given a process graph with at most one cycle (always bounded), an algorithm was presented in [13] to compute the degree in cubic time complexity. However, the general case was left open.

In the remainder of this section, we discuss a new algorithm that makes two improvements over the result in [13]: (1) it computes the degree for the general case, thus solves the open problem from [13], (2) when applying to acyclic and one-cycle graphs, the time complexity is quadratic, which improves the cubic result in [13].

Lemma 5.5. Given a bounded process graph $G = (V, E, s, F)$, the degree of G can be computed in exponential time, and in $O(|V||E|)$ time if G contains at most one cycle.

To compute the degree of a bounded process graph, we use the following steps:

1. Eliminate all cycles of the given process graph. For each node in the new graph, compute the numbers of reachable nodes in different depths and store these numbers in a list, called "*child list*". The method to compute each child list is according to a reversed topological order.
2. Add cycles back to the graph, with the result from step 1, compute the numbers of reachable nodes in different depths for those nodes that are inside cycles. Store these numbers in a list, called "*cycle child list*".

3. Remove all the cycles once more and compute the cycle child list for the source node. The degree of the corresponding process graph is the largest number of this cycle child list.

The detailed algorithm is rather involved and sketched in the online appendix [18].

6 Related Work

Our work is an extension of the work in [13] that focused on non-determinism of a simple graph model. Their model can be mapped to asynchronous processes with their degree of non-determinism coincides with degree of parallelism. The results reported in Section 5 extended their results and solve an open problem.

There were a stream of papers related to degree of non-determinism of finite state machines. These addressed the problems of boundedness [23,15], computing the degree [23,9,15], estimating the upper bound of the degree [23], and complexity bounded on this problem [15,5]. Although the problems are different from ours, it remains to explore whether these techniques can be used in solving our problem.

Our work is also related to workflow execution management. The work in [14] proposed a set of resource patterns for task allocation. While a language for specifying the resource allocation constraints was described in [16,17]. The study in [6] focused on authorization constraints and determining if a workflow can finish under such constraints. Static scheduling issues were studied in [24], where the authors developed an adaptive rescheduling strategy for grid workflow. Finally, while our problem seems relevant to parallel computing, it was not studied in the literature to the best of the authors' knowledge.

7 Conclusions

We focus on a subset of BPMN and examine the worst case number of parallel tasks and develop a set of preliminary results. It is still not clear how one would extend the algorithm to the full set of BPMN. This work also spawns many interesting questions related to planning business process execution. For example, given the resource requirements and cost functions, how can these algorithms be augmented to produce sufficient information for execution planning. Such problems are key to many business processes, e.g., in healthcare delivery. Clearly, this paper merely peeks into a broader topic concerning business operations planning and optimization.

References

1. Abiteboul, S., Segoufin, L., Vianu, V.: Modeling and verifying active xml artifacts. Data Engineering Bulletin 32(3), 10–15 (2009)
2. Barker, A., van Hemert, J.: Scientific Workflow: A Survey and Research Directions. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 746–753. Springer, Heidelberg (2008)

3. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
4. Business Process Model and Notation (BPMN), version 2.0 (January 2011), http://www.omg.org/spec/BPMN/2.0/PDF
5. Chan, T., Ibarra, O.H.: On the finite-valuedness problem for sequential machines. Theoretical Computer Science 23(1), 95–101 (1983)
6. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: Proc. 10th ACM Symp. on Access Control Models and Technologies, SACMAT (2005)
7. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. Int. Conf. on Database Theory (ICDT), pp. 252–267 (2009)
8. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in bpmn. Inf. Softw. Technol. 50, 1281–1294 (2008)
9. Gurari, E.M., Ibarra, O.H.: A note on finite-valued and finitely ambiguous transducers. Theory of Computing Systems 16(1), 61–66 (1983)
10. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath III, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Introducing the guard-stage-milestone approach to specifying business entity lifecycles. In: Proc. Workshop on Web Services and Formal Methods (WS-FM). Springer, Heidelberg (2010)
11. Juve, G., Deelman, E.: Scientific workflows and clouds. Crossroads 16(3) (March 2010)
12. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
13. Potapova, A., Su, J.: On nondeterministic workflow executions. In: Proc. Workshop on Web Services and Formal Methods, WSFM (2010)
14. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005)
15. Sakarovitch, J., de Souza, R.: On the Decidability of Bounded Valuedness for Transducers. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 588–600. Springer, Heidelberg (2008)
16. Senkul, P., Kifer, M., Toroslu, I.H.: A logical framework for scheduling workflows under resource allocation constraints. In: Proc. 28th Int. Conf. on Very Large Data Bases (2002)
17. Senkul, P., Toroslu, I.H.: An architecture for workflow scheduling under resource allocation constraints. Information Systems 30, 399–422 (2005)
18. Sun, Y., Su, J.: On-line Appendix to the Paper "Computing Degree of Parallelism for BPMN Processes" (2011), http://www.cs.ucsb.edu/~su/papers/2011/AppendixICSOC2011.pdf
19. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248. Springer, Heidelberg (1997)
20. van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, p. 161. Springer, Heidelberg (2000)
21. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Tennenholtz, M. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)
22. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Information Systems 30(4), 245–275 (2005)
23. Weber, A.: On the valuedness of finite transducers. Acta Inf. 27, 749–780 (1990)
24. Yu, Z., Shi, W.: An adaptive rescheduling strategy for grid workflow applications. In: Proc. IPDPS (2007)

State Propagation in Abstracted Business Processes

Sergey Smirnov, Armin Zamani Farahani, and Mathias Weske

Hasso Plattner Institute, Potsdam, Germany
{sergey.smirnov,mathias.weske}@hpi.uni-potsdam.de,
armin.zamanifarahani@student.hpi.uni-potsdam.de

Abstract. Business process models are abstractions of concrete operational procedures that occur in the daily business of organizations. Typically one model is insufficient to describe one business process. For instance, a detailed technical model may enable automated process execution, while a more abstract model supports decision making and process monitoring by business users. Thereafter, multiple models capturing one process at various levels of abstraction often coexist. While the relations between such models are studied, little is known about the relations between process instances and abstract models.

In this paper we show how the state of an abstract activity can be calculated from the states of related, detailed process activities as they happen. The approach uses activity state propagation. With state uniqueness and state transition correctness we introduce formal properties that improve the understanding of state propagation. Algorithms to check these properties are devised. Finally, we use behavioral profiles to identify and classify behavioral inconsistencies in abstract process models that might occur, once activity state propagation is used.

1 Introduction

Recent years have seen an increasing interest in modeling business processes to better understand and improve working procedures in organizations and to provide a blue print for process implementation. With an increasing complexity of the processes and their IT implementations, technical process models become intricate. Business users can hardly grasp and analyze such exhaustive models. For instance, monitoring the state of a process instance challenges a manager, once a model enriched with technicalities is used. To support business users, less detailed models are created. As an outcome, one process is typically formalized by several models belonging to various levels of abstraction.

While methods for derivation of abstract process models from detailed ones are well understood, e.g., see [4,5,10,11,14,17], little is known about the relations between process instances and abstract process models. Meanwhile, this knowledge is essential for such tasks as monitoring of process instances by means of abstract models. Only a small share of the named approaches discusses the role of process instances [4,14]. However, even these endeavors have gaps and limitations motivating the current research. This paper assumes that each activity of

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 16–31, 2011.

an abstract process model is refined by a group of activities in a detailed model, yet each activity of the detailed model belongs to some group. Motivated by non-hierarchical activity refinement [4,12,21], we are liberal in terms of activity group definition. For instance, activities of one group can be arbitrary spread over the model. We study acyclic process models.

This paper clarifies the relations between process instances and abstract process models. To achieve this we introduce an approach to derive the state of an activity in the abstract model from the states of concrete process activities, as they happen. The approach is based on activity instance state propagation that determines the state of an abstract activity by the states of their detailed counterparts. We identify two formal properties for state propagation approaches—*state uniqueness* and *state transition correctness*. Further, we develop methods for validation of these properties. The properties should be considered during the design of any state propagation approach and can be validated by the developed algorithms. Finally, we investigate behavioral inconsistencies that might result from state propagation.

The paper is structured as follows. Section 2 motivates the work and identifies the main challenges. In Section 3 we elaborate on the state propagation, its properties and property validation methods. Further, Section 4 explains behavioral inconsistencies observable during state propagation. We position the contribution of this paper against the related work in Section 5 and conclude with Section 6.

2 Motivating Example and Research Challenges

This section provides further insights into the problem addressed by the current study. We start with a motivating example. Further, we informally outline our approach and identify the main research challenges.

Various stakeholders use models with different level of details about a given business process. In this setting several models are created for one process. Consider the example in Fig. 1. Model m describes a business process, where a forecast request is processed. Once an email with a forecast request is received, a request to collect the required data is sent. The forecast request is registered and the collected data is awaited. Then, there are two options: either to perform a full data analysis, or its quick version. The process concludes with a forecast report creation. Model m contains semantically related activities that are aggregated together into more coarse-grained ones. The groups of related activities are marked by areas with a dashed border, e.g., group g_1 includes *Receive email* and *Record request*. Model m_a is a more abstract specification of the forecast business process. Notice that further we reference the most detailed model as *initial*. Each activity group in m corresponds to a high-level activity in m_a, e.g., g_1 corresponds to *Receive forecast request*. Meanwhile, m'_a is even more abstract: its activities are refined by the activities of model m_a and are further refined by activities of m. While the forecast process can be enacted using model m, abstract models m_a and m'_a are suitable for monitoring the state of process

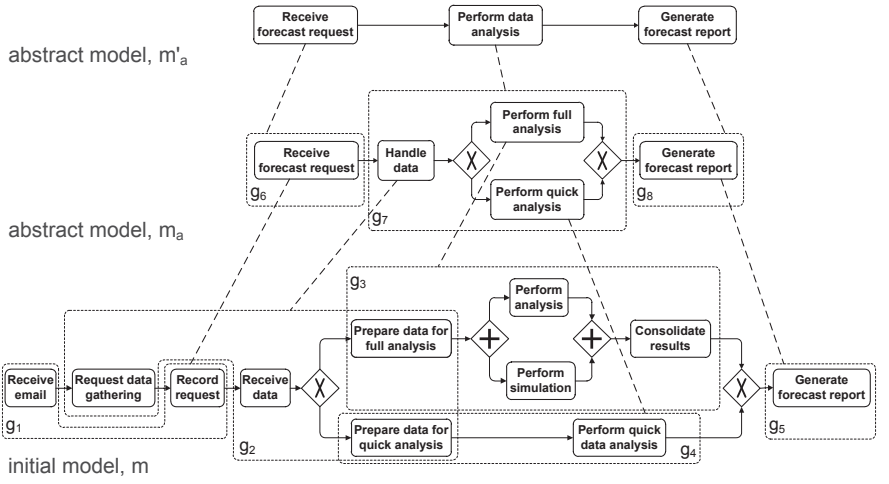


Fig. 1. Models capturing business process "Forecast request handling" at different levels of abstraction

instances. For example, a process participant might leverage model m_a, while the process owner may monitor states of instances by means of m'_a.

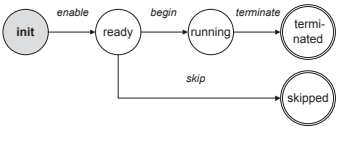


Fig. 2. Activity instance life cycle

We assume that the state of a process instance is defined by the states of its activity instances. The paper adheres to the activity instance life cycle presented in Fig. 2. When an activity is created, it is in the *init* state. We consider process models to be acyclic. Hence, once a process is instantiated, all of its activity instances are created in state *init*. The *enable* state transition brings the activity into state *ready*. If an instance is not required, *skip* transition brings it to state *skipped*. The *skipped* state has to be spread among activities that are not required. This can be realized by a well established approach of dead path elimination [13]. From the *ready* state the activity instance may evolve to *running* state by means of transition *begin*. When the instance completes its work, *terminate* transition brings it to the *terminated* state. The use of one activity instance life cycle implies that all activity instances behave according to this life cycle disregard of the abstraction level of the model an activity belongs to. Throughout this paper we frequently refer to *activity instance states*. As *activities* at the model level do not have states, we interchangeably and unambiguously use terms *activity state* and *activity instance state*.

To monitor process instance state by means of an abstract model, one needs a mechanism establishing the relation between the states of activities in the abstract model and activities of the detailed model. We reference this mechanism as activity instance state propagation. Consider a group of activities g in model m and activity x of the abstract model, such that x is refined by activities of g. State propagation maps the states of instances of activities in g to the state

of x. One can design various state propagation mechanisms depending on the use case at hand. However, we identify two formal criteria to be fulfilled by any state propagation. The first criterion, *activity instance state uniqueness*, is motivated by the observation that each activity instance at every point in time is exactly in one state. Hence, this criterion requires state propagation to be a surjective mapping: each constellation of instance states in group g must result exactly one state for x. Second criterion, *activity instance state transition correctness* requires state propagation to assure that every activity instance behaves according to the declared life cycle, neither adding, nor ignoring predefined state transitions.

In the following section we design a state propagation approach that considers the activity grouping along with the states of activity instances in the groups. This state propagation is simple and can be efficiently implemented. However, this approach does not consider control flow information. Hence, one may observe *behavioral inconsistencies* taking place in the abstract model: while the model control flow prescribes one order of activity execution, state propagation results contradicting activity instance states. Section 4 elaborates on this phenomenon.

3 Activity Instance State Propagation

This section formalizes state propagation. We start by introducing the concepts of a process model and process instance. Next, we design the state propagation method. Further, Section 3.3 proposes the algorithm validating activity instance state uniqueness, while Section 3.4 elaborates on the algorithm for activity instance state transition correctness validation. The role of the algorithms is twofold. First, they validate the developed state propagation. Second, the algorithms can be reused for validation of other state propagation methods.

3.1 Preliminaries

Definition 1 (Process Model). A tuple $m = (A, G, F, s, e, t)$ is a *process model* , where A is a finite nonempty set of activities, G is a finite set of gateways, and $N = A \cup G$ is a set of nodes with $A \cap G = \emptyset$. $F \subseteq N \times N$ is a flow relation, such that (N, F) is an acyclic connected graph. The direct predecessors and successors of a node $n \in N$ are denoted, respectively, by $\bullet n = \{n' \in N | (n', n) \in F\}$ and $n\bullet = \{n' \in N | (n, n') \in F\}$. Then, $\forall\ a \in A : |\bullet a| \leq 1 \wedge |a \bullet| \leq 1$, while $s \in A$ is the only start activity, such that $\bullet s = \emptyset \wedge \forall a \in A \backslash \{s\} : |\bullet a| > 0$ and $e \in A$ is the only end activity, such that $e\bullet = \emptyset \wedge \forall a \in A \backslash \{e\} : |a \bullet| > 0$. Finally, $t : G \rightarrow \{and, xor\}$ is a mapping that associates each gateway with a type.

The execution semantics of a process model is given by a translation to a Petri net [1,8]. As the defined process model has exactly one start activity and end activity the corresponding Petri net is a WF-net. We consider *sound* process models, see [2], that can be mapped to *free-choice* WF-nets [1].

To describe the process instance level, we formalize the activity instance life cycle shown in Fig. 2 as a tuple $(\mathcal{S}, \mathcal{T}, tran, \{init\}, \mathcal{S}')$.

$\mathcal{S} = \{init, ready, running, terminated, skipped\}$ is a set of activity instance states, where *init* is the initial state and $\mathcal{S}' = \{skipped, terminated\}$ is a set of final states. $\mathcal{T} = \{enable, begin, skip, terminate\}$ is a set of state transition labels. The state transition mapping $tran : \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{S}$, is defined as *tran(init, enable) = ready, tran(ready, begin) = running, tran(running, terminate) = terminated, tran(ready, skip) = skipped*. A *process instance* is defined as follows.

Definition 2 (Process Instance). Let $\mathcal{S}$ be the set of activity instance states. A tuple $i = (m, I, inst, stat)$ is a *process instance,* where $m = (A, G, F, s, e, t)$ is a process model, I is a set of activity instances, $inst : A \rightarrow I$ is a bijective mapping that associates an activity with an activity instance, and $stat : I \rightarrow \mathcal{S}$ is a mapping establishing the relation between an activity instance and its state.

As Definition 1 claims the process model to be acyclic, there is exactly one activity instance per process model activity, i.e., $|I| = |A|$. Finally, we formalize the activity grouping by means of function *aggregate*.

Definition 3 (Function Aggregate). Let $m = (A, G, F, s, e, t)$ be a process model and $m_a = (A_a, G_a, F_a, s_a, e_a, t_a)$—its abstract counterpart. Function $aggregate : A_a \rightarrow (\mathcal{P}(A) \backslash \emptyset)$ sets a correspondence between one activity in m_a and the set of activities in m.

Definition 4 introduces an auxiliary function st_{agg} mapping a set of activities to the set of activity instance states observed among the instances of these activities.

Definition 4 (Function State Aggregate). Let $m = (A, G, F, s, e, t)$ be a process model and $m_a = (A_a, G_a, F_a, s_a, e_a, t_a)$—the abstract model of the same process. Function $st_{agg} : A_a \rightarrow (\mathcal{P}(\mathcal{S}) \backslash \emptyset)$ is defined as $st_{agg}(x) = \bigcup_{\forall a \in aggregate(x)} \{stat(inst(a))\}$, where $x \in A_a$.

Fig. 1 illustrates function *aggregate* as follows *aggregate(Receive forecast request) = {Receive email, Record request}*. In the subsequent examples we denote the coarse-grained activities as x and y, where $x, y \in A_a$, while a and b are such activities of the model m, i.e., $a, b \in A$ that $a \in aggregate(x), b \in aggregate(y)$.

3.2 State Propagation

State propagation implies that the state of an activity x in the abstract model m_a is defined by the states of activities $aggregate(x)$ in model m. Consider the example in Fig. 3, where the instances of *Receive email* and *Record request* define the state of *Receive forecast request* instance. We develop *one possible* approach establishing the relation between activity instance states. To formalize state propagation we introduce five predicates, each corresponding to one activity instance state and "responsible" for propagation of this state to an abstract activity. An argument of a predicate is a nonempty set of states $S \subseteq \mathcal{S}$. Set S is populated by the states of activity instances observed in the activity group

$aggregate(x)$, i.e., $S = st_{agg}(x)$. If a predicate evaluates to true, it propagates the respective state to the instance of x. For example, predicate p_{ru} corresponds to the state *running*. Given an instance of *Receive forecast request* and instances of *Receive email* and *Record request*, we evaluate predicate p_{ru} against the set $\{init, terminated\}$. If p_{ru} evaluates to true, we claim the instance of *Receive forecast request* to be *running*, see Fig. 3. The predicates are defined as follows.

- $p_{in}(S) := \forall s \in S : s = init$
- $p_{re}(S) := (\exists s' \in S : s' = ready \wedge \forall s \in S : s \in \{init, ready, skipped\}) \vee (\exists s', s'' \in S : s' = init \wedge s'' = skipped \wedge \forall s \in S : s \in \{init, skipped\})$
- $p_{ru}(S) := \exists s \in S : s = running \vee (\exists s', s'' \in S : s' = terminated \wedge s'' \in \{init, ready\})$
- $p_{te}(S) := \exists s \in S : s = terminated \wedge \forall s' \in S : s' \in \{skipped, terminated\}$
- $p_{sk}(S) := \forall s \in S : s = skipped$

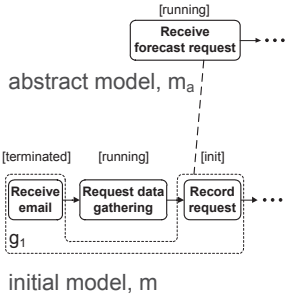


Fig. 3. State propagation example

We name this set of predicates ps. The predicate design implies that activity instance state uniqueness holds. The predicates p_{in} and p_{sk} propagate, respectively, states *init* and *skipped*, if only initialized or skipped activities are observed. The predicate p_{re} propagates state *ready* containing two conditions. The first part of its disjunction requires at least one *ready* activity, while others can be *skipped* or initialized. The second part of the disjunction propagates state *ready*, if in S exists a *skipped* activity, i.e., this activity *was* in state *ready*, and there exists an initialized activity, i.e., that activity *will be* in state *ready*. Predicate p_{ru} propagates state *running*, if 1) a *running* activity is observed or 2) in S exists a *terminated* activity, i.e., this activity *was* in state *running* and there is an initialized or *ready* activity, i.e., that activity *can be* in state *running*. The additional conditions of p_{re} and p_{ru} assure that activity instance state transition correctness holds. Finally, p_{te} propagates state *terminated*, once each activity of the group is either *skipped* or *terminated*. The five predicates construct *activity instance state propagation function* defining the state of activity x instance.

Definition 5 (Activity Instance State Propagation Function). *Activity instance state propagation function* $stp : \mathcal{P}(\mathcal{S}) \rightarrow \mathcal{S}$ maps a set of activity instance states to one activity instance state:

$$stp(S) = \begin{cases} init, & \text{if } p_{in}(S) \\ ready, & \text{if } p_{re}(S) \\ running, & \text{if } p_{ru}(S) \\ terminated, & \text{if } p_{te}(S) \\ skipped, & \text{if } p_{sk}(S). \end{cases}$$

Let $m = (A, G, F, s, e, t)$ be a process model with its process instance $i = (m, I, inst, stat)$ and $m_a = (A_a, G_a, F_a, s_a, e_a, t_a)$—the abstract model with

Algorithm 1. Verification of activity instance state uniqueness

```
1: checkStateUniqueness(Predicate[] ps, LifeCycle lifeCycle)
2: for all S ⊆ lifeCycle.S do
3:    if S ≠ ∅ then
4:       propagated = false;
5:       for all p in ps do
6:          if !propagated then
7:             if p(S) then
8:                propagated = true;
9:          else
10:            if p(S) then
11:               return false
12:       if !propagated then
13:          return false
14: return true
```

process instance $i_a = (m_a, I_a, inst_a, stat_a)$. Then, function $stat_a : I_a \rightarrow \mathcal{S}$ is defined as $stat_a(inst_a(x)) = stp(st_{agg}(x))$.

3.3 Activity Instance State Uniqueness

State propagation mechanism maps the states of activity instances of $aggregate(x)$ to the state of $inst(x)$. Definition 6 formalizes activity instance state uniqueness property.

Definition 6 (Activity Instance State Uniqueness). Let $(\mathcal{S}, \mathcal{T}, tran, \{init\}, \mathcal{S}')$ be an activity instance life cycle. Activity instance state propagation defined by function *stp* based on predicates *ps* fulfills *activity instance state uniqueness* iff $\forall S \subseteq \mathcal{S}$ exactly one predicate of *ps* evaluates to *true*.

The set of states S is observed within $aggregate(x)$. However, an activity group is defined by the user and is not known in advance. Hence, it is not efficient to reason about state uniqueness property explicitly enumerating all activity instance states that occur within activity instance groups. Instead of dealing with concrete activity instance groups, we introduce activity instance group equivalence classes. For a process instance $i = (m, I, inst, st)$ two activity instance groups $I_1, I_2 \subseteq I$ belong to one equivalence class, if in both groups the same set of activity instance states is observed, i.e., $\forall i_1 \in I_1 \exists i_2 \in I_2 : stat(i_1) = stat(i_2) \wedge \forall i_2 \in I_2 \exists i_1 \in I_1 : stat(i_2) = stat(i_1)$. For instance, a pair of activity instances with states (*init*, *terminated*) and an instance triple with states (*init*, *init*, *terminated*) belong to one class with observed states $S = \{init, terminated\}$. As this classification covers all possible state combinations, the algorithm checks all cases. We can consider such classes of activity instance groups, since the predicates make use of existential and universal quantifiers.

Algorithm 1 validates activity instance state uniqueness. The algorithm takes a set of predicates and an activity instance life cycle as inputs; it returns *true*, once the property holds. As the number of equivalence classes is exponential to the number of states in the activity instance life cycle, the computational complexity of Algorithm 1 is also exponential.

Algorithm 2. Validation of activity instance state transition correctness

```
1: checkStateTransitionCorrectness(Predicate[] ps, LifeCycle lifeCycle)
2: for all p in ps do
3:    for all S ⊆ 𝒮 : p(S) = true do
4:       for all s ∈ (S\lifeCycle.𝒮′) do
5:          for all t ∈ lifeCycle.𝒯, where tran(s,t) is defined do
6:             S′ = S ∪ {tran(s,t)}
7:             if stp(S′) ≠ stp(S) and tran(stp(S),t) ≠ stp(S′) then
8:                return false
9:             S′ = S′\{s}
10:            if stp(S′) ≠ stp(S) and tran(stp(S),t) ≠ stp(S′) then
11:               return false
12: return true
```

3.4 Activity Instance State Transition Correctness

Activity instance state propagation must assure that instances of activities in abstract models behave according to the predefined life cycle, see Definition 7.

Definition 7 (Activity Instance State Transition Correctness). Let $(\mathcal{S}, \mathcal{T}, tran, \{init\}, \mathcal{S}')$ be an activity instance life cycle. Activity instance state propagation defined by function *stp* fulfills *activity instance state transition correctness* iff $\forall S \subseteq \mathcal{S}$ each state transition allowed by *tran* from $\forall s \in S$ through $t \in \mathcal{T}$ produces a set $S' = S \cup \{tran(s,t)\}$ such that either $stp(S) = stp(S') \vee stp(S) = stp(S' \backslash \{s\})$ or $tran(stp(S),t) = stp(S') \vee tran(stp(S),t) = stp(S' \backslash \{s\})$.

Algorithm 2 validates activity instance state transition correctness. Its inputs are a set of predicates and an activity instance life cycle. It returns *true*, if state transitions are correct. The key idea of the algorithm is the observation that an instance of an activity x in the abstract model changes its state, when *one* of the activity instances that refines x changes its state. Hence, the validation considers all possible state transitions. For each predicate p of ps the algorithm constructs state sets $S \subseteq \mathcal{S}$, where the predicate evaluates to *true* (lines 2–3). For instance, predicate p_{in} has one such set $\{init\}$. Then, the validation constructs state set S' reachable from S by one state transition of the activity instance life cycle (lines 4–6 and line 9). In the example $S = \{init\}$ evolves to $\{ready\}$ or $\{init, ready\}$. For each of those reachable state sets S' function *stp* is evaluated. If for each S' the state $stp(S')$ equals $stp(S)$ or can be reached from $stp(S)$ using the same state transition as required to reach S' from S, the state propagation rules are valid. Algorithm 2 realizes the checks in lines 7 and 10 and reports correctness in line 12. The algorithm has the running time of $O(2^{|\mathcal{S}|})$.

4 Behavioral Inconsistencies

This section elaborates on the problem of behavioral inconsistencies. We start with the motivation, then introduce auxiliary formal concepts and define the notion of behavioral inconsistency. Finally, we classify behavioral inconsistencies.

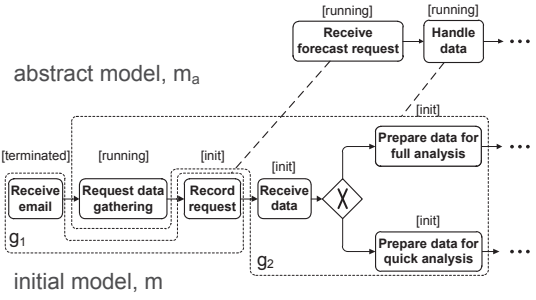


Fig. 4. Behavioral inconsistency in a process instance for the business process in Fig. 1

4.1 Example

An abstract process model dictates activity execution order. Meanwhile, the designed state propagation disregards the control flow, but influences the states of activities in m_a. In this setting one can observe *behavioral inconsistencies*. Fig. 4 exemplifies a behavioral inconsistency. Activities *Receive forecast request* and *Handle data* are refined with groups g_1 and g_2, respectively. According to the state propagation mechanism, once *Receive email* terminates, *Receive forecast request* runs until *Record request* terminates. While *Request data gathering* runs, *Handle data* is in the state *running*. According to the state propagation we observe activities *Receive forecast request* and *Handle data* in state *running* at the same time. However, model m_a prescribes sequential execution of the activities: *Handle data* can be executed, once *Receive forecast request* terminates. Hence, these states are inconsistent with the control flow of m_a.

Behavioral inconsistencies have two reasons. The first reason is activity grouping. Consider the example in Fig. 4, where activities in groups g_1 and g_2 interleave: *Receive email* precedes *Request data gathering* and *Request data gathering* precedes *Record request*. The second reason is the loss of activity *optionality* or *causality* in the abstract model. We say that an activity is optional, if there is such a process trace, where this activity is not observed. Considering the example in Fig. 4 *Prepare data for full analysis* is optional. Activity causality implies that 1) an order of execution for two activities is given and 2) two activities appear together in all process executions. One can observe causality relation for *Receive email* and *Receive data*, but not for *Receive email* and *Prepare data for full analysis*. The next section formalizes the notion of behavioral inconsistencies.

4.2 Formalization of Behavioral Inconsistencies

To formalize the discussion of behavioral inconsistencies we exploit the notion of behavioral profiles [23]. While this discussion can be based on alternative formalisms, we stick to behavioral profiles, as 1) they can be efficiently computed and 2) there are techniques enabling navigation between process models of different abstraction levels based on behavioral profiles [17]. To introduce behavioral profiles we inspect the set of all traces from s to e for a process model

$m = (A, G, F, s, e, t)$. The set of *complete process traces* $\mathcal{W}_m$ for m contains lists of the form $s \cdot A^* \cdot e$, where a list captures the activity execution order. To denote that an activity a is a part of a complete process trace we write $a \in w$ with $w \in \mathcal{W}_m$. Within this set of traces the *weak order* relation for activities is defined.

Definition 8 (Weak Order Relation). Let $m = (A, G, F, s, e, t)$ be a process model, and $\mathcal{W}_m$—its set of traces. The *weak order relation* $\succ_m \subseteq (A \times A)$ contains all pairs (x, y), where there is a trace $w = n_1, \ldots, n_l$ in $\mathcal{W}_m$ with $j \in \{1, \ldots, l-1\}$ and $j < k \leq l$ for which holds $n_j = x$ and $n_k = y$.

Two activities of a process model are in weak order, if there exists a trace in which one activity occurs after the other. Depending on how weak order relates two process model activities, we define relations forming the behavioral profile. While behavioral profiles enable judgment on activity ordering, they do not capture causality. Following on [24] we make use of auxiliary *co-occurrence relation* and *causal behavioral profile*.

Definition 9 (Behavioral Profile and Causal Behavioral Profile). Let $m = (A, G, F, s, e, t)$ be a process model and $\mathcal{T}_m$ be its set of traces. A pair $(a, b) \in (A \times A)$ is in one of the following relations: *1)* strict order relation $\rightsquigarrow_m$, if $a \succ_m b$ and $b \not\succ_m a$; *2)* exclusiveness relation $+_m$, if $a \not\succ_m b$ and $b \not\succ_m a$; *3)* interleaving order relation $||_m$, if $a \succ_m b$ and $b \succ_m a$. The set of all three relations is the *behavioral profile* of m. A pair $(a, b) \in (A \times A)$ is in the *co-occurrence relation* $\gg_m$ iff for all traces $\sigma = n_1, \ldots, n_l$ in $\mathcal{W}_m$ it holds $n_i = a, i \in \{1, \ldots, l\}$ implies that $\exists j \in \{1, \ldots, l\}$ such that $n_j = b$. Then $\{\rightsquigarrow_m, ||_m, +_m, \gg_m\}$ is the *causal behavioral profile* of m.

The behavioral profile relations along with the inverse strict order $\rightsquigarrow^{-1} = \{(x, y) \in (A \times A) \mid (y, x) \in \rightsquigarrow\}$, partition the Cartesian product of activities in one process model. The causality relation holds for $a, b \in A$ if $a \rightsquigarrow_m b \wedge a \gg_m b$.

The example in Fig. 4 witnesses that state propagation allows concurrent activity execution. However, the behavioral profile relations are defined on the trace level and do not capture concurrency. To formalize the *observed* behavior of activities, we introduce relations defined on the process instance level. These relations build on top of causal behavioral profile relations. However, they consider not traces, but process instances.

We say $(x, y) \in \rightsquigarrow_{obs}$ if there is a process instance where x is executed before y, but no instance, where y is executed before x. Similarly, relation $x \rightsquigarrow_{obs}^{-1} y$ means that there is a process instance where y is executed before x, but no instance, where x is executed before y. Relation $x +_{obs} y$ holds if there is no instance where x and y both take place. Relation $||_{obs}$ corresponds to the existence of 1) an instance where x is executed before y, 2) an instance where y is executed before x and 3) an instance where x and y are executed concurrently. Finally, $x \gg_{obs} y$ holds if for every instance, where x is executed, y is executed as well. Then, the behavioral inconsistency can be defined as follows.

Definition 10 (Behavioral Inconsistency). Let $m = (A, G, F, s, e, t)$ be a process model and $i = (m, I, inst, stat)$—its instance. $m_a = (A_a, G_a, F_a, s_a, e_a, t_a)$ is the abstract model obtained from m and having the instance $i_a = (m_a, I_a, inst_a, stat_a)$, where function $stat_a$ is defined as $stat_a(inst_a(x)) = stp(st_{agg}(x))$. We say that there is a *behavioral inconsistency,* if $\exists(x, y) \in (A_a \times A_a)$ for which the causal behavioral profile relations do not coincide with the observed behavioral relations: *1)* $(x, y) \in \rightsquigarrow_{m_a} \wedge (x, y) \notin \rightsquigarrow_{obs}$; or *2)* $(x, y) \in \rightsquigarrow^{-1}_{m_a} \wedge (x, y) \notin \rightsquigarrow^{-1}_{obs}$; or *3)* $(x, y) \in +_{m_a} \wedge (x, y) \notin +_{obs}$; or *4)* $(x, y) \in ||_{m_a} \wedge (x, y) \notin ||_{obs}$; or *5)* $(x, y) \in \gg_{m_a} \wedge (x, y) \notin \gg_{obs}$.

4.3 Classification of Behavioral Inconsistencies

Table 1 classifies behavioral inconsistencies comparing the declared and observed behavioral constraints for abstract process model activities x and y. A table row corresponds to behavioral profile relations declared by an abstract model. Columns capture the observed behavioral relations. A cell of Table 1 describes an inconsistency between the observed and declared behavioral relations. The table presents a complete analysis of inconsistencies, due to extensive exploration of all possible relation combinations.

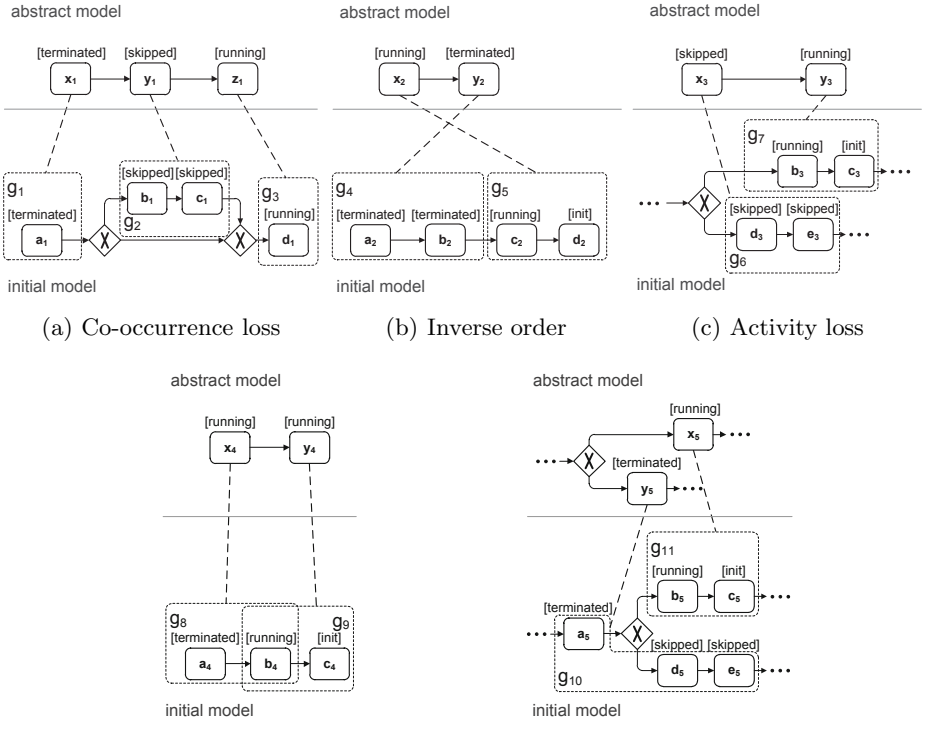
The "+" sign witnesses no inconsistency since the declared and observed constraints coincide. We identify one class of activity groups causing no inconsistency. Consider a pair of activities $x, y \in A_a$. If $\forall(a, b) \in aggregate(x) \times aggregate(y)$ the same causal behavioral profile relation holds, no behavioral inconsistency is observed. A prominent example of activity groups that fulfill the defined requirement are groups resulting from the canonical decomposition of a process model into single entry single exit fragments, see [19,20].

Every cell marked with "±" symbol corresponds to an inconsistency, where no contradiction takes place: an observed relation restricts a declared behavioral relation. Consider, for instance, the behavioral inconsistency, where $x||_{m_a}y$, while $x \rightsquigarrow^{-1}_{obs} y$ and $x \gg_{obs} y$. This inconsistency has no contradiction, as the observed behavior only restricts the declared one. We identify five classes of behavioral inconsistencies marked in Table 1 and illustrate them by the examples in Fig. 5.

`A: Co-occurrence loss` Behavioral inconsistencies of this type take place if the model declares co-occurrence for an activity pair, while both activities are observed not in every process instance. The cause of inconsistency is the loss of

Table 1. Classification of behavioral inconsistencies

		$x \rightsquigarrow_{obs} y$		$x \rightsquigarrow^{-1}_{obs} y$		$x +_{obs} y$	$x\|\|_{obs}y$
		$x \gg_{obs} y$	$x \not\gg_{obs} y$	$x \gg_{obs} y$	$x \not\gg_{obs} y$		
$x \rightsquigarrow_{m_a} y$	$x \gg_{m_a} y$	+	A	B	B	C	D
	$x \not\gg_{m_a} y$	±	+	B	B	C	D
$x \rightsquigarrow^{-1}_{m_a} y$	$x \gg_{m_a} y$	B	B	+	A	C	D
	$x \not\gg_{m_a} y$	B	B	±	+	C	D
$x +_{m_a} y$		E	E	E	E	+	E
$x\|\|_{m_a}y$		±	±	±	±	C	+



(a) Co-occurrence loss (b) Inverse order (c) Activity loss

(d) Order loss (e) Exclusiveness loss

Fig. 5. Examples of behavioral inconsistencies: one example per class

information about the causal coupling of an activity pair. The example in Fig. 5(a) illustrates this inconsistency type. Since activities of group g_2 are skipped, activity y_1 is in state *skipped* as well. However, it cannot be skipped according to the abstract model control flow.

`B: Inverse order.` We say that an *inverse order* inconsistency takes place if the model prescribes $x \rightsquigarrow_{m_a} y$ ($x \rightsquigarrow_{m_a}^{-1} y$), whilst the user observes $x \rightsquigarrow_{obs}^{-1} y$ ($x \rightsquigarrow_{obs} y$). Fig. 5(b) gives an example of such an inconsistency baring its cause: for each $(a, b) \in aggregate(x) \times aggregate(y)$ activities a and b have the order opposite to the order of x and y.

`C: Activity loss.` Once the process model specifies two activities to appear within one instance, whereas only one activity is observed within an instance, *activity loss* inconsistency takes place. Fig. 5(c) exhibits one example of such an inconsistency. While activity groups g_6 and g_7 are exclusive, the corresponding abstract activities x_3 and y_3 are in strict order. Accordingly, either x_3 or y_3 is observed within each instance.

`D: Order loss.` For a pair of activities in (inverse) strict order, the user observes interleaving execution. A behavioral inconsistency of this type is exemplified

in Fig. 5(d). Such inconsistencies have the following roots: 1) $aggregate(x) \cap aggregate(y) \neq \emptyset$ or 2) exist $a_1, a_2 \in aggregate(x)$ and $b_1, b_2 \in aggregate(y)$ such that it holds $a_1 \succ_m b_1$ and $b_2 \succ_m a_2$. In Fig. 5(d) activity b_2 belongs to groups g_1 and g_2. As a consequence, once b_2 runs both sequential activities x_2 and y_2 are running concurrently.

`E: Exclusiveness loss.` While the model prescribes exclusiveness relation for x and y, both activities are observed within one instance. These inconsistencies take place, once in the initial model there exist such a and b that $a \succ_m b$ or $b \succ_m a$. Fig. 5(e) exemplifies this inconsistency. According to the abstract model $x_3 +_{m_a} y_3$. However, in the presented process instance both x_3 and y_3 take place.

5 Related Work

We identify two directions of the related work. The first one is the research on business process model abstraction. The second one is the body of knowledge discussing similarity of process models.

Business process model abstraction has been approached by several authors. The majority of the solutions consider various aspects of model transformation. For instance, [5,10,11,15,17] focus on the structural aspects of transformation. Among these papers [17] enables the most flexible activity grouping. Several papers study how the groups of semantically related activities can be discovered [6,16]. A few works elaborate on the relation between process instances and abstract process models, e.g. [4,14]. In [4] Bobrik, Reichert, and Bauer discuss state propagation and associated behavioral inconsistencies, but do not use the concept of activity instance life cycle. [14] suggests state propagation approach that builds on the activity instance life cycle shown in Fig. 6. In [14] Liu and Shen order three states according to how "active" they are: *not_started* < *suspended* < *running*. The state propagation rules make use of this order, e.g., if a coarse-grained activity is refined by activities in one of the *open* states, the high-level activity is in the most "active" state. Against this background, consider an example activity pair evolving as follows: (*not_started*, *not_started*) to (*not_started*, *running*) to (*not_started*, *completed*). According to the rules defined in [14] the high level activity evolves as *not_started* to *running* to *not_started*, which contradicts the activity instance life cycle. As we mentioned above, the majority of works on business process model abstraction consider only the *model* level. Meanwhile, the papers that take into account process instances have gaps and limitations. For instance, [4,14] motivated us not only to introduce the state propagation approach, but also to identify formal properties for such approaches and develop validation algorithms.

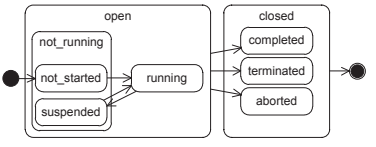


Fig. 6. Activity instance life cycle as presented in [14]

The works on similarity of process models can be refined into two substreams. A series of papers approaches process model similarity analyzing model structure and labeling information, see [7,21]. These works provide methods to discover

matching model elements. Several research endeavors analyze behavioral similarity of process models. In particular, [3] introduces several notions of inheritance and operations on process models preserving the inheritance property. Recently, Weidlich, Dijkman, and Weske investigated behavioral compatibility of models capturing one business process [22]. [9] elaborates on process model similarity considering both model element labeling and model behavior. Considering that processes are inherently concurrent systems, various notions of behavioral equivalence for concurrent systems can be leveraged to compare the behavior of initial and abstract process models [18]. The enumerated papers help to compare the behavior of initial and abstract process models. As such, the notions of behavioral equivalence and behavioral compatibility might give additional insights into the causes of behavioral inconsistencies, see Section 4, and classify them further.

6 Conclusion and Future Work

Although the relations between models capturing one business process on different levels of abstraction have been thoroughly studied earlier, the relations between process instances and abstract process models have been barely explored. The current paper bridged this gap. First, we developed activity instance state propagation mechanism that allows to describe the process instance state by means of an abstract process model. Second, we have identified two formal properties for state propagation and proposed methods for their validation. Finally, we elaborated on behavioral inconsistencies that can be observed, once the assumed abstraction and state propagation mechanisms are used.

We foresee several directions of the future work. The direct next step is the extension of the considered model class. As we leverage dead path elimination to spread activity instance state *skipped* over not executed activities, the state propagation approach is limited to acyclic models. Substitution of dead path elimination with an alternative approach would facilitate handling of cyclic models. Another direction is the further study of the behavioral inconsistencies and methods for their resolution. With that respect, it is valuable to integrate control flow information into state propagation mechanism. Finally, the applications of the introduced technique call for deep investigation. One direct application of our approach is business process monitoring [25], where abstract models help users to follow the progress of running business processes.

References

1. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
2. van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) BPM. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
3. van der Aalst, W.M.P., Basten, T.: Life-Cycle Inheritance: A Petri-Net-Based Approach. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 62–81. Springer, Heidelberg (1997)

4. Bobrik, R., Reichert, M., Bauer, T.: Parameterizable Views for Process Visualization. Technical Report TR-CTIT-07-37, Centre for Telematics and Information Technology, University of Twente, Enschede (April 2007)
5. Bobrik, R., Reichert, M., Bauer, T.: View-Based Process Visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
6. Di Francescomarino, C., Marchetto, A., Tonella, P.: Cluster-based Modularization of Processes Recovered from Web Applications. Journal of Software Maintenance and Evolution: Research and Practice (2010)
7. Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
8. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)
9. van Dongen, B., Dijkman, R., Mendling, J.: Measuring Similarity between Business Process Models. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008)
10. Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering 64(2), 419–438 (2008)
11. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining–Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
12. Knoepfel, A., Groene, B., Tabeling, P.: Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley & Sons, Ltd. (2005)
13. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR, Upper Saddle River (2000)
14. Liu, D.-R., Shen, M.: Business-to-business Workflow Interoperation based on Process-Views. Decision Support Systems 38, 399–419 (2004)
15. Polyvyanyy, A., Smirnov, S., Weske, M.: The Triconnected Abstraction of Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 229–244. Springer, Heidelberg (2009)
16. Smirnov, S., Dijkman, R.M., Mendling, J., Weske, M.: Meronymy-Based Aggregation of Activities in Business Process Models. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 1–14. Springer, Heidelberg (2010)
17. Smirnov, S., Weidlich, M., Mendling, J.: Business Process Model Abstraction Based on Behavioral Profiles. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 1–16. Springer, Heidelberg (2010)
18. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990)
19. 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)
20. 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)

21. Weidlich, M., Dijkman, R., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010)
22. Weidlich, M., Dijkman, R., Weske, M.: Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 78–94. Springer, Heidelberg (2010)
23. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement based on Behavioural Profiles of Process Models. In: IEEE TSE (2010) (to appear)
24. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient Computation of Causal Behavioural Profiles Using Structural Decomposition. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 63–83. Springer, Heidelberg (2010)
25. zur Muehlen, M.: Workflow-based Process Controlling - Foundation, Design and Application of Workflow-Driven Process Information Systems. PhD thesis, University of Münster (2002)

Push-Enabling RESTful Business Processes

Cesare Pautasso[1] and Erik Wilde[2]

[1] Faculty of Informatics, University of Lugano, Switzerland
[2] School of Information, UC Berkeley, USA

Abstract. *Representational State Transfer (REST)* as an architectural style for service design has seen substantial uptake in the past years. However, some areas such as *Business Process Modeling (BPM)* and push services so far have not been addressed in the context of REST principles. In this work, we look at how both BPM and push can be combined so that business processes can be modeled and observed in a RESTful way. Based on this approach, clients can subscribe to be notified when certain states in a business process are reached. Our goal is to design an architecture that brings REST's claims of loose coupling and good scalability to the area of BPM, and still allow process-driven composition and interaction between resources to be modeled.

1 Introduction

Whereas business processes provide a highly suitable abstraction [19] to model the state of the resources published by RESTful Web Services [25], the problem of dealing with the need of notifying clients that a process has completed remains open. More generally speaking, resources backed by a business process can evolve their state independently and as a consequence, a mechanism is required for clients to keep track of state changes triggered by intermediate progress reports or by the completion of the execution of the business process.

One of the main principles of the architectural style of *Representational State Transfer* (REST [8]) is that it is a client/server architectural style, where clients initiate interactions and then interact with resources which are made accessible by servers. The main principles of REST prescribe that resources should be identified in a unified way, that interactions with these resources should be done through a uniform interface, and that interactions are stateless and make use of self-describing messages. Since clients are always the ones initiating interactions (by following links to resources they discovered in previous interactions), resource state changes remain invisible to clients until they interact with the updated resources. Increasingly, use cases emerge (e.g., [11,3,4]) where a reversal of this interaction pattern is desirable, so that clients can be made aware of the changes in resource state as they occur. Traditional Web-style interactions are often described as *pull* (the clients pulling resource state from the server), and thus the complementary functionality is often described as *push*, where resource state changes are pushed to a client.

In this paper, we take a look at the problem of how to combine *Business Process Modeling (BPM)* with REST. The goal is to design an architecture for

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 32–46, 2011.

RESTful business process management featuring push notification of task and process instance state changes. To do so, we propose to use the Atom feed format as a standard representation of process instance execution logs so that clients can subscribe to process instances and tasks published as resources and be notified when these complete their execution. To ensure a timely propagation of notifications, we discuss different alternative optimizations such as the emerging PubSubHubbub (PuSH [9]) and WebSocket [14] protocols.

The rest of this paper is organized as follows. In Section 2 we describe how processes can be published as resources. We use an example (Section 3) to motivate the need for supporting push notification of process state changes. In the following Section 4, we discuss and compare several approaches to achieve push notifications in the context of the REST architectural style. Section 5 describes the architecture of a push-enabled process execution engine. Related Work is presented in Section 6 before we conclude in Section 7.

2 RESTful Business Processes

For the purposes of this paper, we minimize the assumptions on the meta-model used to capture an executable representation of a business process. Our approach is compatible with most existing standards (e.g., BPEL [15], BPMN [18,23]) but does not require full compliance with the actual notations. In this section we describe how the REST constraints of resource addressability, multiple negotiable representations of resources and resource access through a uniform interface can be mapped to the domain of business process management.

2.1 Publishing Processes as Resources

We assume that a process can be broken down as a set of discrete tasks, which are linked by some kind of structured or unstructured set of dependencies (e.g., control and data flow) [6]. Both processes and tasks can be published as resources, and thus should be addressable with a URI. At runtime, processes can be instantiated for execution and typically multiple process instances of the same process are executed at the same time. Each process instance is also a resource and should be identified by its own URI. The state of the execution of a process instance can be defined as the union of the state of the execution of its tasks, which can also be seen as sub-resources of the process instance resource.

To identify the resources, we propose the following URI templates [10]:

```
/{process}
/{process}/{instance}
/{process}/{instance}/{task}
```

The structure of the URI templates helps to highlight the containment and instance-of relationships between processes, their instances and the tasks belonging to them. In particular, we are interested in identifying task as resources only in the context of specific process instances, since we are interested in interacting with tasks at run time, during the execution of the corresponding process

instance, and we are only interested in manipulating design-time process artifacts (i.e., process template descriptions) as a whole.

Following the hypermedia constraint, a client may make use of these relationships to discover the available processes, and retrieve links to the corresponding instances. Each resource instance can be manipulated as a whole, or it can provide links back to clients so that they can interact with its tasks.

2.2 Process Representations

Fetching each resource returns a snapshot of its current state represented using different media types. Different clients may be interested on a different projection over the state of a process resource and thus may use standard content-type negotiation techniques to retrieve a suitable view in the most convenient format. Table 1 lists some examples of possible media types and gives a short description of the information that should be part of the corresponding representation.

Table 1. Process resources can be represented using different media types

Content-Type	Description
Resource: `/{process}`	
`text/plain`	Basic textual description of the process
`text/html`	Web page with a form to start a new process instance and links to the currently running instances
`application/bpmn+xml`	BPMN XML serialization of the process
`application/svg+xml`	Graphical rendering of the process in Scalable Vector Graphics
`application/json`	Process meta-data (e.g., name, author, version, creation date) sent using the JavaScript object notation
`application/atom+xml`	An Atom feed listing all instances of a process as entries
Resource: `/{process}/{instance}`	
`text/html`	Web page with a summary of the state of the execution of the instance with links to its process and to the set of active tasks
`application/svg+xml`	Graphical rendering of the process instance (e.g., with colored task boxes marking their current execution state)
`application/json`	Process instance data: process name, responsible user, start/end timestamp, input/output data parameter values
`application/atom+xml`	An Atom feed listing all tasks of a process instance as entries
Resource: `/{process}/{instance}/{task}`	
`text/plain`	Basic textual description of the task goal, involved roles and required/produced data
`text/html`	Web page with a form to interact with the task (if it is active) or with a summary of the outcome of its execution (if the task is complete)
`application/json`	Task instance data: process name, task name, responsible user/role, start/end timestamp, input/output data parameters
`application/atom+xml`	An Atom feed listing all state transitions of a task as entries

2.3 Uniform Interface

The manipulation of process resources happens through their uniform interface, which consists of the `GET`, `PUT`, `DELETE`, and `POST` methods for resources accessed using the HTTP protocol. Table 2 outlines the semantics of all methods applied to each resource.

Clients interact with the processes deployed for execution. For example, `PUT /{process}` deploys a process model and publishes it under a given identifier. Conversely, `DELETE /{process}` will undeploy a process and render its identifier void. This will also cause all instances of the process to be removed. Thus, the method should only be allowed when there are no active instances of the particular process left in the system.

The `/{process}` resource also acts as a "resource factory" [1] as it allows clients to initiate the execution of new process instances by sending POST requests to it. The request can be serviced in two ways: blocking or non-blocking. In the first case, the client will wait until the execution of the entire process has completed and will receive a response which contains the output (or the reply) of the process. For long-running processes, it may be more convenient to respond immediately with an identifier of the newly started process instance (e.g., (`/{process}/{instance}`)).

Whereas the first (blocking) approach can be seen as a convenient way for letting clients perform an RPC-based invocation of processes, we argue that only the second non-blocking solution correctly publishes the process instance as a resource, by providing clients with a link to its URI (which may be shared among multiple clients that are interested to interact with the process instance and its tasks). The clients may then use such identifier to retrieve the result of the process once it completes its execution, or the clients may be informed of the outcome of the process execution with one of the notification mechanisms that we will discuss in the following Sections.

A client can obtain a global view over a running process instance by GETting its representation, which – depending on the chosen media type – it may contain links to the individual tasks. A client may be interested in only listing all active tasks of a process instance, as opposed to retrieving links to all tasks and then having to poll each task to determine its state. Likewise, a client may be interested in getting notified when the set of active tasks changes, as it may be waiting for one particular task to become ready for execution.

Once a task URI has been retrieved, a client may perform a GET request on it to read task-specific information (e.g., its state, its input/output parameter values) or a PUT request to change the state of a task and set the value of its output parameters. Clients are not allowed either to POST or DELETE individual task resources. Once all tasks have completed their execution, their final state remains associated with the corresponding resources until a `DELETE /{process}/{instance}` request is performed. Only then, all information associated with all tasks of a process instance is removed.

Table 2. Uniform interface semantics for process resources

Method	Description
Resource: `/{process}`	
`GET`	Retrieve a representation of a process, with links to its instances
`PUT`	Deploy a process (or update an already deployed process)
`DELETE`	Undeploy a process
`POST`	Instantiate a new process instance (blocking/non-blocking)
Resource: `/{process}/{instance}`	
`GET`	Retrieve a representation of the current state of a process instance, with links to its tasks
`PUT`	Not Allowed
`DELETE`	Remove the instance (once all tasks have completed)
`POST`	Not Allowed
Resource: `/{process}/{instance}/{task}`	
`GET`	Retrieve the current state of the task instance
`PUT`	Modify the state of the task instance (e.g., mark it as completed)
`DELETE`	Not Allowed
`POST`	Not Allowed

3 Example

We use the classical loan approval process to illustrate how a business process model can be REST-ified and to motivate the need for supporting push notification of its state changes. The process is identified by the `/loan` URI, and two of its tasks (called `choose` and `approve`, marked in black in Figure 1) are published as resources. The other tasks are not visible from clients but carry out important back-end activities, such as checking the validity of incoming loan applications, contacting different banks for the latest rates as well as confirming the loan, if an offer has been chosen by the customer and approved by management.

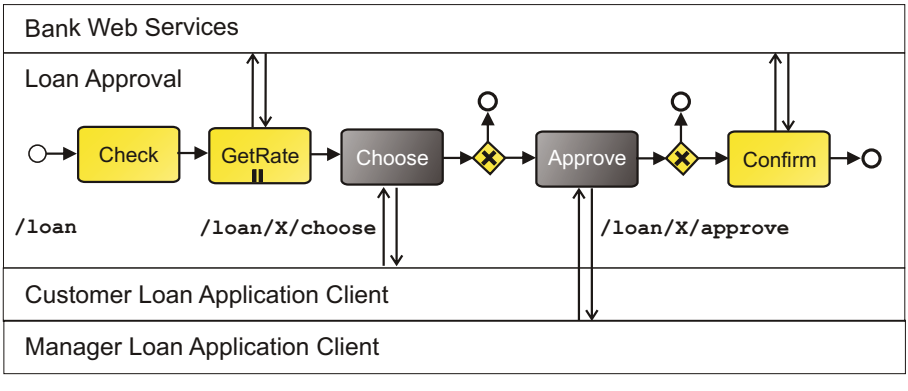


Fig. 1. Example: Publishing the Loan Approval Business Processes as a Resource

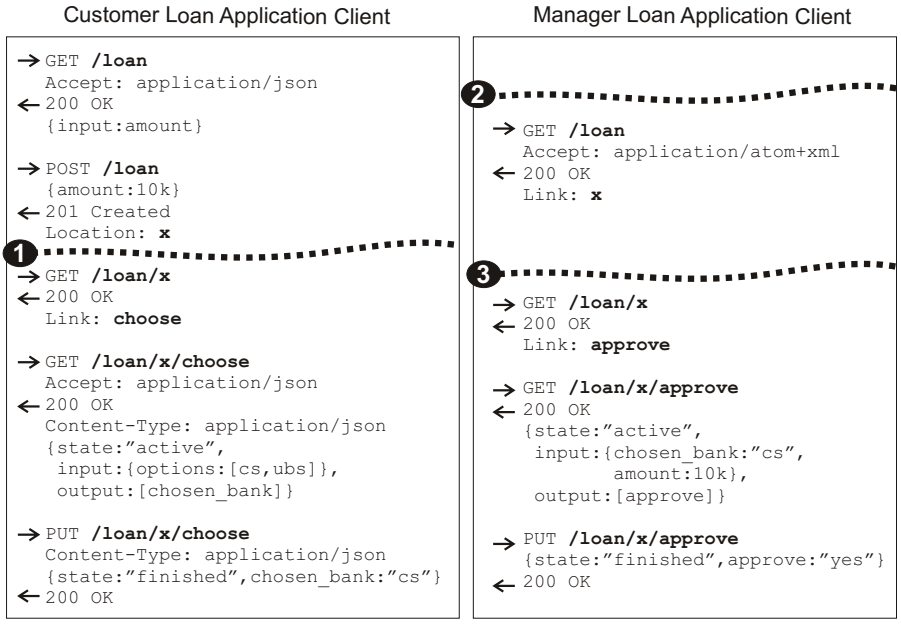


Fig. 2. Example: Client interactions with the `choose` (left) and `approve` (right) task resources

Figure 2 presents in more detail the interaction of two key clients of the process: the customer applying for a loan (left) and the manager in charge of tracking the progress of all loan applications (right).

The customer application client retrieves the form describing the information required to start the process (e.g., the `amount`) with a `GET /loan` request. This is followed by a `POST /loan` request which transfers the filled-out form and uses the information to start running a new process instance, whose URI (`x`) is returned to the client as a hyperlink. The process starts running its tasks, which are validating the client request and gathering the necessary information for making an offer to the client. This may be a time-consuming process, during this time the client may check its progress at anytime by polling the state of the newly started process instance with a `GET /loan/x` request. Eventually the `choice` task becomes ready for execution and the client will be directed to it with a link (event 1). The client then retrieves the state of the task, which amounts to a set of possible options and a form used to express the client choice. The form can be submitted with a `POST /loan/x/choose` request, which will also complete the corresponding task.

The manager application client monitors all ongoing loan applications in the system by periodically issuing `GET /loan` requests. These return a set of URIs to the corresponding process instances (which could be serialized as a feed). In our particular example, the manager follows the link to instance `/loan/x` and watches it until it reaches the task for which his approval is required.

To inspect the state of the application he first retrieves the task instance with `GET /loan/x/approve`, then he submits his approval with the corresponding `PUT /loan/x/approve` request, which will complete the execution of the approve task and trigger the confirmation of the loan with the chosen bank in the backend.

There are three points during these interactions where the clients are waiting for the process to reach a certain state. The customer wants to retrieve an offer and make his choice (event 1 in Figure 2), the manager needs to see which application was started (event 2) and which offers have been accepted by customers and require his approval (event 3). In the following Section we will discuss which are the options to avoid having the client application periodically poll the process resources in order to detect that the corresponding event has occurred during the execution of the process.

4 RESTful Push Interactions

In the context of the client/server style used by REST, push interactions reverse the roles between clients and servers. Thus, clients need to be identifiable, reachable and accessible, so that servers can push state changes of their resources back to them. In the context of the push micro-interaction, clients effectively become servers, and the servers become clients (they initiate interactions once a resource has changed state, and notify those listeners who are interested in these state changes). This reversal in roles can be confusing, and thus we keep the original client/server macro-roles as the ones signifying that the servers are the ones providing access to the resources and managing the identification space of resources, whereas clients are the ones interested in receiving notifications about resource state changes in order to build applications based on these resources.

One excellent example illustrating this micro vs. macro view is a prototype of a push-oriented protocol that has been developed by Google under the name of *PubSubHubbub (PuSH)* [9]. PuSH is based on Atom feeds and allows clients to be notified of feed updates immediately. Traditionally, feeds are a RESTful pull-oriented architecture, and clients need to repeatedly pull feeds to become aware of updates. PuSH introduces an intermediate layer of reverse interactions where clients subscribe to a feed by registering a callback at a so-called *hub*. The hub then notifies all registered clients of feed updates by initiating HTTP interactions with the callbacks once a feed has been updated. While the notification path reverses the traditional interaction pattern (the client must have a HTTP server running at the callback URI, and the hub then sends the update in an HTTP request), the higher-level interaction still is client/server, because the identified resource is the feed or the updated entry, and the underlying reversal of the interaction is simply an optimization that allows clients to be notified faster than they could be by implementing an interval-based polling scheme.

It is interesting to note that in this scenario, the overall REST architecture is not changed in any way. On the macro level, the main resources are feeds and feed entries and the overall goal is to make sure that clients are aware of resource state changes. On the micro level, a new set of identifiers has been

introduced which are the callback URIs that have to be managed by the hubs, and on that micro level, the interaction pattern is reversed. But this is merely an optimization to allow clients to be more efficiently notified, and essentially, this shifts the burden of fast updates from clients (which would have to do frequent polling otherwise) to hubs (which now have to manage potentially large sets of callback identifiers). In both scenarios, it is possible to use intermediaries, with the difference that in the pull scenario they are independent of the specific scenario and simply work because REST and specifically HTTP are designed to take advantage of intermediaries improving the effectiveness of HTTP traffic. In the push scenario, PuSH allows hubs to be chained and thus can also provide some support for scalability, but since interactions are reversed, network traffic cannot be as effectively handled in intermediaries close to the managed resource.

So far, Web architecture itself has not been updated in a way that provides general support for push-oriented communication patterns. In some scenarios, HTTP has been stretched to support push communications, and those scenarios are described in Section 4.1, along with the ongoing efforts of HTML5 to improve browsers as a general-purpose application platform. Apart from Web architecture itself, there also are numerous dedicated push frameworks available on a variety of platforms, and those are described briefly in Section 4.2. The main purpose of these sections is to describe the status quo, and to make the point that for a perfectly RESTful way of push-enabling business processes, there currently is no good alternative provided by Web architecture itself, and thus in the short term it is necessary to adopt some intermediary solution, whereas the long term goal should be to enhance Web architecture in a way that push-oriented interactions are supported as well.

4.1 Using HTTP

On the current Web, the only widely supported protocol is HTTP, and the roles of servers and clients are fixed, servers are managing resources, and clients are initiating interactions by sending HTTP requests. This pattern serves most applications well, because it is very good for implementing scalable pull. However, for applications requiring more immediate notifications of server-side events, some workarounds have become popular that are often summarized under the label of *HTTP long polling* [17]. The idea of this approach is that the client sends a request, but that the request remains pending for a long time, and that the server only sends a response when an event has occurred. This pattern thus emulates push by initiating a pull interaction, and then using the response to push the notification back.

There are several disadvantages of this approach. Some are caused by the fact that long-lived HTTP interactions are not very robust (depending on the underlying network infrastructure) because they are using long-lived TCP connections, and those are sometimes terminated by network components for management or security reasons. Another disadvantage is that this requires the server to manage a large number of open connections, thus breaking the REST statelessness constraint. For a server to be able to support many open connections, it often

takes considerable low-level tweaking of the system to make sure that the server will not run out of system resources. The problems with long polling become particularly pronounced when the events occur infrequently (such as in the business process management domain) and thus the connections are kept open for an extended period of time without any actual communication over them. It is clear that long polling is not an elegant or well-designed solution to the goal of implementing push, but under certain circumstances it works well enough and is actually used in many Web applications.

As the most recent major development in the area of Web technologies, *HTML5* [12] does add some aspects that are somehow related to push services. *Server-Sent Events* [13] add an API that allows a client to expose server-sent events as events in the popular DOM API model. However, the specification only defines the API that should be exposed via DOM, and makes no assumptions about how the events have been transported to the client. Thus, while this specification provides an interesting API to expose push notifications on the client, it does not help with the actual delivery of them across the network. The second push-related HTML5 technology are *Web Sockets* [14], which defines a TCP-like bidirectional connection between a client and a server. While this mechanism can be used for push notifications, it is not limited to them or optimized for them, and mostly can be seen as an attempt to replace HTTP long polling. When used for push, Web sockets have the same limitations as HTTP long polling, meaning that they are resource-intensive on the server side, and as ineffective for infrequent notifications. Furthermore, they may not work reliably in resource-constrained environments, in particular in mobile client settings.

4.2 Dedicated Push Frameworks

The absence of well-designed methods for Web applications to implement push patterns has not gone unnoticed. One popular example of an attempt to solve the problem is *PubSubHubbub (PuSH)*, a protocol that allows the interaction patterns of feeds to be reversed. Instead of polling for updates, clients subscribe to a hub, and feed updates are pushed to them. This is done by the hubs sending HTTP requests to the registered callback URI of the client, which means that the client must be able to run an HTTP server, and that the underlying network allows this reverse connection initiation to happen. PuSH standardizes the data delivery, but does not provide a model for managing subscriptions. Approaches such as *Feed Subscription Management (FSM)* [27] could be used to address this limitation.

There also are specific push-oriented protocols such as the *Extensible Messaging and Presence Protocol (XMPP)* [26], which has been developed mostly in an attempt to have a standardized protocol for *Instant Messaging (IM)*. While IM protocols do cover the use case of delivering notifications to clients in a general and scalable way, they often have the disadvantage of adding a lot of additional functionality specific for the IM use case that is not required for simple push notifications [24]. Furthermore, establishing a new protocol as part of Web architecture that is exclusively used for implementing push notifications seems to

be a decision that might not be as reasonable as reusing the request/response capabilities of the universally supported HTTP and repurposing them.

While we do not have the space to explore platform-specific push protocols and services such as *Apple Push Notifications (APN)* or Android's *Cloud to Device Messaging (C2DM)*, it is interesting to note that all relevant mobile platforms have developed their own flavor of push notifications, and that these have sometimes very different properties. C2DM for example is a very simple service with a best effort implementation and no delivery confirmation notification. BlackBerry's push service, on the other hand, has different Quality-of-Service classes, and in the highest class, delivery notifications are supported. This is of course considerably more expensive in terms of implementing such a service, but on the other hand may be a requirement that some applications have and either can get as part of a platform service, or they have to layer it on top of a more simple service by adding a more sophisticated "transport layer" themselves.

In summary, even if some attempts (e.g., the Juggernaut Ruby on Rails plugin is a notable example) have been made to provide some degree of transparency, the current landscape of push support for Web applications is incomplete and fragmented. Developers either have to use the suboptimal method of HTTP long polling, or, when communicating with mobile clients, need to work with possibly a variety of platform-specific notification services that require a lot of integration work if multiple platforms need to be supported. We thus believe that the Web should provide a method for push notifications, so that Web-oriented applications have a single and reliable way of implementing push interaction patterns across the widest possible selection of resources and consuming clients.

5 Architecture

Our solution for push-enabling RESTful business process management systems is based on representing process resources as feeds and then using one of the previously discussed mechanisms to notify clients of changes to the process feed.

5.1 Representing Process Resources as Feeds

We propose to use the Atom standard media type as a suitable XML-based representation of process instances, since – for the purposes of monitoring and tracking their state – they can be seen as a collection of tasks.

As shown in Figure 3, the standard feed meta-data (such as the title, author, id, and updated timestamp) can be easily mapped to the meta-data used to describe process instances. For example, the updated timestamp can be computed based on the time associated with the latest event (i.e., state change) of a process instance. Links can be provided back to the process instance itself (with the self relation) and also to the process from which the instance was instantiated (with the template relation). Each feed entry represents a (public) task of the process instance. Again, the updated timestamp marks the time of the most recent change of the state of the task (which for tasks that are still waiting to

become active it is equivalent to the instantiation time of the process). Links to the individual task resources can be added so that a more detailed view of the task can be obtained in addition to what is found in the summary text, which could be used to store a textual description of the current state of the task.

A similar mapping can be provided for the whole collection of process instances for a given template, which can be also represented as a feed. We also suggest to use the feed format to represent the history of a task execution, so that it is possible to use a standard representation to log and monitor all state changes of a task instance.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
        <title>Loan Approval Process</title>
        <subtitle>Instance x</subtitle>
        <link href="http://rest.jopera.org/loan/x" rel="self" />
        <link href="http://rest.jopera.org/loan" rel="template" />
        <link href="http://pubsubhubbub.appspot.com/" rel="hub" />
        <id>http://rest.jopera.org/loan/x</id>
        <updated>2011-06-10T11:11:30Z</updated>
        <author><name>Cesare Pautasso</name><email>cp@jopera.org</email></author>
        <entry>
                <title>Choose Task (Ready)</title>
                <link href="http://rest.jopera.org/loan/x/choose" />
                <id>http://rest.jopera.org/loan/x/choose</id>
                <updated>2011-06-10T11:12:20Z</updated>
                <summary>State: ready</summary>
        </entry>
        <entry>
                <title>Approve Task (Waiting)</title>
                <link href="http://rest.jopera.org/loan/x/approve" />
                <id>http://rest.jopera.org/loan/x/approve</id>
                <updated>2011-06-10T11:11:30Z</updated>
                <summary>State: waiting</summary>
        </entry>
</feed>
```

Fig. 3. Atom Feed corresponding to a Process Instance of the Running Example

5.2 Push-Enabled RESTful Process Execution Engine

As part of the JOpera [21] project, we have built a RESTful business process execution engine. In this section we focus on how the architecture of the system has been extended to support push notifications.

The layered architecture shown in Figure 4 augments the process engine kernel (which executed the process template code and manages the state of the corresponding process instances) with a REST layer, whose purpose is to publish processes as resources. To do so, it uses a Web server (such as Jetty) that supports both the HTTP and WebSockets protocols to handle synchronous client requests and asynchronous notifications.

For the first kind of synchronous requests (such as retrieving the current state of a process instance), the REST layer maps the RESTful Web Service API specified in Section 2 to the internal API of the process engine. To do so, the request and response payloads are processed by a set of media type

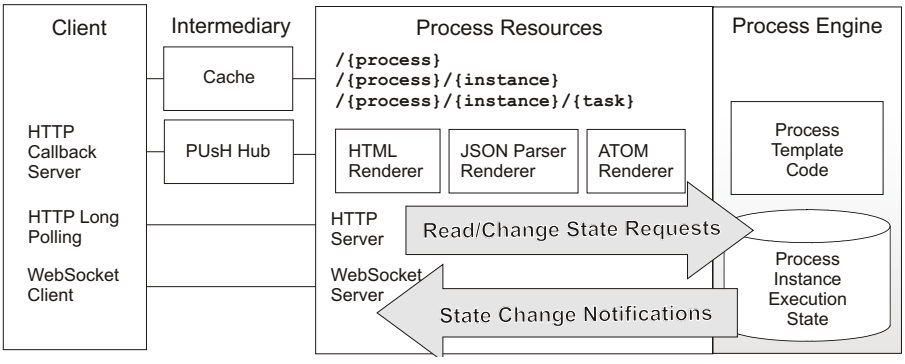


Fig. 4. Architecture of a Push-Enabled Process Execution Engine

parsers and renderers which share the responsability of turning the internal representations of the engine into the standard, externally visible formats (e.g., JSON, Atom, HTML) used to interact with the clients. These requests can be exchanged directly between a client and the server or go through an intermediary cache.

Concerning asynchronous notifications, we have experimented with some of the different mechanisms described in Section 4. Internally, the JOpera engine provides a call-back mechanism, which supports the dynamic installation of listeners which can intercept any state change of process and task instances. Thus, the REST layer can efficiently observe the behavior of the processes and propagate such state changes to interested clients using WebSockets and PuSH. Clients supporting the WebSocket protocol can open a connection using the `ws:/process/instance` URI, so that they can receive a message from the corresponding resource whenever its state changes. For simplicity, the message on the WebSocket is only used to indicate that an event has occurred and identify the affected resource. A client receiving it is expected to issue a basic `GET` HTTP request on the corresponding resource URIs to refresh its view on the current state of the resource. This way, all the existing mechanisms for caching, content-type negotiation and access control can be reused.

For clients supporting callbacks with PubSubHubbub, the feeds returned by the engine also contain a link relation 'hub' pointing them to where they can subscribe to receive notifications. Whenever a state change of a process instance occurs, the REST layer is notified by the engine and will ping the hub to inform it that the corresponding feed has changed. The hub will thus proceed to refresh the feed and efficiently call back all its subscribers.

Clients that neither support callbacks or WebSocket connections can still rely on HTTP long polling techniques. In our implementation we use a specific query parameter to specify whether a `GET /process/instance?notify=[stream|next]` request should be immediately answered with a snapshot of the current state (if the `notify` query parameter is missing), or the connection should block until some event occurs (`notify` query parameter is present). In this case we support two kinds of replies. With `notify=next` the response payload contains

a snapshot of the updated resource state and the HTTP connection is terminated once the transfer is complete. With `notify=stream` the HTTP connection remains open so that multiple events can be notified. This second variant works well in conjunction with browsers that support the execution of JSONP callbacks, which are written to the open HTTP response to indicate and identify the event.

6 Related Work

Generally speaking, extending REST with push capabilities can considered similar to the *Asynchronous REST (A+REST)* style proposed in *ARRESTED* [16]. However, while in that work the authors propose broadcast notifications, it is more realistic to assume that notifications are not being broadcast, but instead are sent using a publish/subscribe pattern or using multicast mechanisms [7]. From the REST perspective, the main point is that push has to built around the identification constraint, meaning that RESTful push needs to built around the idea that it is possible to get push notifications about updated resources. PuSH's approach of using feeds may be a very good starting point, because by using feeds it becomes possible not to simply subscribe to a resource, but to subscribe to a collection, and any changes to that collection will result in changes to the collection URI resource (the feed), as also discussed in [20] in the context of semantically annotated resources. Complementary to PuSH, a survey which includes a performance evaluation of different AJAX push and pull techniques can be found in [2].

The concept of using a business process modeling language to control the state of resources was first proposed in [19]. The idea of a RESTful Web service API to access the state of workflow instances has been also described in [28], where an ad-hoc solution based on client callback URIs was proposed to deal with the problem of push notifications. The feature of hypermedia-based discovery of the active tasks of a workflow was also featured in the Bite [5] project. In our architecture we introduce a more general design based on hierarchically nested feeds. As opposed to our previous work on the BPEL for REST [22] and BPMN for REST [23] extensions, in this paper we propose an orthogonal approach to publish processes as resources which does not require to add any language features beyond the ability to control which tasks should be published as resources.

7 Conclusions

In this paper we have shown how to use different push techniques to solve an important problem that occurs when publishing business processes as resources. We have presented the design of a RESTful Web Service API for a business process execution engine and motivated with an example the pressing need for clients to be notified when the process execution reaches a certain task. By publishing individual process instances as resources and by giving them a unique

identifier it becomes possible to support interactions that make use the uniform interface. By projecting part of the state of a process instance over a standard feed representation, it becomes possible for clients to subscribe to it and use optimizations such as the PubSubHubbub protocol to scale the corresponding delivery of notification callbacks. Without a feed or a similar construct, it becomes considerably harder to build push architectures, because of a lack of a collection construct (identified by a URI) and the individual items in that collection (also identified by URI). Pushing updates from the collection URI allows servers to notify clients of new resources (by adding hyperlinks to the push notifications), which allows REST's statelessness, global addressability and uniform interface constraints to be satisfied even in this scenario of a reverse interactions.

In future work we plan to further extend the system architecture with XMPP support and perform a scalability and performance evaluation to study the impact of the new REST layer on the performance of the business process engine and to compare the different alternative push mechanisms included in the architecture proposed in this paper. It will also be interesting to explore the design tradeoffs in the context of mobile clients, due to the emerging need to efficiently interact with a running business process from a mobile client.

References

1. Allamaraju, S.: RESTful Web Services Cookbook. O'Reilly & Associates, Sebastopol (2010)
2. Bozdag, E., Mesbah, A., Van Deursen, A.: A comparison of push and pull techniques for ajax. In: Proc. of the 9th IEEE International Symposium on Web Site Evolution (WSE 2007), pp. 15–22 (2007)
3. Brush, A.J.B., Bargeron, D., Grudin, J., Gupta, A.: Notification for Shared Annotation of Digital Documents. In: SIGCHI Conference on Human Factors and Computing Systems (CHI 2002), April 2002, pp. 89–96. ACM Press, Minneapolis (2002)
4. Christensen, J.H.: Using RESTful web-services and cloud computing to create next generation mobile applications. In: Proc. of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA 2009, Orlando, Florida, USA, pp. 627–634 (2009)
5. 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)
6. Eshuis, R., Grefen, P.W.P.J., Till, S.: Structured Service Composition. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 97–112. Springer, Heidelberg (2006)
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys 35(2), 114–131 (2003)
8. Fielding, R.T., Taylor, R.N.: Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002)
9. Fitzpatrick, B., Slatkin, B., Atkins, M.: PubSubHubbub, `http://code.google.com/p/pubsubhubbub/`
10. Gregorio, J.: URI Template. Internet Draft Draft-Gregorio-Uritemplate-04 (March 2010)

11. Guinard, D., Trifa, V., Wilde, E.: A Resource Oriented Architecture for the Web of Things. In: Second International Conference on the Internet of Things (IoT 2010), Tokyo, Japan (November 2010)
12. Hickson, I.: HTML5 — A Vocabulary and Associated APIs for HTML and XHTML. World Wide Web Consortium, Working Draft WD-html5-20110525 (May 2011)
13. Hickson, I.: Server-Sent Events. World Wide Web Consortium, Working Draft WD-eventsource-20110310 (March 2011)
14. Hickson, I.: The WebSocket API. World Wide Web Consortium, Working Draft WD-websockets-20110419 (April 2011)
15. Jordan, D., Evdemon, J.: Web Services Business Process Execution Language Version 2.0. OASIS Standard (April 2007)
16. Khare, R., Taylor, R.N.: Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems. In: 26th International Conference on Software Engineering, May 2004, ACM Press, Edinburgh (2004)
17. Loreto, S., Saint-Andre, P., Salsano, S., Wilkins, G.: Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. Internet RFC 6202 (April 2011)
18. OMG: BPMN: Business Process Modeling Notation 2.0. Object Management Group (2010)
19. Overdick, H.: Towards Resource-Oriented BPEL. In: Proc. of the 2nd ECOWS Workshop on Emerging Web Services Technology, WEWST 2007 (November 2007)
20. Passant, A., Mendes, P.N.: sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub. In: 6th Workshop on Scripting and Development for the Semantic Web, Crete, Greece (May 2010)
21. Pautasso, C.: JOpera: Process support for more than Web services, `http://www.jopera.org`
22. Pautasso, C.: RESTful Web Service Composition with BPEL for REST. Data & Knowledge Engineering 68(9), 851–866 (2009)
23. Pautasso, C.: BPMN for REST. In: Proc. of the 3rd International Workshop on the Business Process Management Notation, Luzern, Switzerland (November 2011)
24. Pohja, M.: Server Push for Web Applications via Instant Messaging. Journal of Web Engineering 9(3), 227–242 (2010)
25. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly & Associates, Sebastopol (2007)
26. Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. Internet RFC 6120 (March 2011)
27. Wilde, E., Liu, Y.: Feed Subscription Management. Tech. Rep. 2011-042, School of Information, UC Berkeley, Berkeley, California (May 2011)
28. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing Web Services Choreography Standards — The Case of REST vs. SOAP. Decision Support Systems 40(1), 9–29 (2005)

QoS Analysis for Web Service Compositions Based on Probabilistic QoS

Huiyuan Zheng[1], Jian Yang[1], Weiliang Zhao[1], and Athman Bouguettaya[2]

[1] Department of Computing, Macquarie University, Australia
{huiyuan.zheng,jian.yang,weiliang.zhao}@mq.edu.au
[2] School of Computer Science and Information Technology, RMIT University, Australia
athman.bouguettaya@rmit.edu.au

Abstract. Quality of Service (QoS) analysis and prediction for Web service compositions is an important and challenging issue in distributed computing. In existing work, QoS for service compositions is either calculated based on constant QoS values or simulated based on probabilistic QoS distributions of component services. Simulation method is time consuming and can not be used in real-time applications for dynamic Web service compositions. In this paper, we propose a calculation method to estimate the QoS of a service composition, in which the probability distributions of the QoS of component services can be in any shape. Experimental results show that the proposed QoS calculation approach significantly improves the efficiency in probabilistic QoS estimation.

1 Introduction

Web services technology creates the opportunity for building composite services by combining existing elementary or complex services (i.e. the component services) from different enterprises and in turn offering them as high-level services or processes (i.e. the composite services) [13]. QoS analysis becomes increasingly challenging and important when complex and mission critical applications are built upon services with different QoS. Thus solid model and method support for QoS predication in service composition becomes crucial in further analysis of complexity and reliability in developing service oriented distributed applications.

The QoS of a Web service is specified in service level agreement (SLA) between the service provider and service consumers. Most QoS in SLA are expressed as constant values [14]. Recently, probabilistic QoS has gained attentions because of the dynamic and unpredictable nature of Internet and its benefits have been recognized as being accurate and flexible [8,5,11]. With QoS modeled as probability distributions, service clients can have a better understanding of the performance of a service, and they can thereby have a relatively precise perception of the QoS of a composite service. A service provider also benefits from probabilistic QoS in SLA, because setting QoS as constant values in SLA does not reflect the dynamic characteristics of certain QoS metrics and may lead to pessimistic contracts [11].

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 47–61, 2011.

When the QoS of its component services are modeled as probability distributions, exiting work adopts simulation approach to compute the QoS distribution for a composite service [11]. Simulation method is time consuming. It works fine if used in design time when the architecture and component Web services of a service composition are determined. However, the service environment can be dynamic. Service-based processes should dynamically change to adapt to this environment. Composition engines, such as e-flow [4] and SELF-SERV [2], are designed for the purpose of run time composition. In these composition engines, QoS of the composite service needs to be estimated in real-time. Simulation method for QoS estimation will become a bottle-neck in real-time scenarios.

In order to overcome the problems mentioned above, we propose a method to estimate QoS for composite services. The basic idea is: (1) a QoS metric (e.g. execution time) of a component service is modeled as a probability distribution; (2) A composite service is modeled as a service graph and is composed of four basic patterns as *Sequential, Parallel, Conditional, and Loop*. QoS aggregation operations are defined and formulae are developed to compute QoS probability distributions for these patterns; (3) QoS calculation for a composite service becomes a matter of iteratively applying the QoS aggregation operations for these composite patterns. In comparison with the existing work, the contributions of this paper can be summarized as follows:

- In the proposed method, QoS for a composite service is calculated, not simulated. Experiments show that the proposed method is not only accurate but also much more efficient than simulation approach.
- The proposed method provides a more general and systematic approach compared with existing methods. As a result, the problems dealt with in the existing methods for QoS aggregation become the special cases in the proposed method. In this work, we do not have any assumptions on the forms of the QoS distributions, i.e., they can be any shaped probability distributions.

In the rest of the paper we will use the term *component QoS* and *composite QoS* to refer to QoS of component service and QoS of composite service respectively. We will also use QoS and QoS metric interchangeably.

The remainder of the paper is organized as follows: Section 2 discusses the work related to QoS estimation. In Section 3, the method of modeling and processing the structure for a composite service is introduced. In Section 4, QoS calculation method for Web service compositions is provided. Experiments are carried out in Section 5. Section 6 concludes the paper.

2 Related Work

We will first review QoS modeling method for Web services. Then, QoS monitoring methods for Web services will be summarized. Through these QoS monitoring methods, history QoS, i.e. QoS sample data, of a Web service can be obtained. The QoS sample data is a source of generating probability distributions for Web

services. Finally, current QoS estimation methods for service compositions will be discussed.

QoS Models: Existing research in service QoS representation can be categorized as: single values representation, multiple values representation, and standard statistical distributions. In most work, each QoS metric is represented as a constant value [7,14]. As the QoS of a Web service changes with time and environment settings, single value-modeled QoS does not reflect this variation. Standard statistical distributions are adopted to model QoS to solve the problem [3,11]. [3] mentions that a QoS metric can be specified as a distribution function, such as Exponential, Normal, Weibull, and Uniform. [11] argues that the contracts between Web service provider and client can be expressed as QoS probability distributions. T location-scale distribution is adopted to fit the original monitored QoS data of Web services. However, the reality is that an actual QoS probability distribution can come in any shape, which may not be able to fit into any well known statistical distributions. A more precise and general QoS modeling method has been proposed in our previous work [16], which is basically a free shaped probability distribution.

QoS Monitoring: The QoS of a Web service can be obtained through QoS monitoring. There are three strategies for QoS monitoring depending on where the measurement takes place: (1) Client-side monitoring: the measurement of QoS is run on the client side [9,12]. QoS metric that depends on user experience, such as response time, can be measured on the client side. (2) Server-side monitoring: the measurement of QoS is run on the server side [1]. This technique requires access to the actual Web service implementation, which is not always possible in practice. (3)Third party based monitoring: the measurement of QoS is run on a third party [18]. Third parties will periodically probes the service from different geographic locations under various network conditions and generate the QoS.

Composite QoS Aggregation: For single values represented QoS, aggregation method [3,7] is proposed to calculate the composite QoS. A composition can be regarded as being composed of different composition patterns. Formulae to calculate QoS for these patterns are given. But these formulae can only be applied to single values. For multiple values represented QoS [6], the calculation method is pretty much the same as it is for single values, except that the probability of each QoS value of the composite service are taken into account. For standard distribution represented QoS [3,11], simulation approaches are applied to estimate the composite QoS. A simulation needs to be run for thousands of times before a QoS sample for the composite service can be obtained. Simulation method is time consuming. An efficient method is necessary for estimating the QoS probability distributions of composite services, which is the focus of this paper.

Our previous work [15,16,17] provide solutions to problems in QoS analysis of Web service compositions. [17] proposes a modeling method for Web service compositions and composition patterns, as well as a QoS analysis method for Web service compositions whose component QoS are modeled as constant values. [15] designs an algorithm to explore the process of a Web service composition with

complex structures, such as unstructured loop patterns and nested composition patterns. Examples and experiments in [16] show that the inaccurate modeling of the QoS for component Web services can generate misleading QoS analysis results for Web service compositions. Discussion has been given on how to model the QoS probability distribution of a Web service accurately so that the QoS of a Web service composition can be estimated precisely. This paper gives a detailed solution to analyzing the QoS for Web service compositions whose component QoS are modeled as probability distributions. Experiments have been done to show the efficiency of our method over the existing ones. The techniques in [15,16,17] and this paper compose a systematical approach to analyzing the QoS for Web service compositions, which can be seen further in Section 3.

3 Preliminaries

A composite service can be built up based on four basic composition patterns: Sequential Pattern, Parallel Pattern, Conditional Pattern, and Loop Pattern. By recursively replacing patterns with single nodes having the same QoS as the composition patterns, a composite service will finally be represented by one node and the QoS of this node is the QoS of the composite service. Based on the QoS estimation method described above, it can be seen that three techniques are needed to compute the QoS for a Web service composition: (1) A modeling method for composite services and composition patterns; (2) A QoS calculation method for the four basic composition patterns; (3) An algorithm to explore the model of a composite service, identify composition patterns, calculate the QoS for the patterns, replace the patterns with the nodes with equivalent QoS, and finally get the QoS for the composite service.

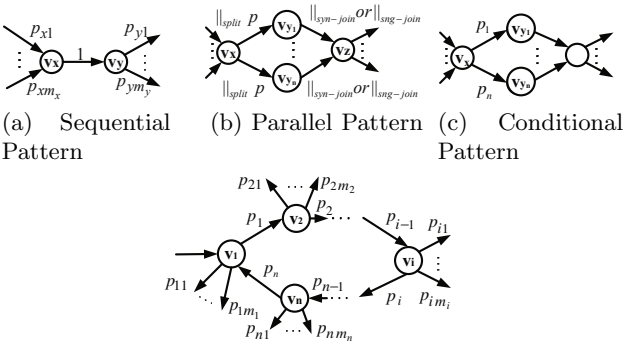
The solutions for (1) and (3) have been given in our previous work [17] and [15] respectively. We will briefly summarize the solutions for (1) and (3) in this section, then we will focus on (2), i.e., the QoS calculation method for composition patterns, in the next section.

3.1 Modeling Composite Services and Composition Patterns

A composite service is modeled as a Service Graph (see Figure 2 for an example of a Service Graph), in which each vertex represents a component service and each arc denotes a transition from one component service to another under a certain probability. Composition patterns (see Table 1 and Figure 1 for descriptions of the patterns) can be defined based on the definition of the Service Graph. The formal notations of a Service Graph and composition patterns are given in [17].

3.2 Model Processing Algorithm for Composite QoS Computation

In this paper, we will adopt the algorithm designed in our previous work [15] to calculate the QoS for a composite service. The input of the algorithm is the



(a) Sequential Pattern (b) Parallel Pattern (c) Conditional Pattern

(d) Loop Pattern

Fig. 1. Basic Composition Patterns

Table 1. Descriptions for Composition Patterns

Pattern	Description	
Sequential	In a Sequential Pattern (see Figure 1(a)), a Web service is invoked immediately after the completion of a preceding Web service.	
Parallel	Synchronized merge:	In a Parallel Pattern with synchronized merge (see Figure 1(b) in which '$||_{syn-join}$' represents synchronized merge), Web services are executed concurrently. The subsequent Web service will be invoked when all the Web services in the parallel pattern have finished running.
	Single merge:	In a Parallel Pattern with single merge (see Figure 1(b) in which '$||_{sng-join}$' represents single merge), Web services are executed concurrently. The subsequent Web service will be invoked when one of the Web services in the parallel pattern has finished running.
Conditional	In a Conditional Pattern (see Figure 1(c)), the Web services are run exclusively.	
Loop	In a Loop Pattern (see Figure 1(d)), the Web services are run repeatedly. A Loop Pattern has more than one entry or exit point.	

Service Graph of a composite service, as well as the QoS of the component services. The output of the algorithm is the QoS for the composite service.

Different from [15] in which the QoS are single values, the QoS in this paper are probability distributions. Therefore, the QoS calculation methods for composition patterns in the algorithm should be changed accordingly, which will be discussed in detail in the following section.

4 Probabilistic QoS Aggregation

QoS aggregation formulae are developed in this section to calculate the QoS for composition patterns.

4.1 Approach Overview and Underlying Assumptions

A QoS metric of each service (either a component service or a composite service) is seen as a random variable and the following assumptions have been made:

(1) The QoS[1] of different component services are mutually independent, i.e. the QoS of any two component services are independent. If the QoS of two services are independent, the QoS of one service does not affect the QoS of the other service.

(2) QoS control is out of the scope of this paper. We only consider the case that the developer of a composite service makes use of the component services but has no control of the QoS of the component services. The QoS probability distributions of a component service are statistically estimated and have already taken into account different QoS influencing factors such as workload.

(3) The transition probabilities from one service to another in a composite service can either be provided according to the experience of the service developer at design time or be statistically estimated based on the execution history of the service. Detailed method of obtaining the transition probabilities in a composite service can be found in [3].

(4) QoS are represented as histograms with the same start point and width of intervals. There is more accurate method of getting the QoS probability distribution based on a QoS sample, which is out of the scope of this paper and can be found in [16].

4.2 QoS Probability Distribution Calculation for Composition Patterns

Classification of QoS Metrics. The QoS metrics are classified into five categories according to their characteristics in different composition patterns, which are: additive, multiplicative, concave (i.e. minimum), convex (i.e. maximum), and weighted additive. For example, the QoS metric execution time reflects an additive behavior in a Sequential Pattern and convex behavior in a Parallel Pattern with synchronized merge. In this paper, the discussion of QoS analysis is based on these categories instead of individual QoS metrics, which makes the QoS analysis approach more general and fits more QoS metrics.

Examples of additive, multiplicative, concave, and convex QoS metrics are cost, reliability, execution time of a Parallel Pattern with single merge, and execution time of a Parallel Pattern with synchronized merge respectively.

It is worth mentioning that multiplicative QoS metrics such as availability, reliability, and accessibility, are represented as a statistical percentage value (e.g. 90%) rather than a distribution. Therefore, only four types of QoS metrics: additive, concave, convex, and weighted additive will be discussed for the computation of composite QoS distribution in the rest of the paper.

QoS Calculation Operations. If the component QoS is represented by single value, to calculate the composite QoS, operations are sum, minimum, maximum, and weighted sum for additive, concave, convex, and weighted additive

[1] Unless indicated otherwise, QoS is referred as the same QoS metric.

QoS metrics respectively. However, the computation of the QoS distribution for a composition pattern based on probability-distribution represented component QoS is far more complex than based on single-value represented component QoS. We thus define four operations on probability distributions, which are: QoSSSum, QoSMin, QoSMax, and QoSWeightedSum, to distinguish the arithmetic operations on single values.

- QoSSSum(denoted as $\circledast$): operates on the component QoS distributions taking into consideration of the addition of their QoS values;
- QoSMin(denoted as $\circledwedge$): operates on the component QoS distributions taking into consideration of the minimum of their QoS values;
- QoSMax(denoted as $\circledvee$): operates on the component QoS distributions taking into consideration of the maximum of their QoS values;
- QoSWeightedSum(denoted as $\odot$): operates on the component QoS distributions taking into consideration of the addition of their QoS values with path probabilities as weights. It is mainly used in Conditional and Loop Patterns.

These operations and their relationships with composition patterns and QoS metrics are summarized in Table 2.

Table 2. Operations for QoS Aggregation

Pattern	Operation	QoS Metric
Sequential	QoSSSum	Additive
	QoSMin	Concave
Parallel	QoSSSum	Additive
	QoSMin(single-merge)	Concave
	QoSMax(synchronized-merge)	Convex
Conditional	QoSWeightedSum	any
Loop	QoSSSum&QoSWeightedSum	Additive
	QoSMin&QoSWeightedSum	Concave

Formulae are developed for these operations. We introduce the following naming conventions:

- q is a variable representing a QoS metric;
- $f(q)$ denotes the density function of the probability distribution (**PDF**);
- $F(q)$ denotes the cumulative distribution function (**CDF**); $F(q)$ and $f(q)$ have the following cumulative relationship: $F(q) = \int_{-\infty}^{q} f(x)dx$ for continuous distributions or $F(q) = \sum_{q_i <= q} f(q_i)$ for discrete distributions.

It should be noted that although the discussion is based on distributions, the developed formulae are also applied to single values. This is because single values can also be represented as distributions with the help of Dirac delta function[2]. For example, if the cost of a Web service is M, then $f(q) = \delta(q - M)$. If the

[2] $\delta(x)$ is the Dirac delta function. $\delta(x) = +\infty$ when $x = 0$ and $\delta(x) = 0$ when $x \neq 0$.

cost of a Web service is N_1 with a probability of p_1 and N_2 with a probability of p_2 ($p_1 + p_2 = 1$), then the distribution of this Web service can be expressed as $f(q) = p_1\delta(q - N_1) + p_2\delta(q - N_2)$.

QoSSum. Computing the PDF of the QoSSum of two component QoS distributions is a problem of deducing the PDF of the sum of independent variables, which is the convolution of each of their density functions [10],

$$f(q) = f_1(q) \circledast f_2(q) = (f_1 * f_2)(q) = \int_{\eta=0}^{q} f_1(\eta) f_2(q - \eta) d\eta \tag{1}$$

where $f(q)$ is the PDF of the QoS of a composition pattern, $f_1(q)$ and $f_2(q)$ are the PDFs of the component services.

Let us take a simple example of the execution time of a Sequential Pattern with two component services. The PDFs of the execution time of the two component services are $f_1(t)$ and $f_2(t)$ respectively. The probability for the execution time of the first service being τ ($\tau \in (0, t)$) and the second service being $t - \tau$ ($t \in (0, +\infty)$) is $f_1(\tau) f_2(t - \tau)$. Therefore, the probability for the Sequential Pattern being finished at time t is the integral of $f_1(\tau) f_2(t - \tau)$ over $(0, t)$ where τ is the variable, i.e. $f(t) = \int_{\tau=0}^{t} f_1(\tau) f_2(t - \tau) d\tau$. The result is the same as what we get from Formula (1).

QoSMin. The probability distribution of the QoSMin of n component QoS distributions is the distribution of the minimum of n independent variables which can be calculated as:

$$F(q) = F_1(q) \circledcirc \ldots \circledcirc F_i(q) \circledcirc \ldots \circledcirc F_n(q) = 1 - \prod_{i=1}^{n} [1 - F_i(q)] \tag{2}$$

where $F(q)$ is the CDF of the QoS of a composition pattern; n is the number of component services within this pattern; and $F_i(q)$ is the CDF of the QoS of component service i.

Then the PDF can be obtained by differentiating both sides of Equation (2) with respect to q:

$$f(q) = f_1(q) \circledcirc \ldots \circledcirc f_i(q) \circledcirc \ldots \circledcirc f_n(q) = \sum_{i=1}^{n} f_i(q) \prod_{j=1,\ldots,n \& j \neq i} [1 - F_j(q)] \tag{3}$$

where $f(q)$ is the PDF of a composition pattern; n is the number of component services; $f_i(q)$ is the PDF of component service i; and $F_j(q)$ is the CDF of component service j.

Let us take a QoS metric, response time as an example. Assume that X and Y are two Web services in a Parallel Pattern with single merge. The probabilities for them to be finished within time t are $F_X(t)$ and $F_Y(t)$ respectively. The probability for neither of them being able to finish within time t is $(1 - F_X(t))(1 - F_Y(t))$, therefore, the probability for either of them being able to finish within time t is $1 - (1 - F_X(t))(1 - F_Y(t))$. The fact that at least one of the Web services

can be finished within t means that t is the shorter execution time of the two Web services.

QoSMax. The distribution of the QoSMax of n component QoS distributions is the distribution of the maximum of n independent variables which can be calculated as:

$$F(q) = F_1(q) \circledcirc \ldots \circledcirc F_i(q) \circledcirc \ldots \circledcirc F_n(q) = \prod_{i=1}^{n} F_i(q) \tag{4}$$

where $F(q)$ is the CDF of the QoS of a composition pattern; n is the number of component services within this pattern; and $F_i(q)$ is the CDF of the QoS of component service i.

The PDF can be obtained by differentiating both sides of Equation (4) with respect to q:

$$f_(q) = f_1(q) \circledcirc \ldots \circledcirc f_i(q) \circledcirc \ldots \circledcirc f_n(q) = \sum_{i=1}^{n} f_i(q) \prod_{j=1,\ldots,n \& j \neq i} F_j(q) \tag{5}$$

where $f(q)$ is the PDF of a composition pattern; n is the number of component services within this pattern; $f_i(q)$ is the PDF of component service i; and $F_j(q)$ is the CDF of component service j.

Let us take execution time as an example. Assume that X and Y are two concurrently running Web services in a Parallel Pattern with synchronized merge. The probability for X and Y to be finished within time t is $F_X(t)$ and $F_Y(t)$ respectively. Therefore, the probability for both of them to be finished within time t is $F_X(t)F_Y(t)$. The fact that both Web services can be finished within t means that t is the longer execution time of the two Web services.

QoSWeightedSum. The QoS distribution for the QoSWeightedSum of component QoS distributions can be calculated as

$$f(q) = f_1(q) \odot \ldots \odot f_i(q) \odot \ldots \odot f_n(q) = \sum_{i=1}^{n} p_i f_i(q) \tag{6}$$

where $f(q)$ is the PDF of a composition pattern; n is the number of component services within this pattern; $f_i(q)$ is the PDF of component service i; and p_i is the execution probability for component service i.

Here we can take execution time as an example. Assume X and Y are two Web services within a Conditional Pattern with the execution probabilities being p_1 and p_2 respectively. The probabilities for X and Y to be finished at time t are $f_X(t)$ and $f_Y(t)$ respectively. Therefore, the probability for the path of X to be finished at time t is $p_1 f_X(t)$ and for the path of Y to be finished at time t is $p_2 f_Y(t)$. Therefore, the probability for the Conditional Pattern to be finished at time t is $f(t) = p_1 f_X(t) + p_2 f_Y(t)$.

QoS Probability Distribution Calculation for Composition Patterns. So far, we have discussed the operations and formulae involved in computing

composite QoS distributions. Next, we will explain how component QoS distributions are aggregated for different composition patterns. Here, QoS metrics cost and time (execution time or response time) will be discussed as examples.

For a composition pattern with two component services, assume the probability distribution of the composite QoS is $c(q)$ for cost, $t(q)$ for time, and the probability distributions of component QoS are $c_1(q)$ and $c_2(q)$ for cost, $t_1(q)$ and $t_2(q)$ for time. According to Table 2, there are:

- in a Sequential Pattern:
  $c(q) = c_1(q) \circledast c_2(q)$; $t(q) = t_1(q) \circledast t_2(q)$.
- in a Parallel Pattern with synchronized merge:
  $c(q) = c_1(q) \circledast c_2(q)$; $t(q) = t_1(q) \circledcirc t_2(q)$.
- in a Parallel Pattern with single merge:
  $c(q) = c_1(q) \circledast c_2(q)$; $t(q) = t_1(q) \circledwedge t_2(q)$.
- in a Conditional Pattern:
  $c(q) = c_1(q) \odot c_2(q)$; $t(q) = t_1(q) \odot t_2(q)$.
- in a Loop Pattern:
  The QoS computation for Loop Patterns is more complicated than other patterns. Next, we will discuss it in detail.

In [17], we have given detailed discussion on the structure analysis method to compute the QoS for an arbitrary Loop Pattern whose component QoS are fixed constant values. To sum up the method in [17], statistically, a Loop Pattern can be seen as a Conditional Pattern with a Sequential Pattern in each path. With the formula for calculating the execution probability of each path of the Conditional Pattern given in [17] (see Formula 7) and the formulae of computing the QoS of a Sequential Pattern and Conditional Pattern given in this paper, the distribution of the QoS of a Loop Pattern can be computed.

$$p_{path_{li}} = (\prod_{k=1}^{n} p_k)^l (\prod_{k=0}^{i-1} p_k)(1 - p_i) \tag{7}$$

where p_k is the transition probability from vertex v_k to v_{k+1} and $p_k = 1$ when $k = 0$, l is the number of times that the Loop is executed, n is the number of vertices in the Loop, and i is the index of the vertex where the Loop is jumped out of.

To compute the QoS distribution for a Loop Pattern, we can set a threshold value, TH, for $p_{path_{li}}$. When $p_{path_{li}} < TH$, the probability for the loop still being run is quite small. Therefore, the execution path with a probability smaller than TH can be ignored. It means that $l = L$ times of looping is enough if L satisfies $(\prod_{k=1}^{n} p_k)^L (\prod_{k=0}^{i-1} p_k)(1 - p_i) < TH$.

The transition probability for each outgoing arc of a Loop Pattern (i.e. p_{ij} in Figure 1(d) where $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$) has to be changed accordingly. Detailed formula on calculating the probabilities of the outgoing arcs can be found in our previous work [17]. Then, the Loop Pattern can be replaced by one vertex.

5 Experiment

In this section, experiments have been done to compare the performance of the proposed QoS calculation method (referred to as *calculation method*) with simulation method. In a simulation method, the execution of a composite service is simulated by exploring the Service Graph of the composite service. One single value for per QoS metric of the composite service is obtained for each run of a simulation by aggregating the QoS of each vertex that has been visited during the exploration of the Service Graph. After running the simulation for a number of times, a QoS sample (containing all the simulated QoS) for the composite service can be obtained. This QoS sample can be used to generate the QoS probability distribution for a composite service.

5.1 Validation

First, we shall test the accuracy of the calculation method and the simulation method mentioned earlier.

A composite service and its component services (see Figure 2) are deployed. Experiments have been done to monitor the QoS of the deployed composite service. The monitored QoS are referred to as *experimental result*. By comparing the composite QoS obtained by the simulation method and the calculation method (referred to as *simulation result* and *calculation result* respectively) with experimental result, the accuracy of the simulation method and the calculation method can be verified. We only consider the QoS metric execution time in the experiments.

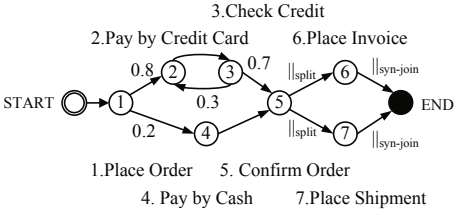
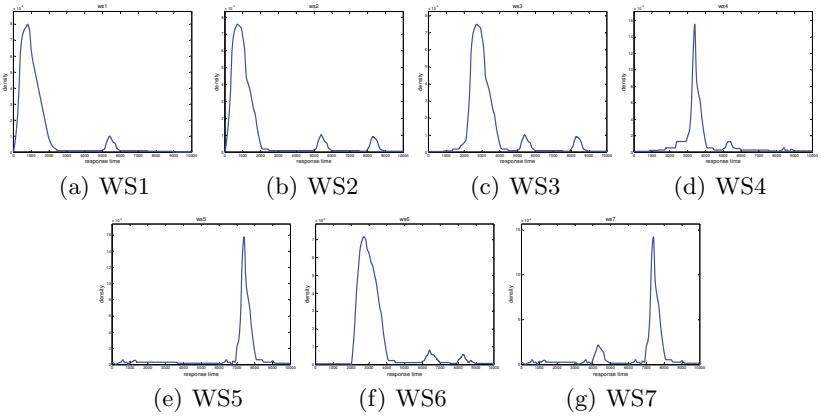


Fig. 2. An Example of A Service Graph

The seven component Web services in Figure 2 are developed and deployed on Apache Tomcat 5.5 server. Their execution time distributions follow the distributions in Figure 3[3]. The BPEL process executing the composite service in Figure 2 is developed and deployed on an Active BPEL engine. The detailed information on service deployment is as follows:

(1) The simulation of the QoS probability distribution for a component Web service: An array containing 10000 values whose distribution conforms to the probability distribution of the Web service is generated and stored in a file. For each execution, the Web service will randomly read one value from the file and suspend for the indicated amount of time before it sends out a response.

[3] These distributions are generated manually.



(a) WS1 (b) WS2 (c) WS3 (d) WS4

(e) WS5 (f) WS6 (g) WS7

Fig. 3. Probability Distributions of execution time

(2) The simulation of the transition probabilities within a composite service: A random number generator conforming to a uniform distribution is used. At component service 1 *Place Order*, a random number is generated and compared with 0.8. If it is smaller than 0.8, the output of service 1 is "Credit Card"; otherwise, the output is "Cash". At service 3 *Check Credit*, if the generated number is smaller than 0.7, the output is "Approved"; otherwise, it is "Disapproved".

(3) Experimental result: The developed composite service is invoked for 10,000 times. For each invocation, an execution time is recorded. A histogram, shown in Figure 4, is generated based on the recorded data sample.

(4) Simulation result: Simulation result is in the form of a sample. To distinguish the simulation result from the histogram of experimental result, the simulated QoS are shown as dot-dashed curves in Figure 4(a), i.e. we plot the probability densities at different execution time in Figure 4(a) instead of histogram bars.

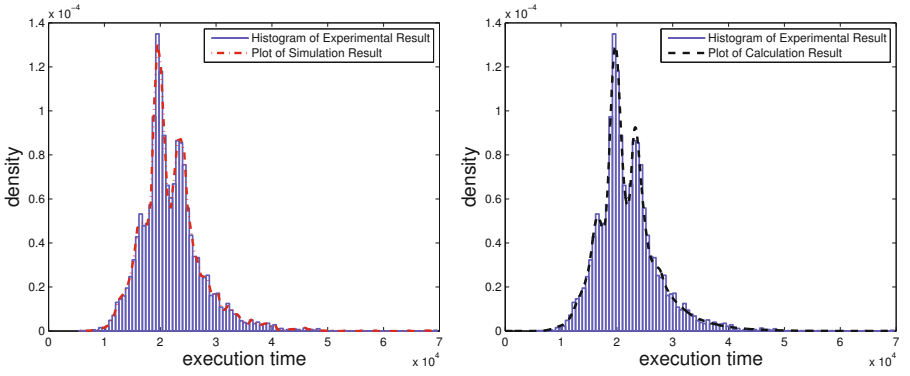
(5) Calculation result: The calculation result is shown as dashed curves in Figure 4(b) by plotting the probability densities at different execution time.

It can be seen from Figure 4 that both the simulation result and the calculation result fit the experimental result very well. The accuracy of both methods has been verified.

5.2 Efficiency

Next, the efficiency of using calculation method and simulation method will be compared.

We perform tests on Mac OS X 10.6.6 with 1.86 GHz Intel Core 2 Duo processor and 2 GB memory. Both the proposed QoS calculation and simulation methods are implemented using C/C++ language. We test the time spent on QoS estimation by calculation and simulation methods for Sequential Patterns, Parallel Patterns, Conditional Patterns, and Loop Patterns respectively. The results are plotted by Matlab and shown in Figures 5, 6, 7, and 8 respectively. The



(a) Experimental and Simulation Results (b) Experimental and Calculation Results

Fig. 4. Results of Validation

x-axis represents the number of component services in a composite service and the y-axis represents the time (in μs) spent on estimating the QoS distribution for a composite service. As the time spent on calculation method is significantly shorter than simulation method, we present the computation time of different methods in logarithmic scale. In each of Figures 5, 6, 7, and 8, there are four dashed lines and two solid lines. The four dashed lines represent the time spent on simulation method when the simulation is run for 5000, 10000, 15000, and 20000 times respectively. The two solid lines represent the time spent on calculation method when the probability distribution of each component QoS has 512 and 1024 bins respectively. One thing is to be noted: the time spent by simulation method changes irregularly for any Loop Patterns. This is because in the experiment, the transition probabilities in the Loop Pattern, the number of component services that can jump out of the Loop Pattern, the component services that jump out of the Loop Pattern, and the jumping out probabilities all change randomly when the number of component services changes.

Based on the performance comparison between calculation and simulation methods, it can be seen that the proposed QoS calculation method is far more

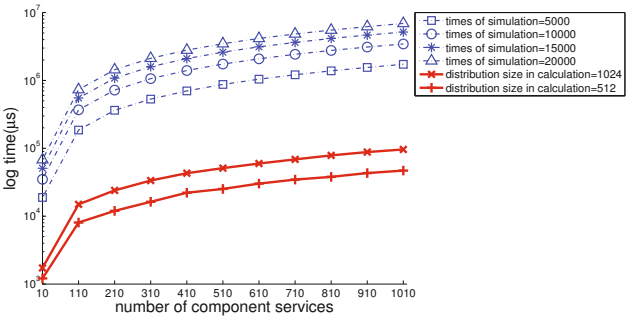


Fig. 5. Performance Comparison - Sequential Pattern

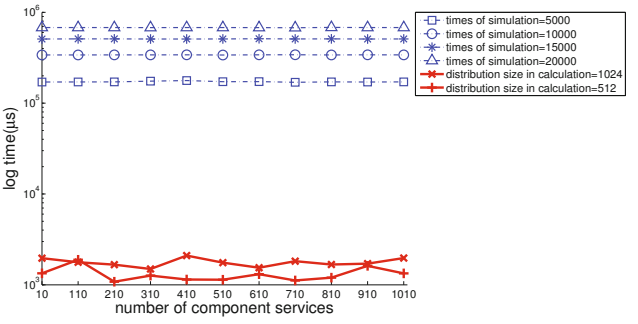


Fig. 6. Performance Comparison - Parallel Pattern

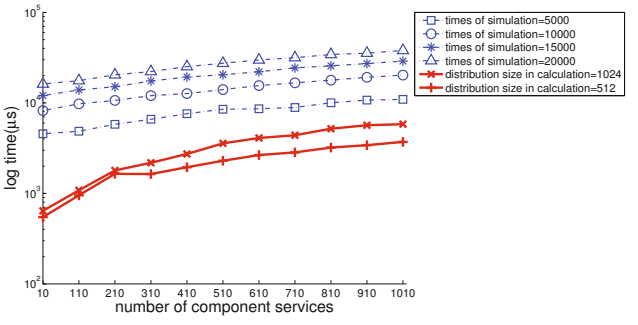


Fig. 7. Performance Comparison - Conditional Pattern

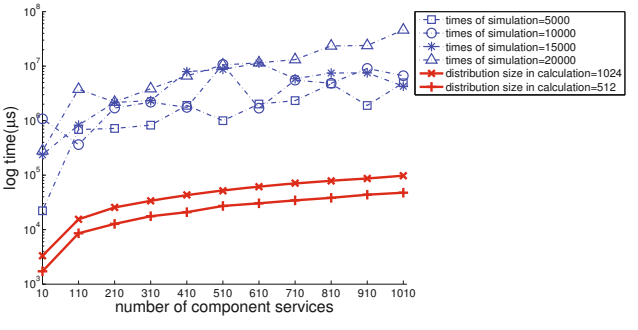


Fig. 8. Performance Comparison - Loop Pattern

efficient and outperforms simulation method in terms of computing QoS for all the basic composition patterns.

6 Conclusion

In this paper, we propose a systematic approach to calculate the QoS probability distribution for composite services. Experimental results show that the proposed QoS calculation method is far more efficient than existing method and can be used in real-time scenarios.

The proposed method is based on the assumption that the QoS of component services are independent of each other, which is not always the case in reality. Research will be done to relax this assumption to make the proposed QoS aggregation method more robust to fit into any environment.

References

1. Artaiam, N., Senivongse, T.: Enhancing service-side qos monitoring for web services. In: ACIS, pp. 765–770 (2008)
2. Benatallah, B., Sheng, Q.Z., Ngu, A.H.H., Dumas, M.: Declarative composition and peer-to-peer provisioning of dynamic web services. In: ICDE, pp. 297–308 (2002)
3. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Quality of service for workflows and web service processes. Journal of Web Semantics 1, 281–308 (2004)
4. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and Dynamic Service Composition in eFLOW. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
5. Comuzzi, M., Pernici, B.: A framework for qos-based web service contracting. TWEB 3(3), 10:1–10:52 (2009)
6. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the qos of web-services-based workflows. Inf. Sci. 177(23), 5484–5503 (2007)
7. Jaeger, M., Rojec-Goldmann, G., Muhl, G.: Qos aggregation for web service composition using workflow patterns. In: EDOC, pp. 149–159 (2004)
8. Klein, A., Ishikawa, F., Honiden, S.: Efficient QoS-Aware Service Composition with a Probabilistic Service Selection Policy. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 182–196. Springer, Heidelberg (2010)
9. Mani, A., Nagarajan, A.: Understanding Quality of Service for Web Services. IBM Software labs, India
10. Papoulis, A.: Probability, random variables, and stochastic processes. McGraw-Hill, New York (1965)
11. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic qos and soft contracts for transaction-based web services orchestrations. IEEE Transactions on Services Computing 1(4), 187–200 (2008)
12. Rosenberg, F., Platzer, C., Dustdar, S.: Bootstrapping performance and dependability attributes of web services. In: ICWS, pp. 205–212 (2006)
13. Yang, J., Papazoglou, M.P.: Service components for managing the life-cycle of service compositions. Inf. Syst. 29(2), 97–125 (2004)
14. 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)
15. Zheng, H., Yang, J., Zhao, W.: Qos analysis and service selection for composite services. In: SCC, pp. 122–129 (2010)
16. Zheng, H., Yang, J., Zhao, W.: Qos probability distribution estimation for web services and service compositions. In: SOCA, pp. 1–8 (2010)
17. Zheng, H., Zhao, W., Yang, J., Bouguettaya, A.: Qos analysis for web service composition. In: SCC, pp. 235–242 (2009)
18. Zheng, Z., Zhang, Y., Lyu, M.R.: Distributed qos evaluation for real-world web services. In: ICWS, pp. 83–90 (2010)

Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations*

Dragan Ivanović[1], Manuel Carro[1,2], and Manuel Hermenegildo[1,2]

[1] School of Computer Science, T. University of Madrid (UPM), Spain
idragan@clip.dia.fi.upm.es, {mcarro,herme}@fi.upm.es
[2] IMDEA Software Institute, Spain

Abstract. Service compositions put together loosely-coupled component services to perform more complex, higher level, or cross-organizational tasks in a platform-independent manner. Quality-of-Service (QoS) properties, such as execution time, availability, or cost, are critical for their usability, and permissible boundaries for their values are defined in Service Level Agreements (SLAs). We propose a method whereby constraints that model SLA conformance and violation are derived at any given point of the execution of a service composition. These constraints are generated using the structure of the composition and properties of the component services, which can be either known or empirically measured. Violation of these constraints means that the corresponding scenario is unfeasible, while satisfaction gives values for the constrained variables (start / end times for activities, or number of loop iterations) which make the scenario possible. These results can be used to perform optimized service matching or trigger preventive adaptation or healing.

Keywords: Service Orchestrations, Quality of Service, Service Level Agreements, Monitoring, Prediction, Constraints.

1 Introduction

Service-Oriented Computing is a paradigm that has been increasingly gaining ground as the basis for development of highly flexible, dynamic, and distributed service-based applications (SBAs). Key to the development of SBAs are service compositions, that allow the application designer to put together several loosely-coupled specialized component services, often provided and controlled by third parties, to perform more complex, higher-level, and/or cross-organizational tasks [7]. Trends in service-oriented application design indicate increased reliance on third-party services available on Internet [19].

In that context, quality of service (QoS) properties of individual services and their compositions are critical for overall usability. For externally offered services, service-level agreements (SLAs) define boundaries of permissible values for QoS attributes, such as execution time, availability, or cost. Potential and actual SLA violations can be

* 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). The authors were also partially supported by Spanish MEC project 2008-05624/TIN *DOVES* and CM project P2009/TIC/1465 (*PROMETIDOS*).

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 62–76, 2011.

avoided or mitigated using some form of adaptation (e.g., rebinding or changing the service selection preferences), or triggering structural changes both in the design and the running instance [7,10]. For structurally constrained compositions of non-cyclic shape, flexible provisioning techniques have also been proposed [18].

Therefore, the task of analyzing and predicting QoS metrics for service compositions, both at design time and at the level of an executing instance, is of great theoretical and practical importance. Among the recently proposed approaches we can cite the application of statistical reasoning based on historical data (e.g., data mining) to predict likely SLA violations and their probable causes [15,23], or to apply techniques related to model checking and online testing [10,8].

In this paper, we take a different approach based on generating a constraint model for QoS metrics of an executing composition based on its structure, the semantics of its building blocks, and its current state of execution at a given moment. Previous works [4,3,13] also used the composition structure as the basis to derive properties thereof. In terms of results, instead of trying to find the most likely SLA conformance or violation scenario, we identify the possible cases of SLA conformance and violation at a given point of execution and infer conditions under which these may occur.

We consider service orchestrations, which are compositions with a centralized control flow. They may involve a wide range of workflow patterns [22] — including parallel flows, different splits/joins, loops, branches, etc. — and are usually expressed using some dedicated notation, such as BPMN [16], BPEL [14], Yawl [20] or DecSerFlow [21], or other adequate formalism. In this paper, we use abstract (but executable) notation for orchestrations from which we formulate a constraint satisfaction problem (CSP) [6,1] that models the situation of SLA conformance or violation.

The rest of the paper proceeds as follows: Section 2 presents a motivating example. Section 3 then describes how the CSP can be automatically formulated on the basis of an orchestration continuation, to take into account the known assumptions about third-party components, as well as to include internal structural parameters of branches and loops. In Section 4 we present an experimental evaluation, Section 5 gives some implementation notes, and finally Section 6 presents conclusions.

2 Motivation

Consider a scenario where a provider of multimedia content (text, audio and video) needs to periodically update and reconfigure program streams offered to individual clients (users), based on their historical usage patterns. That may require choosing between different mixtures of available streams (such as news, sport, entertainment, etc.) presented to a user, genres within them, and type of multimedia materials. The choice may depend on the frequency of use (casual vs. frequent users), user interests, and bandwidth adequate to serve different types of content (e.g. low quality vs. HD video). In such a scenario, the provider would run the reconfiguration process from time to time when serving user requests, although typically not for each access. Reconfiguration depends on other (usually back-end) administrative and analytic services, and should not cause noticeable glitches in content delivery. The SLA for the content delivery service does provide some window for running the reconfiguration process on top of it, but it

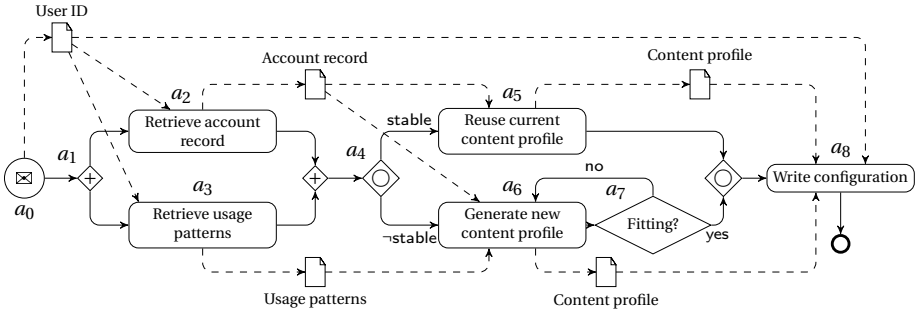


Fig. 1. An example orchestration to reconfigure content provided to a user

is normally very restricted. Therefore, the running time of the reconfiguration process and its availability are of the utmost importance.

Fig. 1 depicts an example orchestration implementing the reconfiguration process, using BPMN notation [16]. It starts with the reception of user ID (activity a_0), which spawns in parallel (a_1) the retrieval of the users' account record (a_2) and the user's usage patterns (a_3). If the usage pattern is stable (a_4), the user's current content profile is reused (a_5). Otherwise, a new content profile is generated (a_6) based on the account record and the current usage patterns. For efficiency, first minor variations in content profile parameters are attempted; if these are not likely to fit the usage pattern (a_7), more radical changes are attempted, and so on. Finally, the content profile (either the current one or a new one) is written to the configuration database (a_8).

In this example, the configuration process may affect responsiveness of the main multimedia content delivery service, and therefore we want to continuously monitor and predict reconfiguration running time, having in mind the overall SLA. At any point in the execution of the reconfiguration orchestration, including its start, and within that particular context, there are a number of interesting objectives to aim at:

Predicting Certain SLA Violations: If we are able to predict that the orchestration cannot possibly meet the SLA constraints, then we can either abort it (effectively postponing the reconfiguration), or adapt it by switching to a simpler and/or more robust version. Conversely, if we are reasonably sure that the execution will be SLA-conformant, we can plan to use the potential slack in a productive way.

Predicting Possible SLA Violations: If we can predict that SLA violations may occur, but not necessarily so, and we can identify potential points of failure, then we can prepare, ahead of time, adequate adaptation and healing mechanisms, and/or try to decrease the risk of violation by using fail-safe component services.

Inferring the Necessary Preconditions: If we not only predict, but understand *why* an SLA violation may or must happen, we can use that information to identify bottlenecks, to develop criteria for selection of components, and to drive either runtime or design-time adaptation.

In this paper we present a unified constraint-based approach and analysis framework that makes it possible to perform runtime prediction of SLA violation / conformance for service orchestrations, based on monitoring information and a constraint model of an

abstract semantics of the orchestration structure. Predictions are based on and expressed in a form that describes the circumstances under which SLA violations and conformant executions of an orchestration may take place, which can be used to reason about the orchestration and its components.

3 Constraint-Based QoS Prediction

3.1 The General Prediction Framework

An SLA typically defines, among other things, which QoS attributes are relevant in the context of the provider-client contract, and what values of these QoS attributes are acceptable. For QoS attributes expressed as numbers on a measurement scale, QoS constraints given by an SLA are often expressed as ranges of permissible values for each attribute. More complex relationships between SLA attributes — such as trade-offs between cost and speed — can be devised, but in our analysis we will assume that the QoS constraints are given as lower and upper bounds on appropriate QoS metrics.

Furthermore, we will focus on an important subset of QoS metrics that are *monotonic* and *cumulative* in the sense that they express an amount of a physical or logical resource consumed by each activity in an orchestration, so that the amounts from subsequent activities add together into an aggregate metrics. Running time is an obvious example of a cumulative metrics, because consumed time is never recovered. In this paper we will assume, for simplicity, that metrics are accumulated by through addition (which is a fairly common case). Note that some metrics whose natural aggregation function is not addition can be easily mapped into additive metrics. For instance, the aggregation function for the availability (the probability of successful access) p of n subsequent operations can be calculated as $\prod_{i=1}^{n} p_i$, where $0 < p_i \leq 1$ is the availability of the i-th component. Using the transformation $\lambda = -\log p$, we can transform the original multiplicative metric of p into the additive $\lambda = \sum_i \lambda_i$.

An important feature of a cumulative QoS metrics is that, at any point in execution of an orchestration, its value can be calculated as a cumulative function (such as addition) of two components: the previously *accumulated metrics* and an estimate of the *pending metrics* for the remainder of the execution of the orchestration, until it finishes. For some metrics, their accumulated value needs to be be measured taking into account the history of the actual execution up to the current execution point (e.g. elapsed time from the start of execution), while for other metrics the current value at any execution point does not depend on the previous history. For example, in the case of availability the current metrics always represents "availability so far". Since it is being measured at some execution point which has obviously been reached, the probability p of being available up to the point of measure is 1 (and then $\lambda = 0$).

Let us present intuitively how accumulated metric values and a prediction for the rest of execution can be applied to predict SLA violations. We will use Fig. 2, taken from [12]. Points $\mathscr{A}$-$\mathscr{D}$ on the x-axis stand for the start, finish and two intermediate points in time during the execution of an orchestration. Let us assume that at the initial point $\mathscr{A}$ we have a prediction (solid line) for the QoS metrics for the rest of the execution. According to this prediction, the QoS at the finish falls under the limit *Max* given by some SLA. However, at point $\mathscr{B}$ we notice that some deviations have occurred up to

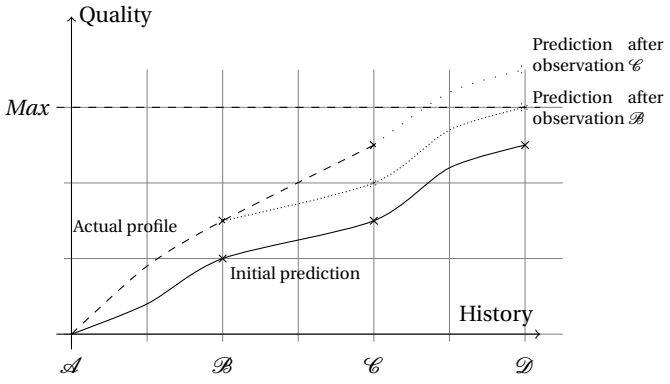


Fig. 2. Actual and predicted QoS throughout history

that moment (the dashed line). Therefore, we adjust our prediction, which now seems to indicate borderline SLA compliance. At point $\mathscr{C}$, further measured deviations lead to another adjustment of the QoS prediction, this time indicating a likely violation of the SLA.

An important aspect of such prediction scheme is the existence of a time horizon between the detection of the possibility of an SLA violation and its actual occurrence. In our example, it is the period between $\mathscr{B}$ and the point of failure which lies somewhere between $\mathscr{C}$ and $\mathscr{D}$. This "window" makes it possible to warn about (potential) future SLA violations ahead of time. A prediction technique also needs to identify conditions that increase or decrease likelihood of an SLA violation, in order to filter false positives from true positives and thus increase the reliability of prediction. These conditions can be related to internal parameters of the orchestrations, such as the truth value of branching conditions or the number iterations in a loop. For our constraint-based approach, this will be illustrated in Section 3.5.

3.2 QoS Prediction Architecture

The architecture of the constraint-based QoS prediction framework is shown in Fig. 3. A process engine executes service orchestrations and interacts with external services by exchanging messages. In the process, it publishes lifecycle events such as signaling the start or end of a process, invocation of a component service, and reception of a reply. Also, from time to time, the process engine publishes the current point of execution of a running orchestration in the form of a continuation (explained in the following subsection). That is typically not done at each step, but at specific milestones such as service invocations, loop iterations and branches. Deciding how to determine the optimal granularity for publishing points is a matter for future work.

The events published by the process engine are sent via an event bus. The constraint-based QoS predictor can be connected to that bus and listen to lifecycle events (or a subset of events of interest). When a continuation is published, it is pushed by the event bus to the predictor. The predictor performs the analysis, and publishes QoS predictions back to the event bus, together with QoS metric bounds inferred by the

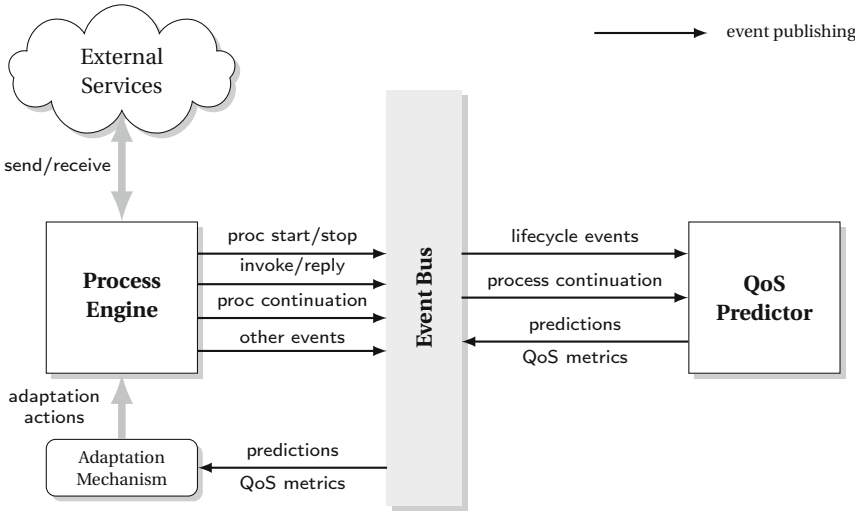


Fig. 3. Architecture of the QoS Analysis Framework

analysis. That information can be accessed by an adaptation mechanism, which can use the published predictions and the QoS metrics to prepare adequate adaptation actions on the orchestration definition, an executing instance, or both. Such adaptation actions may include, among other things, selection of components to minimize the risk of failure, changes in the structure of the process, or intervention on the orchestration data.

3.3 Representing Orchestrations and Their Continuations

In order to estimate how much the remainder of the execution can contribute to a given QoS metrics, we need to have some knowledge about where in the execution we are placed — or, more precisely, what remains to be executed: it is the orchestration activities yet to be executed which need to be taken into account to predict the remainder of the metric value. In our case we represent this still-not-executed part of the orchestration explicitly, in the form of a *continuation*. A continuation [17] is an abstract object (such as a set of data structures or a function) that represents the control state of a computation — i.e., the precise execution point of a program (including the associated data) and whatever remains to be executed.

In our case we are interested in continuations of running instances of orchestrations. A continuation is always implicit in the state of a process engine, even when the chosen programming language does not make it accessible as such: it is determined, for example, by the activity being executed, the representation of the orchestration and the data in the orchestraton. In our approach, we rely on keeping available at all moments an explicit representation of the continuation, inspect its structure (which in general becomes progressively simpler as execution proceeds) and use it to generate constraints which model the conditions under which the execution can meet / not meet the QoS stated in the SLA.

continuation ::=	a .	
a, a_1, a_2 ::=	{*elementary operation*}	(elementary operation)
	$\mid a_1 \,,\, a_2$	(sequence)
	$\mid (\{cond\} \rightarrow a_1 \,;\, a_2)$	(if-then-else)
	$\mid a_1 \wedge a_2$	(and-join)
	$\mid a_1 \vee a_2$	(or-join)
	$\mid$ `while`({*cond*}, *a*)	(while loop)
	$\mid$ `foreach`(*x*, *list*, *a*)	(list comprehension)
	$\mid$ `invoke`(*partner*, *out*, *in*)	(invoke a service)
	$\mid$ `reply`(*out*)	(send a reply)
	$\mid$ `relax`	(do nothing)
	$\mid$ `stop`	(finish)

Fig. 4. Abstract syntax for orchestrations

The (simplified) abstract syntax we will use is shown in Fig. 4. It is based on the concrete syntax used by a prototype orchestration engine which we developed as experimentation base for this paper and that uses Prolog as the language to express branch and loop conditions and elementary operations. A *simple activity* represents a basic unit of work, such as a calculation or assignment. Similarly, *cond* encodes a logical condition that is used for if-then-else branching or `while` loop iteration. List comprehension is simplified using `foreach`. Communication with the environment is done using `invoke` and `reply`. Besides sequences, both parallel OR and AND splits/joins are supported. Most BPMN constructs can be translated straightforwardly. A translation of the example process from Fig. 1 (with some low-level details omitted) is shown on the left of Fig. 5.

The continuation at every point of the execution of Fig. 5 is not explicit in the orchestration representation, but is rather kept by the interpreter which executes it (which we do not have space to describe in detail in this paper). This continuation represents what is left to execute after every computation step, and is updated every time a step is taken. For instance, after taking the *else* branch in the orchestration from Fig. 5 (left), the continuation is a sequence of activities in lines 6-9, 11 and 12.

1	(`invoke(account_svc,` *UserID, AccRec*`)`	$500 \le T_1 \le 800$	(assumption: `account_svc`)
2	∧ `invoke(usage_svc,` *UserID, UsagePatt*`)`	$200 \le T_2 \le 500$	(assumption: `usage_svc`)
3	),	$T_3 = \max(T_1, T_2)$	(∧-join)
4	(*stable*(*UsagePatt*)}	$Cond \in \{0, 1\}, 0 \le T_4 \le 10$	(condition)
5	→ `invoke(reuse_svc,` *AccRec, Profile*`)`	$100 \le T_5 \le 400$	(assumption: `reuse_svc`)
6	; `invoke(gen_svc,` (*AccRec, UsagePatt*), *Profile*`)`,	$200 \le T_6 \le 600$	(assumption: `gen_svc`)
7	`while(` {*unfit*(*Profile*)},	$k \in \mathbf{N}, 0 \le T_7 \le 10$	(`while` condition)
8	`invoke(gen_svc,` (*AccRec, UsagePatt*), *Profile*`)`)	$200 \le T_8 \le 600$	(assumption: `gen_svc`)
9	)	$T_9 = k \times (T_7 + T_8) + T_7$	(`while` duration)
10	),	$(Cond = 1 \wedge T_{10} = T_4 + T_5) \vee (Cond = 0 \wedge T_{10} = T_4 + T_6 + T_9)$	(if)
11	`invoke(conf_svc,` (*UserID, Profile*), _`)`,	$100 \le T_{11} \le 300$	(assumption: `conf_svc`)
12	`stop`.	$T = T_3 + T_{10} + T_{11}$	(total running time)

Fig. 5. Orchestration for Fig. 1 (left) and its associated running time constraints (right)

3.4 Deriving QoS Constraints from Continuations

A *constraint* is a relation that restricts values of variables that, in our case, represent values of QoS metrics associated with the constructs in the orchestration and their basic components. The particular relations which are generated depend both on the QoS metric that is to be captured and on the structure of the continuation. In our approach, after deriving the constraints from the structure of the given continuation, constraint solving techniques (see Section 3.6) are used to infer admissible ranges for variables that lead to either SLA satisfaction or violation.

We require that these constraints lead to a conservative prediction of QoS fulfillment: under the assumption that our knowledge about the QoS characteristics of the basic orchestration components (i.e., atomic activities or external services) is correct,[1] we want that any prediction we make about the conformance of an execution w.r.t. the stated SLA is also correct. In this direction, we make no assumptions on the (in)dependence of behavior of individual components. I.e., if the behavior of two external services seems to be strongly linked (because of e.g. past history), we do not take this apparent correlation into account for the sake of prediction safety. Such information, if available, could be added to try to make predictions more precise: for example, given that some service took less time than expected to answer, we might assume that the same is going to happen to some other service which is apparently historically related. While this seems to help in making predictions more accurate, it also makes them potentially unsafe.

We illustrate constraint derivation with two metrics: *running time* and *availability*. For a continuation consisting of a (complex) activity a representing the remainder of the execution, the total running time of the orchestration is a sum of the elapsed time since the start T_a and the pending time $T(a)$. The total availability is equal to the pending availability $\lambda(a)$, as explained before. We derive $T(a)$ and $\lambda(a)$ structurally, and then constrain them against the SLA limit: $T_{\max}$ for the maximal allowed execution time by and $\lambda_{\max}$ for the negative logarithm of the minimal allowed availability (see Section 3.1). The resulting constraints:

For SLA conformance: $T_a + T(a) \leq T_{\max}$ and $\lambda(a) \leq \lambda_{\max}$.
For SLA violation: $T_a + T(a) > T_{\max}$ and $\lambda(a) > \lambda_{\max}$.

are solved to obtain the (approximate, but safe) ranges for $T(a)$ and $\lambda(a)$, and thus for the total QoS, for the two cases of conformance and violation, respectively.

We generate the above constraints by formulating a constraint for each simple activity contained inside a (usually relating the value of the QoS metric for the activity with its expected bounds) and combining these constraints (using disjunctions and conjunctions according to the structure of a) into a larger constraint which provides bounds for $T(a)$. The right hand side of Fig. 5 shows the set of constraints corresponding to the process on the left. We will now detail how constraints for simple and complex activities are generated.

[1] Note that in reality this knowledge is always inexact and subject to dynamic changes. However, we are putting ourselves in the situation that this knowledge is exact, and we want to ensure that, at least in this optimistic situation, the constraints we generate meet safeness requirements.

Simple activities. For a simple activity a — a simple operation, `relax` or `stop` — and simple operations (in curly braces), the assumption is that they include only elementary constructs and do not entail complex computations. A lower bound for this is always $T(a) \geq 0$, and an upper bound depends on the execution environment (computer clock, CPU, etc.). It is usually on the order of microseconds, and should be experimentally determined for each architecture. In the example we have put some reasonable limits, which do not necessarily reflect a real situation. As for the availability, since no external components are involved, in this case we have $\lambda(a) = 0$.

Sequences. Since we are considering cumulative metrics,[2] the metric values are accumulated for the case of sequences: for sequence $a \equiv a_1, a_2$ we have $T(a) = T(a_1) + T(a_2)$ and $\lambda(a) = \lambda(a_1) + \lambda(a_2)$.

Service invocations. For an activity a that is an `invoke` to an external service, for both the running time $T(a)$ and the availability $\lambda(a)$ the analyzer needs to rely on empirically or analytically derived estimates, which include the local message handling and network delivery. In our approach, we deal with the ranges of *possible* values, rather than with *probable* or *expected* values. That means that in absence of any information, we simply have $T(a) \geq 0$ and $\lambda(a) \geq 0$, but the upper bounds on $T(a)$ and $\lambda(a)$, if known, must be safe, or else the prediction will be too optimistic and fail to detect some cases of possible SLA violations.

Parallel flows. In the case of a parallel flow $a \equiv a_1 \wedge a_2$, $T(a)$ must lay somewhere between $\max(T(a_1), T(a_2))$, when a_1 and a_2 run fully in parallel, and $T(a_1) + T(a_2)$, which is the worst, sequential case of execution. Therefore, it is safe to take

$$\max(T(a_1), T(a_2)) \leq T(a) \leq T(a_1) + T(a_2)$$

as a conservative approximation.

This approximation can however be too cautions and may lead to overly pessimistic estimates. If we have additional information about the semantics of the orchestration language and the implementation of the execution engine, we can refine the estimate for $T(a)$. For instance, if the execution of local activities is single threaded, while external services invocations are ensured to run in parallel, we can use the following scheme. Consider the case where a_1 and a_2 are sequences ending with an `invoke` activity, i.e., $a_1 \equiv a_{11}, a_{12}, \ldots, a_{1k}, a_1^*$ and we call $a_1' \equiv a_{11}, a_{12}, \ldots, a_{1k}$ (respectively for a_2). We will assume that a_1' and a_2' are sequences of activities to be executed locally by a single thread, even if they appear in different branches of the flow, while a_1^* and a_2^* can be executed remotely in parallel. In this case, the total estimated time for the flow is

$$\max\big(T(a_1') + T(a_1^*), T(a_2') + T(a_2^*)\big) \leq T(a) \leq T(a_1') + T(a_2') + \max\big(T(a_1^*), T(a_2^*)\big)$$

If, say, a_1^* is not an external `invoke`, but a_2^* is, then $T(a_1^*)$ is part of $T(a_1')$. If neither a_1^* nor a_2^* are external `invoke`s, then simply $T(a_1^*) = T(a_2^*) = 0$. This structural analysis can of course be easily extended to more than two parallel flows. The running time of an OR-parallel flow can be conservatively approximated using the case of AND-parallelism.

[2] Or those that can be converted into a cumulative (e.g. additive) equivalents.

From the point of view of availability, parallel flows do not affect the total risk of failure, since the total availability depends on availability of all used components, regardless of their order of execution. Therefore, for $a \equiv a_1 \wedge a_2$ or $a \equiv a_1 \vee a_2$, we have $\lambda(a) = \lambda(a_1) + \lambda(a_2)$.

Conditionals. For a conditional $a \equiv (\{cond\} \rightarrow a_1 \,;\, a_2)$, where a_1 is the *then* part and a_2 is the *else* part, the metric depends on how the condition is evaluated. We introduce a Boolean variable b_{cond} to represent the result of the condition evaluation, so that we can state the following disjunctive constraint: either (1) $b_{cond} = 1$ and $T(a) = T(\{cond\}) + T(a_1)$, $\lambda(a) = \lambda(a_1)$, or (2) $b_{cond} = 0$ and $T(a) = T(\{cond\}) + T(a_2)$, $\lambda(a) = \lambda(a_2)$. The value of b_{cond} is generally unknown, but can be constrained to either 0 or 1 as the result of constraint solving. This makes it explicit that either the then or the else part can be taken, but not both.

Loops. In case of a loop a — `while` or `foreach` with body a_1 — we introduce an integer variable $k_a \geq 0$ that stands for the number of loop iterations. Then, we have $T(a) = k_a \times (T(\{cond\}) + T(a_1)) + T(\{cond\})$ and $\lambda(a) = k_a \times \lambda(a_1)$. The actual value of k_a is generally unknown, but its inclusion into the constraints allows us to reason about the maximal or minimal number of loop iterations that lead to SLA compliance or violation.

3.5 Using Computational Cost Functions

To improve the precision of the predictions, the constraint-based predictor is able to use computational cost functions for service orchestrations [13], which, in this case, express lower and upper bounds of the number of loop iterations as a function of the input data to the orchestration. These computation cost functions may be automatically inferred at the start of an orchestration, statically determined at design time, or manually asserted for known cases. The inference of the computation cost functions depends on the semantics of the workflow constructs and the (sub-)language of conditions and elementary operations in which the orchestration is expressed.

If computation cost functions are available, the default constraint for the number of iterations of loop a ($0 \leq k_a$) can be strengthened to $\ell \leq k_a \leq u \wedge 0 \leq k_a$, where ℓ and u are, respectively, lower and upper bounds on the number of iterations, which depend on the actual values of the input data. In the absence of one (or both) of the bounds, the corresponding constraint is simply not generated (as in Fig. 5, right).

3.6 Solving the Constraints

The constraints derived from the orchestration continuation relate the QoS metrics for the entire continuation with those of individual activities, component services, Boolean results of evaluating the conditions, the number of loop iterations, and the limits from the SLA. As such, they represent a constraint satisfaction problem [6] that can be solved for values of the constrained variables, which, in our case, include QoS metrics, Boolean conditions and loop iteration counters. Depending on the type of problem and the particular constraint solver used, solving the CSP may involve several iterations of *constraint propagation* and *problem splitting* [6,1], which are used to reduce the equations in the

original CSP to a series of simpler ones, before attempting to assign to the constrained variables values that satisfy the constraints.

In our case, we use the *interval constraints* (*ic*) solver from the *ECL^{i}PS^e* Constraint Logic Programming (CLP) system [2,5]. The underlying Prolog subsystem of *ECL^{i}PS^e* is used for constructing the constraints from a continuation, handling information on QoS metrics of component services, and reporting the results. The solver handles constrained variables over bound and unbound integer (discrete) and real number (dense) domains. The values of the constrained variables are represented as (possibly unbound) real or integer intervals. Integer variables with bounded domains are handled in a manner similar to finite domain solvers [6]. The solver directly supports disjunctive constraints (which we use for conditionals) and reified (Boolean valued) constraints.

The solver produces results given as bounds on values of the constrained variables, obtained from propagation of arithmetic constraints, or fails if the constraints cannot be satisfied. In our case, as mentioned before, we always solve two CSPs, one modeling SLA conformance and another one modeling SLA violation.

The constraint solver is complete, i.e., it does not discard feasible solutions. Therefore, upon constraint satisfaction, the answer intervals for the variables include all admissible values, and values outside these intervals cannot possibly satisfy the constraints. On the other hand, it may be that some combinations of values inside the answer intervals do not satisfy the constraints. Let us see an example: the constraint $0 \leq T(a_1) + T(a_2) \leq 100$ has as answer $T(a_1) \in [0..100] \wedge T(a_2) \in [0..100]$. This contains all feasible solutions (for example, $T(a_1) = 0 \wedge T(a_2) = 100$) but also combinations of values which do not satisfy the constraints (for example, $T(a_1) = 50 \wedge T(a_2) = 51$). Of course, if the latter values are fed into the constraint solver together with the initial constraint, the constraint solver will determine that the system is unsolvable.

4 Experimental Evaluations

Table 1 shows the results of running our QoS prediction framework applied to the orchestration in Fig. 5 (corresponding to the workflow in Fig. 1) and using execution time as QoS metric. The assumptions on ranges for the invocations of external services are shown at the bottom. These ranges would be updated by the QoS predictor based on the observation of invoke/reply events published by the process engine. Note that we are only concerned with the range of *possible* running times for each component, not the probability distributions within these ranges, and therefore we only need only to adjust the boundaries of the corresponding ranges.

The top part of Table 1 shows the results for the case of an unbound number of `while` loop iterations, which is the default if no additional information is provided. A series of successive assumed running time limits (500, 750, 1 500 and 3 000 ms) was considered, and both the SLA compliance (*success*) and *violation* results are shown. The meaning of the rest of the rows are as follows:

duration shows the predicted running time ranges for the orchestration in ms.

cond(if) is a Boolean value showing the possible evaluations of the condition in the conditional (1 for the "then" branch and 0 for the "else" branch).

iter(while) shows the range of possible iteration counts in the `while` loop (corresponding to the repetition after testing the condition in the "else" branch).

Table 1. Sample QoS prediction results

Case 1: Unconstrained iterations									
		Successive running time SLA ranges							
		0 ms .. 500 ms		500 ms .. 750 ms		750 ms .. 1 500 ms		1 500 ms .. 3 000 ms	
Variable	Metrics	*success*	*violation*	*success*	*violation*	*success*	*violation*	*success*	*violation*
duration	ms	—	600 .. +∞	600 .. 750	750 .. +∞	750 .. 1 500	1 500 .. +∞	1 500 .. 3 000	3 000 .. +∞
cond(if)	bool	—	0 .. 1	1	0 .. 1	0 .. 1	0	0	0
iter(while)	nat	—	0 .. +∞	—	0 .. +∞	0 .. +∞	0 .. +∞	0 .. 11	3 .. +∞
E.C.D.T.	ms	—	0	500	450	500	1 200	700	2 700
% E.C.D.T.		—	0%	66%	60%	33%	80%	23%	90%
Lead	ms	—	500	250	300	1 000	300	2 300	300

Case 2: Between 1 and 10 iterations									
		Successive running time SLA ranges							
		0 ms .. 500 ms		500 ms .. 750 ms		750 ms .. 1 500 ms		1 500 ms .. 3 000 ms	
Variable	Metrics	*success*	*violation*	*success*	*violation*	*success*	*violation*	*success*	*violation*
duration	ms	—	600 .. 7 820	600 .. 750	750 .. 7 820	750 .. 1 500	1 500 .. 7 820	1 500 .. 3 000	3 000 .. 7 820
cond(if)	bool	—	0 .. 1	1	0 .. 1	0 .. 1	0	0	0
iter(while)	nat	—	1 .. 10	—	1 .. 10	1 .. 10	1 .. 10	1 .. 10	3 .. 10
E.C.D.T.	ms	—	0	500	250	500	1 000	900	2 500
% E.C.D.T.		—	0%	66%	33%	33%	66%	30%	83%
Lead	ms	—	500	250	500	1 000	500	2 100	500

	Component running time assumptions					
	local op.	account_svc	usage_svc	reuse_svc	gen_svc	conf_svc
Running time (ms)	0 ms .. 10 ms	500 ms .. 800 ms	200 ms .. 500 ms	100 ms .. 400 ms	200 ms .. 600 ms	100 ms .. 300 ms

E.C.D.T. *earliest certain detection times*: the earliest time at which a certain violation or success can be detected.

% E.C.D.T. percentage of the total (maximum) execution time which elapsed up to the E.C.D.T.

lead time between E.C.D.T. and the closest moment in which the orchestration can finish (i.e., the shortest time span to react in the worst case).

The results show that the lowest limit of 500 ms could not be met under the initial assumptions regarding execution times for atomic activities and external services. The 750 ms limit can be met, if the conditional evaluates to 1, meaning that the `while` loop is avoided. The 1 500 ms limit can be met in both cases of the conditional, but can be violated only for the case of taking the "else" branch. Finally, for the range of running times between 1 500 ms and 3 000 ms, the prediction shows that, under the given assumptions, the only possible situation for both compliance and violation is taking the "else" branch, with the number of iterations in the range 0 .. 11 and 3 .. +∞, respectively. Note that for the latter limit, between 0 and 2 iterations guarantees compliance, and more than 11 iterations guarantees violation of the limit. An adaptation mechanism can, use these predictions to prepare and trigger adaptation actions that may prevent, minimize, or compensate for possible SLA violations ahead of time.

The earliest time at which a success or violation can be predicted depend on the particular execution. Let us look at an example: in Table 1, Case 1, columns "750 ms .. 1500 ms", successes have been detected at 500 ms and SLA violations at 1200 ms. The reason that successes have been detected before violations is that these correspond to different executions: in the case of violation, the "else" branch (with the loop) has

been taken, it is detected that there will be a violation after some iterations. On the other hand, if the "then" branch is taken, certainty of success is immediately detected, as there are no loops to be taken. With this interpretation in mind, the constraint-based predictor is able to detect SLA violation with certainty up to between 300 and 500 ms in advance, while SLA conformance can be detected as early as after 500 or 700 ms of running time. In relative terms, SLA conformance has been detected in the experiments when only between a 23% and a 66% of the maximum execution time has elapsed, and SLA violations have been detected in some cases when only a 60% of the execution has elapsed.

The middle part of Table 1 shows a hypothetical case where, based on input data and computational cost functions, the predictor is able to infer that the actual number of loop iterations, in case the "else" branch is taken, must fall between 1 and 10. The results follow the same pattern as in the first case, but this time the predictor is able to infer that the duration of the orchestration under the assumptions must fall between 600 and 7 820 ms. This inferred running time range for the orchestration can be used by other parts of the runtime system (including predictors themselves) to update their QoS metrics assumptions on the deployed components. Note that the guarantee of at least one loop iteration increases the lead for the earliest certain detection of violations to 500 ms.

The average net time for performing one running time limit compliance/ violation prediction depicted in Table 1 (not counting the time for sending and receiving data over the network), based on the average from 10 000 executions, was 0.574 ms on a small end-user non-dedicated machine.[3]

5 Implementation Notes

We have tested the approach using a prototype implementation of the architecture from Fig. 3, which includes the process engine, the QoS predictors, and the event bus, organized as a distributed and scalable system of components that communicate using reliable messaging. The tests included deployments on Linux and Mac OS X 32 and 64 bit platforms.

In our running prototype, the QoS predictors are implemented in ECL^iPS^e Constraint Logic Programming system, while the process engine (that executes orchestrations) is implemented in Ciao Prolog [9]. Both Prolog dialects support a variety of constraint logic programming techniques, but have, at the moment, slightly different orientation and strong points. ECL^iPS^e provides very robust, industrial-scale constraint solvers which can easily handle very complex problems involving thousands of constraints and variables, while Ciao is a flexible multi-paradigm programming environment with sophisticated support for concurrency. Fortunately the fact that they are both Prolog-based systems greatly facilitates interfacing and putting together the required architecture.

In our prototype, the language in Fig. 4 is used to define service orchestrations and to maintain instance control state throughout execution, so that there is no additional

[3] The tests were run on a 32-bit 2GHz Intel Core Duo notebook with 2GB of RAM, running Mac OS X 10.6.7 and ECL^iPS^e version 6.0_167.

overhead in communicating continuations to QoS predictors, other than message transfer times. Also, any adaptation that changes the orchestration structure for a running instance can be simply implemented by replacing one continuation with another.

The messaging subsystem is implemented using ZeroMQ [11], which provides fast and reliable multi-part binary message exchange primitives on top of TCP networking and IPC subsystems, including request-reply, push-pull and publish-subscribe patterns. We have developed Prolog (Ciao and ECL^iPS^e) bindings to ZeroMQ with data (term) serialization that provide transparent higher-level data exchange primitives.

6 Conclusions

We have devised and implemented a method which makes it possible to predict possible situations of SLA conformance and violation, and to obtain information on the internal parameters of the orchestration (branch conditions, loop iterations) that may occur in these situations. The method is based on modeling QoS metrics of a service orchestration using constraints, based on assumptions on the behavior of the orchestration components. That analysis can, in principle, be applied at each step in an orchestration based on the current continuation. This allows periodic or continuous updating of the predicted bounds for QoS metrics for the orchestration and therefore a continuous assessing of conformance to SLA, which can be useful for proactive adaptation and self-healing. This approach can be combined with automatically inferred computational cost functions for service orchestrations, which can express the bounds of internal parameters (such as loop iterations) as functions of input data given to the orchestration instance, to provide a higher level of prediction precision. We have implemented the method in a prototype and reported some efficiency results.

Our future work will concentrate on making the implementation of all elements of the QoS prediction architecture laid out in this paper more complete and robust, including the process engine, beyond the prototype stage. We also plan to add support for different execution engines, targeting specifically those that have well-defined interfaces for event-listening plugins or can be adapted accordingly (e.g. because the implementation is open-source).

References

1. Apt, K.R.: Principles of Constraint Programming. Cambridge University Press (2003)
2. Apt, K.R., Wallace, M.G.: Constraint Logic Programming Using ECLIPSE. Cambridge University Press (2007)
3. Cardoso, J.: About the Data-Flow Complexity of Web Processes. In: 6th International Workshop on Business Process Modeling, Development, and Support: Business Processes and Support Systems: Design for Flexibility, pp. 67–74 (2005)
4. 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)
5. Cisco Systems. ECLIPSE User Manual (2006)
6. Dechter, R.: Constraint Processing. Morgan Kauffman Publishers (2003)

7. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering 15, 313–341 (2008), doi:10.1007/s10515-008-0032-x
8. Dranidis, D., Metzger, A., Kourtesis, D.: Enabling Proactive Adaptation through Just-in-Time Testing of Conversational Services. In: Di Nitto, E., Yahyapour, R. (eds.) ServiceWave 2010. LNCS, vol. 6481, pp. 63–75. Springer, Heidelberg (2010)
9. Hermenegildo, M.V., Bueno, F., Carro, M., López, P., Mera, E., Morales, J.F., Puebla, G.: An Overview of Ciao and its Design Philosophy. Theory and Practice of Logic Programming (2012), http://arxiv.org/abs/1102.5497
10. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A framework for Proactive Self-Adaptation of Service-Based Applications Based on Online Testing. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 122–133. Springer, Heidelberg (2008)
11. iMatix Corporation. 0MQ - The Reference Manual, version 2.1 (June 2011)
12. Ivanović, D., Carro, M., Hermenegildo, M.: An Initial Proposal for Data-Aware Resource Analysis of Orchestrations with Applications to Predictive Monitoring. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 414–424. Springer, Heidelberg (2010)
13. Ivanović, D., Carro, M., Hermenegildo, M.: Towards Data-Aware QoS-Driven Adaptation for Service Orchestrations. In: Proceedings of the 2010 IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, July 5-10. IEEE (2010)
14. Jordan, D., et al.: Web Services Business Process Execution Language Version 2.0. Technical report, IBM, Microsoft, et al (2007)
15. Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., Leymann, F.: Runtime Prediction of Service Level Agreement Violations for Composite Services. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 176–186. Springer, Heidelberg (2010)
16. Object Management Group. Business Process Modeling Notation (BPMN), Version 1.2 (January 2009)
17. Reynolds, J.C.: The discoveries of continuations. LISP and Symbolic Computation Journal 6, 233–247 (1993)
18. Stein, S., Payne, T.R., Jennings, N.R.: Robust execution of service workflows using redundancy and advance reservations. IEEE T. Services Computing 4(2), 125–139 (2011)
19. Tselentis, G., Dominigue, J., Galis, A., Gavras, A., Hausheer, D.: Towards the Future Internet: A European Research Perspective. IOS Press, Amsterdam (2009)
20. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
21. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: The Role of Business Processes in Service Oriented Architectures. Dagstuhl Seminar Proceedings, vol. 06291 (2006)
22. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
23. Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., Leymann, F.: Monitoring and analyzing influential factors of business process performance. In: EDOC, pp. 141–150. IEEE Computer Society (2009)

Optimizing Decisions in Web Services Orchestrations

Ajay Kattepur[1], Albert Benveniste[1], and Claude Jard[2]

[1] IRISA/INRIA, Campus Universitaire de Beaulieu, Rennes, France
[2] ENS Cachan, IRISA, Université Européenne de Bretagne, Bruz, France

Abstract. Web services orchestrations conventionally employ exhaustive comparison of runtime quality of service (QoS) metrics for decision making. The ability to incorporate more complex mathematical packages are needed, especially in case of workflows for resource allocation and queuing systems. By modeling such optimization routines as service calls within orchestration specifications, techniques such as linear programming can be conveniently invoked by non-specialist workflow designers. Leveraging on previously developed QoS theory, we propose the use of a high-level flexible query procedure for embedding optimizations in languages such as Orc. The *Optima* site provides an extension to the sorting and pruning operations currently employed in Orc. Further, the lack of an objective technique for consolidating QoS metrics is a problem in identifying suitable cost functions. We employ the *analytical hierarchy process (AHP)* to generate a total ordering of QoS metrics across various domains. With constructs for ensuring *consistency* over subjective judgements, the AHP provides a suitable technique for producing objective cost functions. Using the *Dell Supply Chain* example, we demonstrate the feasibility of decision making through optimization routines, specially when the control flow is QoS dependent.

Keywords: Web Services, QoS, Optimization, Orc, AHP.

1 Introduction

A *composite* web service is an application whose implementation calls other self-contained *atomic* services. A composite web service *orchestration* specifies the interaction, management and coordination between these atomic services. Such a composite service can take decisions to invoke or pass parameters to atomic services depending on returned data and quality of service (QoS) metrics. Traditional orchestrations make use of simple comparisons of returned values from atomic services for decision making purposes. While such comparisons are plausible in small orchestrations, involved operations such as multi-criteria decisions from a directory of hundreds of distributed services would require optimizations strategies. With QoS metrics modeled as random variables [1], the use of probabilistic contracts for service level agreements (SLAs) [2] becomes mandatory. Optimizing these random variables for decision making is a natural extension of the probabilistic nature of both composition as well as contracts.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 77–91, 2011.

As switching between technologies while developing workflows is detrimental, integration of optimization techniques as part of the specifications of a service orchestration or choreography is required. We show that optimization of QoS metrics can be formulated within concurrent programming languages like Orc [11]. Employing specialized *sites* that perform optimization routines, alternatives to conventional sorting and searching techniques may be incorporated within workflow specifications.

As the designers of such workflows are assumed to be non-specialists in optimization modeling, we propose techniques for formulating complex queries through simple user judgements / constraints. This will relieve the dependency on domain-specific and involved concepts such as queuing and process management theory in order to generate realistic cost functions. Weighing parameters effectively is done by employing the analytic hierarchy process (AHP) [4]. It provides a simple approach for retaining consistency of subjective evaluations of QoS metrics across different domains.

To prevent deadlock in an orchestration where there are intricate links between parameters, it is essential that optimal settings are employed. This is demonstrated in the QoS dependent choreography of the *Dell* supply chain example [9]. By modeling this choreography as a *linear programming* problem, we demonstrate the efficacy of our technique to ensure contractual obligations with shared resources. Due to the tractable nature of AHP, cost functions can be generated to set suitable resupply batch sizes for varying demand rates. This exemplifies clearly a situation where the control flow is dependent on optimal setting of parameters.

The paper is organized as follows: Section 2 presents background material required for understanding the rest of the paper. This includes optimization models, Orc language for orchestrations, the analytic hierarchy process and QoS aspects of web services. The methodology proposed in this paper is outlined in Section 3 with emphasis on formulating optimizations in web services. Section 4 elucidates the Dell logistics example as an optimization of QoS metrics. Extending this notion to general orchestration problems, in Section 5, we formulate a general site that provides such optimization routines in the Orc context. Results for optimization runs of both examples are presented in Section 6. This is followed by related work and conclusions in Sections 7 and 8, respectively.

2 Fundamentals

2.1 Optimization Models

Optimization problems may be formulated as [6]:

$$\begin{aligned} &\textbf{min} \quad f_0(a,x) \\ &s.t. \quad f_i(a,x) \leq 0, \quad i = 1,\ldots,m \end{aligned} \tag{1}$$

where f_0 is the objective function, f_i are the set of constraint functions dependent on the input vector $x = (x_1, x_2, \ldots, x_N)^T$ and model parameters

$a = (a_1, a_2, \ldots, a_M)^T$. This can be solved in a variety of linear, non-linear, stochastic and exhaustive search techniques. Approximate bounds to reduce stochastic uncertainty can also be used. This can lead to three categories of minimization problems.

– Minimization of primary expected costs subject to secondary cost constraints.

$$\begin{aligned} &\textbf{min} \;\; F_0(a,x) \\ &s.t. \;\; F_i(a,x) \leq F_i^{max}, \quad i = 1, \ldots, m \end{aligned} \tag{2}$$

where $F_0(a,x)$ is the primary goal, $F_i(a,x)$ are secondary constraints with worst-case bounds represented by F_i^{max}.

– Minimization of the cost function with positive weights $k_0, k_1, \ldots, k_m$.

$$\textbf{min} \;\; \sum_{i=0}^{m} k_i F_i(a,x) \tag{3}$$

– Minimization of the maximum weighted expected costs.

$$\textbf{min} \;\; \max_{0 \leq i \leq m} k_i F_i(a,x) \tag{4}$$

Such formulations of cost functions with constraints can be applied to a variety of decisions within the web services framework.

2.2 QoS in Web Services

Available literature on industry standards in QoS [3] provide a family of QoS metrics that are needed to specify SLAs. These can be subsumed into the following four general QoS observations [1]:

1. $\delta \in \mathbb{R}_+$ is the service latency. When represented as a distribution, this can subsume other metrics such as availability and reliability of the service.
2. $\$ \in \mathbb{R}_+$ is the per invocation service cost.
3. $\zeta \in \mathbb{D}_\zeta$ is the output data quality. This can represent other metrics such as data security level and non-repudiation of private data over a scale of values.
4. $\lambda \in \mathbb{R}_+$ is the inter-query interval, equivalent to considering the query rate for a service. Performance of the service will depend on negotiations with the amount of queries that can be made in a given time interval.

Along with QoS, the web service performs its task and returns some functional data $\rho \in \mathbb{D}_\rho$ as the output. The tuple of *(Data value, QoS value)* is used for the decision process within orchestrations. The implementation of Orc allows such typing to be specified for input and output parameters, which can be extended to QoS typing for orchestrations.

[1] Aspects such as scalability, interoperability and robustness are not dealt with as they are specific to the supplier side operation (not necessarily part of SLAs).

For comparing metrics with differing scales and units of measurement, a normalization or scaling technique is needed. As developed in [5] [16], the normalization of QoS values $\mathbf{q}_i$ in a domain $\mathbb{D}_Q$ can be performed using a scaling function, prior to optimization. The scaling function $S(\mathbf{q}_i)$ in eq. (5) ensures that the range of QoS values falls within $[0, 1]$ for equivalent comparison. Essentially, this prevents larger scale values in domains (eg. latency) nullifying optimal selection in smaller valued domains (eg. boolean valued availability).

$$S(\mathbf{q}_i) = \frac{\mathbf{q}_i - \mathbf{q}_{min}}{\mathbf{q}_{max} - \mathbf{q}_{min}} \tag{5}$$

where $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the minima and maxima of the (available) distributions of these QoS domains. A generic range of values for metrics such as data quality or service invocation costs may be reduced to a comparable scales via this method. An example of scaling measured values is shown in Table 1. The measured values are scaled to the range $[0, 1]$ with the scaling invariant to changes in measurement units of, for instance, the response time δ.

Table 1. Scaling QoS metrics across domains to the range [0,1]

Metric	**Measurement $\mathbf{q}_i$**	**Scaled Value $S(\mathbf{q}_i)$**
$\delta(hours)$	(0.017, 0.001, 0.0095, 0.01)	(1, 0, 0.53125, 0.5625)
$\delta(seconds)$	(61.2, 3.6, 34.2, 36)	(1, 0, 0.53125, 0.5625)
$\$(Euros)$	(9.5, 3.4, 6.8, 12)	(0.7093, 0, 0.3953, 1)
$\zeta([1, 10])$	(6, 1, 3, 8)	(0.7143, 0, 0.2857, 1)

2.3 Analytic Hierarchy Process

Multiple dimensions in web services' QoS are only partially ordered, with comparisons between domains not possible. In order to use optimization routines, a total ordering of these domains is mandatory. To reconcile this, the analytic hierarchy process (AHP) can be used. Introduced by [4], AHP can be used to objectify subjective evaluations of multi-criteria decisions, which essentially develops tradeoffs between domains. In order to briefly explain the AHP, we make use of an example.

Consider the pairwise assignment of relative ranks for QoS metrics as defined by a user. It is a matrix that defines the relative change between dependent QoS metrics δ, \$, ζ, λ and ρ. For simplicity, all parameters are classified as the same hierarchical level with values assigned using the relative comparison shown in Fig. 1. This in turn will produce a matrix $\mathbf{W} = (\mathbf{w}_{ij})$ as shown in eq. 6 with the subjective pairwise comparison of criterion.

$$\mathbf{W} = \begin{array}{c} \\ \delta \\ \$ \\ \zeta \\ \lambda \\ \rho \end{array} \begin{array}{c} \begin{array}{ccccc} \delta & \$ & \zeta & \lambda & \rho \end{array} \\ \begin{pmatrix} 1 & 1 & 5 & 3 & 5 \\ 1 & 1 & 5 & 3 & 5 \\ 1/5 & 1/5 & 1 & 1 & 2 \\ 1/3 & 1/3 & 1 & 1 & 3 \\ 1/5 & 1/5 & 1/2 & 1/3 & 1 \end{pmatrix} \end{array} \tag{6}$$

The Fundamental Scale for Pairwise Comparisons		
Intensity of Importance	**Definition**	**Explanation**
1	Equal importance	Two elements contribute equally to the objective
3	Moderate importance	Experience and judgment slightly favor one element over another
5	Strong importance	Experience and judgment strongly favor one element over another
7	Very strong importance	One element is favored very strongly over another; its dominance is demonstrated in practice
9	Extreme importance	The evidence favoring one element over another is of the highest possible order of affirmation
Intensities of 2, 4, 6, and 8 can be used to express intermediate values. Intensities 1.1, 1.2, 1.3, etc. can be used for elements that are very close in importance.		

Fig. 1. Comparison Scale for AHP [4]

The principal eigenvector of the *positive reciprocal matrix* $\mathbf{W}$ provides the relative rankings of the parameters. As the principal diagonal of the matrix $\mathbf{W}$ consists of real values, the principal eigenvector (and corresponding highest eigenvalue) are also real valued.

Theorem 1. ***Perron Frobenius Theorem****: For a given positive matrix* $\mathbf{W}$*, the only positive vector* v *and only positive constant* c *that satisfy* $\mathbf{W}v = cv$*, is a vector* v *that is a positive multiple of the principle eigenvector of* $\mathbf{W}$ *and the only such* c *is the principal eigenvalue of* $\mathbf{W}$*.*

This eigenvector may be normalized to provide the *priority vector* for the QoS metrics. This will generate a weighted cost function for minimization, which is superior to cost function weights obtained by least squares [8]. For the example above, the linear cost function after generating the normalized weight vector is shown in eq. (7) with scaling of values done previously according to eq. (5).

$$\mathbf{Z} = 0.3625\delta + 0.3625\$ + 0.0935\zeta + 0.1237\lambda + 0.0579\rho \tag{7}$$

A unique feature of the AHP is its ability to estimate consistency in the subjective evaluation of criteria.

Definition 1. *A* $n \times n$ *positive reciprocal matrix* $\mathbf{W} = (\mathbf{w}_{ij})$ *is a* ***Consistent Matrix****, if the highest eigenvalue* c_{max} *equals* n*. This is equivalent to* $\mathbf{w}_{ij} = v_i/v_j$*, where the eigenvector* v *corresponds to eigenvalue* c_{max}*. Since small changes in* $\mathbf{w}_{ij}$ *imply changes in* c_{max}*, the deviation from* n *is a deviation from consistency given by* $(c_{max} - n)/(n - 1)$ *which is called the* ***consistency index (CI)****.*

This technique evaluates the perturbation in the highest eigenvalue due to changes in subjective evaluation of metrics in $\mathbf{W}$. The values of the consistency index are used to generate a consistency ratio (CR), that is used to determine the consistency of the comparison. The consistency ratio must be ≤ 0.1, indicating deviations from subjective evaluations are less than an order of magnitude [4]. For the example above, the highest eigenvalue has the value 5.122, producing a CI = 0.0280 and a CR = 0.0252, which is within the specified limits. Techniques outlined in [8] provide steps and tools to improve consistency in the weight matrix.

3 Methodology

The following steps are used to solve optimization problems in web services:

1. **Scaling Inputs:** Obtain the pair of QoS domains and vector of values $(\mathbb{D}_Q, \mathbf{q})$ required for evaluation of the orchestration. For each domain $\mathbb{D}_Q$, scale the values $\mathbf{q}$ to the range $[0, 1]$ as specified in eq. (5).
2. **Consistent Judgements:** Extract the comparative judgement matrix $\mathbf{W} = (\mathbf{w}_{ij})$ from the user. From this, obtain the maximum eigenvalue c_{max} and the corresponding normalized eigenvector υ. If this judgement matrix is not *consistent*, examine the judgment for an entry $\mathbf{w}_{ij}$ for which $\mathbf{w}_{ij}\upsilon_j/\upsilon_i$ is the largest, and see if this entry can reasonably be made smaller. Such a change of $\mathbf{w}_{ij}$ also produces a new comparison matrix with a smaller eigenvalue, resulting in a possibly consistent matrix [8]. This process may be performed either manually or automatically through iterative perturbations of $\mathbf{W}$ until consistency is achieved. Once a consistent matrix is obtained, the objective function $\mathbf{Z}$ to be minimized with linear weights υ and $(\mathbb{D}_Q, \mathbf{q})$ values may be generated.
3. **Constraints:** The scaled optimization constraints $\mathbf{C}$ in the form $(\mathbb{D}_Q, \preceq, K_Q)$, where $\mathbb{D}_Q$ is a QoS domain, $\preceq$ is a specified partial order and K_Q is the threshold value (constant or distribution quantiles), may also be set by the user.
4. **Optimization:** With a selected constraint satisfying solver with inputs $(\mathbf{Z}, \mathbf{C})$, optimization is performed. If constraints $\sum_{i=1}^{N} x_i = 1, x_i \in \{0, 1\}$ for model variables x exists in $\mathbf{C}$, it implies an integer programming problem (eg. selecting a single site). In the absence of such a constraint, the solver employs a conventional linear programming approach (eg. finding an optimal setting from a continuous distribution).

The only inputs required from the user are the judgement matrix and constraints over QoS domains. This methodology is intended to enhance previous theory [7] with optimization routines to compare returned QoS token values.

4 Formulating Optimization Problems

In this section, we investigate the Dell supply chain, a choreography of *Dell Plant* and *Supply* orchestrations with a shared *Revolver* resource. This exemplifies the optimization of setting inventory levels to ensure efficient control flow and preventing contractual deviations.

In the *Dell* supply chain [9], QoS metrics are functional in nature, with slight changes in optimal settings sending the supply chain to a dead state. The *Dell* application is a system that processes orders from customers interacting with the Dell webstore. According to [9], this consists of the following prominent entities:

– *Dell Plant* - Receives the orders from the Dell webstore and is responsible for the assembly of the components. For this they interact with the *Revolvers* to procure the required items.

- *Revolvers* - Warehouses belonging to Dell which are stocked by the suppliers. Though Dell owns the revolvers, the inventory is owned and managed by the *Suppliers* to meet the demands of the *Dell Plant*.
- *Suppliers* - They produce the components that are sent to the revolvers at Dell. Periodic polling of the *Revolvers* ensures estimates of inventory levels and their decrements.

The interaction between the *Dell Plant*, *Revolvers* and the *Suppliers* may be summarized in Fig. 2. The requests made by the plant for certain items will be favorably replied to if the revolvers have enough stock. This stocking of the revolvers is done independently by the suppliers. The suppliers periodically poll (withdraw inventory levels) from the revolvers to estimate the stock level. In such a case, a contract can be made on the levels of stock that must be maintained in the revolver. The customer side agreement limits the throughput rate. The supplier side agreement ensures constant refueling of inventory levels, which in turn ensures that the delay time for the customer is minimized. Thus, it represents a *choreography* comprising two plant-side and supplier-side orchestrations interacting via the revolver as a shared resource.

The critical aspect in the Dell choreography is efficient management of revolver levels. As discussed in [9], for the efficient working of the supply chain, the interaction between the Dell Plant and the Supply-side workflows should be taken into account. This will involve optimizing critical QoS metrics listed in Table 2. They are also presented informally in Fig. 2.

Table 2. QoS Metrics for the Dell Supply Chain

t	Unit of time with $t \in 1, 2, ...T$ hours
λ_t	Number of queries per unit time that the plant requests the revolver
δ_{cust}	Waiting time for the plant
μ_t	Stock level for an item in the revolver at time t
μ_c	Critical stock levels of the item in the revolver
μ_{max}	Maximum stock level allowed in the revolver
ρ	Inventory polling period of the supplier
β	Size of the refueling batch from the supplier
δ_{sup}	Delay period for refueling the revolver
$\upsilon_{\mu_c}, ...\upsilon_\beta$	Normalized eigenvector from the consistent AHP matrix

For the plant-side behavior, the demand λ_t reduces the current revolver level ($\mu_t = \mu_{t-1} - \lambda_t$). Constant polling at a rate ρ ensures the re-fueling of revolver inventory within a supply delay δ_{sup}. When the value of the revolver token drops below a critical level μ_c, the supplier begins the process of refueling the inventory. The refueling batch size β is governed by the maximal capacity of the revolver μ_{max}. Optimal setting of these parameters minimizes the customer waiting time δ_{cust}. If the supplier does not refuel on time, the choreography sets

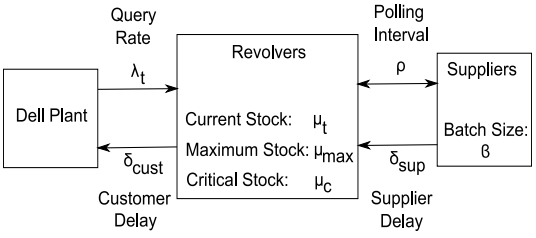


Fig. 2. QoS interactions in the Dell supply chain

into deadlock with the plant waiting for (possible) restocking. A deadlock occurs when a choreography reaches a state that (1) is not final and (2) can not be left without violating the message ordering of the choreography.

Considering estimated distributions of customer demand and refueling delays, effective settings for supplies may be set. Using AHP weights, the optimization procedure is given as a linear programming problem (without any integer constraints). More classical logistics cost functions [10] can also be applied to similar problems.

$$\textbf{minimize } \mathbf{Z} = \upsilon_{\mu_c}\mu_c + \upsilon_{\mu_{max}}\mu_{max} + \upsilon_\beta\beta \tag{8}$$

Subject to the following user-specified constraints:

$$0 \leq \mu_c \leq \mu_{max} \tag{9}$$

$$0 \leq \beta \leq \mu_{max} \tag{10}$$

$$\mu_{max} - \mu_c + (\lambda_t \times \delta_{sup}) = \beta \tag{11}$$

$$0 \leq \mu_{max} \leq K \times \lambda_t \tag{12}$$

Constraints in eqs. (9) and (10) limit the revolver critical level μ_c and the supplier batch size β to be less than the maximal revolver capacity μ_{max}. The constraint in eq. (11) essentially controls the revolver batch size, dependent on the critical / maximum level in the revolver and the plant query rate λ_t. Estimates of λ_t are provided to the supplier during the polling period through measured decrements in the revolver levels. The supplied batch β also incorporates the decrement in inventory since the critical level was detected, and the delay in restocking δ_{sup}. Finally, the constraint in eq. (12) prevents overstocking of items in the revolver by limiting the capacity to be proportional to the demand. Optimal setting of these parameters is tested by the constant demand for products λ_t which must be delived while minimizing the customer delay δ_{cust}. Essentially, these constraints ensure the revolver level does not fall to zero, which would mean rejection or long delays in orders (deadlock in the choreography).

5 Optimization Routines in Orc

While the previous sections demonstrate the utility of optimization techniques when applied to decisions in workflows, it is imperative to provide a convenient

technique to embed such mathematical packages within orchestrations. Extending Orc [11] with a suitable interface will enable smooth integration of optimization libraries for the utility of workflow designers. In this section, we provide a high-level specification of optimizing QoS metrics within Orc.

Orc [11] serves as a simple yet powerful concurrent programming language to describe web services orchestrations. The fundamental declaration used in the Orc language is a *site*. The type of a *site* is itself treated like a service - it is passed the types of its arguments, and responds with a return type for those arguments. An Orc *expression* represents an execution and may call external services to publish some number of values (possibly zero).

Orc has the following combinators that are used on various examples as seen in [11]. The *Parallel* combinator $X|Y$, where X and Y are Orc expressions, runs by executing X and Y concurrently; returns from X and Y are interleaved. Whenever X or Y communicates with a service or publishes a value, $X|Y$ does so as well. The execution of the *Sequential* combinator $X >t> Y$ starts by executing X. Sequential operators may also be written compactly as $X \gg Y$. Values published by copies of Y are published by the whole expression, but the values published by X are not published by the whole expression; they are consumed by the variable binding. If there is no response from either of the sites, the expression does not terminate. The *Pruning* combinator, written $X <t< Y$, allows us to block a computation waiting for a result, or terminate a computation. The execution of $X <t< Y$ starts by executing X and Y in parallel. Whenever X publishes a value, that value is published by the entire execution. When Y publishes its first value, that value is bound to t in X, with the execution of Y immediately terminated. The *Otherwise* combinator, written $X; Y$ has the following execution. First, X is executed. If X completes, and has not published any values, then Y executes. If X did publish one or more values, then Y is ignored. The publications of $X; Y$ are those of X if X publishes, or those of Y otherwise.

Consider the following two Orc expressions - one of which chooses the fastest responding service; another produces the lowest costing service value:

```
def minLatencySite() = s <s< (Site_1 |...| Site_N)
def minCostSite() = (Site_1,...,Site_N) >(c_1,...c_N)> minimum([c_1,...c_N])
```

Combining these expressions in Orc can currently be done with priorities, that is, choosing a site with lower cost over one with lower latency, or vice versa. This can be detrimental in typical situations involving more than one QoS metric. Finding an optimal service that provides a "middle path" solution from various domains can be beneficial. Such an expression in Orc with weights `w`:

```
def optimalSite() = (Site_1,...Site_N) >((d_1,c_1),...,(d_N,c_N))>
minimum([ w*d_1 + (1-w)*c_1 ,..., w*d_N + (1-w)*c_N ])
```

A drawback of the above formulation is that exhaustive comparison of metrics are still used. In order to overcome this, the selection of services can be formulated as an optimization problem. Such a formulation is useful in a variety of

orchestrations where the control flow is dependent on optimal resolution of competition between services. A point to note here is that the fastest service cannot be given priority as the orchestration waits for responses from all services (until timeout).

In [7], the "best" operator provides a general function for comparison of a variety of metrics. We propose an extension of this to satisfy more complex queries, when "enumerate and evaluate" is both ineffective and slow. Moreover, there are no standard sets of QoS parameters that are declared in general for all orchestrations - which draws the need for a framework for totally ordered metrics.

5.1 QOrc: Upgrading Orc for QoS Management

A proposal is making use of a QoS enhanced orchestration declaration called *QOrc*. Every invoked service responds with not only the desired output data but also with a set of QoS values. So, selection of a service can entail complex queries dependent on a variety of parameters for optimization. Consider a *site* `Optima` that may be invoked during an orchestration run. This site has input tuple `(QoS, AHPWeight, Constraint, Routine)` where `QoS` is the set of QoS domains with a list of corresponding values, `AHPWeight` is a set of (normalized) weights dependent on AHP criterion, `Constraint` are the (normalized) constraint functions and `Routine` is the optimization protocol to be employed. The user can specify the routine to be either binary integer or linear programming depending on the problem. A typical implementation in Orc is:

```
type Latency = Number
type Cost = Number

val l = Buffer()
val c = Buffer()
val QoS = ((Latency,l),  (Cost,c))
val AHPWeight = (0.3,0.7)
val Constraint = ((Latency,(<),0.5), (Cost,(<),0.8))
val Routine = ''binary integer''
```

For example, the following orchestration describes optimal selection from three generic services, while using the `Optima` site.

```
(Site1(), Site2(), Site3()) >((l1,c1), (l2,c2), (l3,c3))>
Optima(((Latency,[l1,l2,l3]), (Cost,[c1,c2,c3])), (0.3,0.7),
((Latency,(<),0.5), (Cost,(<),0.8)), ''binary integer'')
```

A library of optimization routines available as services allow complex decision making in orchestrations, even to non-specialized users of such tools. As described in the COIN-OR (COmputational INfrastructure for Operations Research) project [12] [13], a host of solvers and APIs are provided for integrating optimization. A variety of input formats such as AMPL (A Modeling Language

for Mathematical Programming), MPS (Mathematical Programming System) and GAMS (General Algebraic Modeling System) may be used to specify the problems.

5.2 Interfacing QOrc to Optimization Services

We use the example of the LP file format used for the open-source *lpsolve*[2] solver to demonstrate the compatibility of an input from Orc. The input syntax of the LP format uses an *Objective Function* with associated *Constraints* and variable *Declarations*. With the inputs provided from the Orc `Optima` site, the optimization problem can be conveniently formulated to the binary integer problem. Formulation of linear or more complex quadratic problems can follow this procedure to conceal intricacies of mathematical packages from non-specialist users. The transformation of these inputs, through an interface, into a LP optimization routine is represented below:

- Generate variables `x1, x2 ... xN`, where `N` equals the number of participating services. These are the variables that will be the valued as 1 or 0 during optimization and represent the selection / rejection of a particular `SiteN()`.
- The `AHPWeight` values `(w1, w2)`, and corresponding `QoS` values `[l1,...,lN], [c1,...,cN]` are combined with the variables to generate a linearly weighted cost function `(w1 l1 + w2 c1)x1 +...+ (w1 lN + w2 cN)xN`.
- The `Constraint` values provide the specified domains, partial orders and corresponding thresholds `(K1, K2)`, which are transformed into (`l1 x1 +...+ lN xN <= K1`; `c1 x1 +... + cN xN <= K2`).
- As the `Routine` "binary integer" is set, values `x1, x2,..., xN` are further constrained to be binary valued. A further constraint automatically specified is the `x1 + x2 +...+ xN = 1`, restricting only a single site is selected by the optimization procedure.

The results of such a transformation produces a LP format of the problem, that can be solved by the lpsolve optimization solver. Due to the elegant nature of Orc, this is equivalent to calling another (possibly external) *Site* with input `Optima` format and output LP format.

```
/* Objective function */
min: (0.3 l1 + 0.7 c1) x1 + (0.3 l2 + 0.7 c2) x2 + (0.3 l3 + 0.7 c3) x3;
/* Variable bounds */
l1 x1 + l2 x2 + l3 x3 <= 0.5;
c1 x1 + c2 x2 + c3 x3 <= 0.8;
x1 + x2 + x3 = 1;
bin x1, x2, x3;
```

In the current stage of implementation calculation of normalized AHP weight vector is be performed using MATLAB. The optimization of QoS values generated from distribution fitting of actual web services' readings is also done

[2] http://lpsolve.sourceforge.net/5.5/

through MATLAB routines. This can be enhanced in future with direct calls to optimization packages (local or external) from within Orc as described. This would prevent switching between technologies while developing workflows in Orc and associated management of QoS dependent decisions.

6 Optimal Decision Results

The results of the optimization procedure are described in this section. Rather than concentrating on optimization aspects (primal-dual feasibility, relative error, number of iterations, etc.), we focus on the implications of using optimization as a tool in orchestrations.

The AHP weight matrix shown in Table 3 is used for the optimization of the problem described in Section 4 using the `linprog` function in MATLAB. These are the judgement criteria that can be fixed by the user / service orchestrator as the inputs to the optimization solver. The polling period ρ is set to a constant of 1 hour to limit model parameters. By setting the customer demand and

Table 3. Parameters for Dell supply chain optimization

	$\mathbf{w}_{\mu_c}$	$\mathbf{w}_{\mu_{max}}$	$\mathbf{w}_{\beta}$	**Normalized Vector** v
$\mathbf{w}_{\mu_c}$	1	1/3	1/5	0.1047
$\mathbf{w}_{\mu_{max}}$	3	1	1/3	0.2583
$\mathbf{w}_{\beta}$	5	3	1	0.6370
$c_{max} = 3.0385$, CI $= 0.0193$, CR $= 0.0370$				

supplier delay distributions, the optimization produces the distributions of the refuel batch size, critical and maximum stock levels as shown in Fig. 3. These settings, when applied to the Dell system provides the system performance as shown in Fig. 4. The cumulative distribution of the revolver inventory remains stochastically above the critical distribution for 10000 runs, being refueled periodically by the supplier. As a consequence, the revolver stock level μ_t does not drop to zero throughout the simulation period. This demonstrates that the optimization formulation through AHP is robust to changes in inputs of demand and delay distributions. Though scaling as in eq. (5) has been employed, the normalized values are omitted from figures (to demonstrate realistic outputs).

As further seen in one particular setting of the Dell example in Fig. 5, the linear programming method converges within a few iterations to the optimal value. This is true for well formulated linear programming problems with optimal outputs produced (relative errors of the order of $\leq 10^{-6}$) for most input settings.

Parameters for optimal evaluations such as relative error, maximum number of iterations and so on can be set conveniently with most generic optimization solvers. Such a precise setting of parameters are needed for orchestrations like the Dell supply chain, to prevent unwarranted delay in production and supply of parts (choreography deadlock). This example highlights the crucial use of optimization and associated packages for managing QoS in complex workflows.

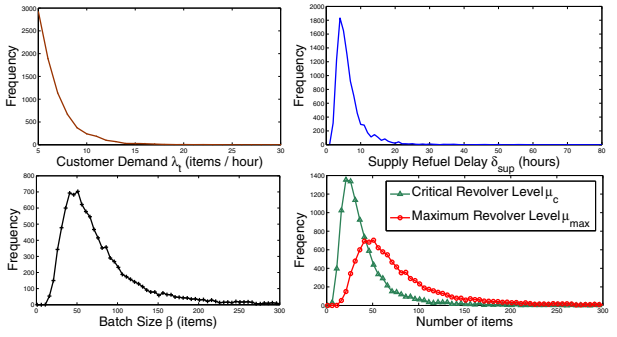


Fig. 3. Optimal setting of parameters in the Dell Supply Chain

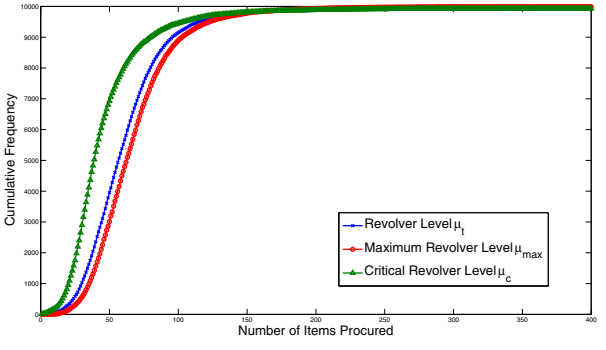


Fig. 4. Distributions of the inventory levels in the Dell system

```
Residuals:   Primal      Dual       Duality     Total
             Infeas      Infeas       Gap        Rel
             A*x-b     A'*y+z-f      x'*z       Error
-------------------------------------------------------
Iter    0:  1.76e+002 5.59e+000 1.35e+003 1.00e+002
Iter    1:  2.07e-014 3.43e-002 8.13e+001 6.73e-001
Iter    2:  1.37e-012 1.25e-016 5.39e+000 1.31e-001
Iter    3:  2.75e-012 1.67e-016 4.76e-002 1.22e-003
Iter    4:  1.48e-012 1.39e-017 2.47e-006 6.36e-008
Iter    5:  6.47e-015 5.46e-014 2.47e-013 7.18e-014
Optimization terminated.
```

Fig. 5. Optimization output for a single setting of the Dell example in MATLAB

7 Related Work

Analysis of QoS in web services orchestrations has received considerable attention. In [1], Hwang et al. use QoS parameters as random variables for composition. Rosario et al. [2] provide a framework for probabilistic contracts modeling QoS parameters as random variables. Instead of using fixed hard bound values for parameters such as response time, the authors proposed a soft contract monitoring approach to model the QoS bounds. This is further developed with a

theory for QoS modeling within the Orc framework in [7]. We extend the "best" operator from this theory to accommodate alternatives to exhaustive search.

Though there are many techniques available for optimizing functions [6] routines needed to incorporate them into orchestrations is still a developing area. In the paper by Alrifai and Risse [14] the use of mixed integer programming is proposed to find the optimal decomposition of global QoS constraints into local constraints. Optimal QoS compositions make use of genetic programming in Canfora et al. [15] and linear programming in Zeng et al. [16]. The use of a reputation guided selection and feedback dependent policy for web services is outlined in [5]. In [17], the optimization of dynamic service compositions are modeled as a multidimension-multichoice knapsack problem (MMKP). MMKP of medium sizes can be solved by most commercial integer-linear programming solvers, as employed in this paper. A framework for specifying optimizations within Orc workflows would aid in deploying real-world applications. This can then be combined with a host of optimization solvers [12] [13] applied to most QoS dependent decisions in service orchestrations.

In this paper, we extend the concepts of optimizing cost function defined via AHP to complex queries in workflows. Extending such a framework to orchestrations can provide more complex queries to be incorporated with flexibility in comparing domains. The Dell optimization example from [9] provide realistic case studies within the web service framework where optimal QoS values affect functioning of the orchestration.

Analytical hierarchy process developed by Saaty [4] has been shown to be applied to diverse fields including manufacturing, logistics, finance and management. Work by Ho [18] reviews the combination of AHP to mathematical models including linear programming, integer linear programming, mixed integer linear programming, and goal programming. An application of AHP for automated negotiation of SLAs are studied in [19]. In [20], another multi-criteria decision making approach (PROMETHEE) is used to extend the decision making for exhaustive comparison of web services' QoS.

8 Conclusion

With increasing need for decision making capabilities in services orchestrations, the use of mathematical packages like optimization should be employed for leveraging QoS dependent choices. Embedding optimization routines as part of orchestration specifying languages like Orc provides the capability to use these tools for runtime decision making in a variety of workflows. A simple extension of user defined criterion and constraints is proposed to specify such optimization problems for non-specialist workflow designers. By applying the AHP, we show that a consistent minimizing cost function can be developed for total ordering QoS metrics. Demonstrating this methodology for the Dell supply chain example, it is shown to be effective in solving realistic problems in resource allocation and logistics. Such techniques are required to estimate optimal decisions on runtime, dependent on variations in associated QoS parameters.

References

1. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. Elsevier Information Sciences 177, 5484–5503 (2007)
2. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations. IEEE Trans. on Services Computing 1(4), 187–200 (2008)
3. W3c, QoS for Web Services: Requirements and Possible Approaches. W3C Working Group Note (November 2003)
4. Saaty, T.L.: How to make a decision: The analytic hierarchy process. European J. of Operational Research 48(1), 9–26 (1990)
5. Limam, N., Boutaba, R.: Assessing Software Service Quality and Trustworthiness at Selection Time. IEEE Trans. on Software Engineering 36(4), 559–574 (2010)
6. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer Series in Operational Research (2006)
7. Rosario, S., Benveniste, A., Jard, C.: A Theory of QoS for Web Service Orchestrations. HAL INRIA Research Report (2009)
8. Saaty, T.L.: Decision-making with the AHP: Why is the principal eigenvector necessary. Elsevier European J. of Operational Research 145, 85–91 (2003)
9. Kapunscinski, R., Zhang, R.Q., Carbonneau, P., Moore, R., Reeves, B.: Inventory Decisions in Dell's Supply Chain. Interfaces 34(3), 191–205 (2004)
10. Rardin, R.L.: Optimization in Operations Research. Prentice Hall (1998)
11. Misra, J., Cook, W.R.: Computation Orchestration: A Basis for Wide-area Computing. J. of Software and Systems Modeling 6(1), 83–110 (2007)
12. Fourer, R., Ma, J., Martin, K.: Optimization Services: A Framework for Distributed Optimization. COIN-OR (2008)
13. Fourer, R., Goux, J.: Optimization as an Internet Resource. Interfaces 31(2), 130–150 (2001)
14. Alrifai, M., Risse, T.: Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition. In: Intl. World Wide Web Conf., Spain (2009)
15. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Conf. on Genetic and Evolutionary Computation, USA, pp. 1069–1075 (2005)
16. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for Web services composition. IEEE Trans. on Software Engineering 30(5), 311–327 (2004)
17. Yu, T., Zhang, Y., Lin, K.: Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. ACM Trans. on the Web 1(1) (2007)
18. Ho, W.: Integrated analytic hierarchy process and its applications A literature review. European J. of Operational Research 186(1), 211–228 (2008)
19. 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)
20. Seo, Y.-J., Jeong, H.-Y., Song, Y.-J.: Best Web Service Selection Based on the Decision Making Between QoS Criteria of Service. In: Yang, L.T., Zhou, X.-s., Zhao, W., Wu, Z., Zhu, Y., Lin, M. (eds.) ICESS 2005. LNCS, vol. 3820, pp. 408–419. Springer, Heidelberg (2005)

Decidability Results for Choreography Realization

Niels Lohmann and Karsten Wolf

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
{niels.lohmann, karsten.wolf}@uni-rostock.de

Abstract. A service choreography defines a set of permitted sequences of message events as a specification for the interaction of services. Realizability is a fundamental sanity check for choreographies comparable to the notion of soundness for workflows.

We study several notions of realizability: partial, distributed, and complete realizability. They establish increasingly strict conditions on realizing services. We investigate decidability issues under the synchronous and asynchronous communication models. For partial realizability, we show undecidability whereas the other two problems are decidable with reasonable complexity.

1 Introduction

A *choreography* describes the interaction of services. In the literature on services, this term has been used for representing the behavior of a system composed of services ("interconnected models") or for the restriction of that behavior to the communication events ("interaction model"). In this paper, we follow the second interpretation. To be more precise, a choreography is typically understood as a *specification* of interaction that can be used as a contract between organizations. This specification is later compared to those interactions that implement the specification. If the implementation produces those interactions which are specified in the choreography, this implementation *realizes* the choreography. Consequently, the question of *realizability* is a fundamental sanity property for choreographies.

The realizability problem has several dimensions. The first dimension is concerned with the notation in which the choreography is given. Several languages have been proposed for choreography description, including WS-CDL [8], Let's Dance [20], UML collaboration diagrams [3], and BPMN 2.0 [14]. They all have in common that they permit the specification of a regular set of sequences of message events. For covering all these languages, we abstract from the syntactic sugar of these languages and assume a choreography to be given in the shape of a finite automaton.

The second dimension for the realizability problem is the communication model assumed. In this paper, we consider synchronous as well as asynchronous communication. In the asynchronous case, we do not assume that messages arrive in the same order in which they have been sent. In the spectrum of reasonable communication models (cf. [10] for a survey), we thus consider the models with the tightest, respectively loosest coupling between sender and receiver of a message. We do not consider FIFO based models.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 92–107, 2011.

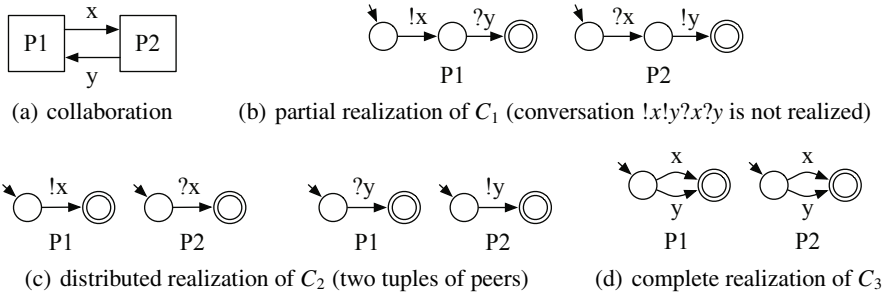


Fig. 1. Collaboration (a) and peer implementations for the partially realizable choreography $C_1 = \{!x!y?x?y,\ !x?x!y?y\}$ (b), the distributedly realizable choreography $C_2 = \{!x?x,\ !y?y\}$ (c), and the completely realizable choreography $C_3 = \{x,\ y\}$ (d)

In the third dimension of the realizability problem, we need to determine what it exactly means for an implementation to conform to a choreography. Following earlier considerations [12], we study three concepts: partial, distributed, and complete realization. In a partial realization, the implementation produces some, but not necessarily all sequences of message events specified in the choreography, cf. Fig. 1(b). Here, the choreography is seen as a space of opportunities which need not be exhausted by the implementation. Distributed choreography follows the same intention, but assures that the choreography does not contain junk sequences which cannot be contained in any realization. Hence, a choreography is distributedly realizable if there is a (possibly infinite) family of implementations such that each specified sequence of the choreography is realized in at least one of them, cf. Fig. 1(c). Complete realizability, in turn, requires that all sequences specified in the choreography can be produced in a single implementation, cf. Fig. 1(d). The three concepts form a hierarchy; that is, complete realizability implies distributed, and distributed implies partial realizability.

Contribution. We show that, for both considered communication models, partial realizability is undecidable whereas distributed and complete realizability are decidable. Our undecidability results depend on a reduction of the famous undecidable Post correspondence problem (PCP). The decision procedures for distributed and complete realizability depend on standard language theoretic constructions such as projection, checking language equivalence, and minimization of automata. Thus, despite exponential worst case complexity, we may assume mature algorithms with reasonable run times.

Organization. After giving the formal definitions of our concepts (Sect. 2), we study first partial (Sect. 3), then distributed (Sect. 4), and finally complete realizability (Sect. 5). In each of the sections, we first present our results for synchronous communication in full detail. Then, for space reasons, we just briefly discuss how these arguments need to be modified in the asynchronous case. In Sect. 6 we discuss related work before Sect.7 concludes the paper and lists open problems.

2 Basic Definitions

2.1 Interconnected Models and Interaction Models

Throughout this paper, fix a finite set of message channels M that is partitioned into asynchronous message channels M_A and synchronous message channels M_S. From M, derive a set of message events $E := !E \cup ?E \cup M_S$, consisting of asynchronous send events $!E := \{!x \mid x \in M_A\}$, asynchronous receive events $?E := \{?x \mid x \in M_A\}$, and synchronization events. Furthermore, we distinguish a non-communicating event $\tau \notin E$. For an event $x \in E$, define *channel*$(x) = a$ if $x = a$, $x = ?a$, or $x = !a$.

Definition 1 (Peer, collaboration). *A* peer $P = [I, O]$ *consists of a set of input message channels $I \subseteq M$ and a set of output message channels $O \subseteq M$, $I \cap O = \emptyset$. A* collaboration *is a set $\{[I_1, O_1], \ldots, [I_n, O_n]\}$ of peers such that $I_i \cap I_j = \emptyset$ and $O_i \cap O_j = \emptyset$ for all $i \neq j$, and $\bigcup_{i=1}^{n} I_i = \bigcup_{i=1}^{n} O_i$.*

A peer and a collaboration (cf. Fig. 1(a)) can be seen as a syntactic signature of a service and a composition, respectively. The behavior itself (i.e., the order in which messages are exchanged and when a peer terminates) is modeled by *peer automata*. A peer automaton is a state machine whose transitions are labeled by message events or τ.

Definition 2 (Peer automaton). *A* peer automaton $A = [Q, \delta, q_0, F, \mathcal{P}]$ *is a tuple such that Q is a set of states, $\delta \subseteq Q \times (E_I \cup E_O \cup \{\tau\}) \times Q$ is a transition relation, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, and $\mathcal{P} = \{[I_1, O_1], \ldots, [I_n, O_n]\}$ is a nonempty set of peers. Thereby, $E_I := \{?x \mid x \in M_A \cap \bigcup_{i=1}^{n} I_i\} \cup (M_S \cap \bigcup_{i=1}^{n} I_i)$ are the input events of A and $E_O := \{!x \mid x \in M_A \cap \bigcup_{i=1}^{n} O_i\} \cup (M_S \cap \bigcup_{i=1}^{n} O_i)$ are output events of A.*

A implements *the peers $\mathcal{P}$, and for $[q, x, q'] \in \delta$, we also write $q \xrightarrow{x} q'$. A is called a* single-peer automaton, *if $|\mathcal{P}| = 1$. A is called a* multi-peer automaton, *if $|\mathcal{P}| > 1$ and $\mathcal{P}$ is a collaboration. A is called* τ-free *if $q \xrightarrow{x} q'$ implies $x \neq \tau$ for all $q, q' \in Q$. A is called* deterministic *if A is τ-free and $q \xrightarrow{x} q'$ and $q \xrightarrow{x} q''$ imply $q' = q''$. A is called* finite *if the number of states reachable from q_0 is finite. An* accepting run *of A is a sequence of events $x_1 \cdots x_m$ such that $q_0 \xrightarrow{x_1} \cdots \xrightarrow{x_m} q_f$ with $q_f \in F$.*

The interplay of peers is modeled by their *composition*. In case of asynchronous communication, pending messages are represented by a multiset. Denote the set of all multisets over M_A with $Bags(M_A)$, the empty multiset with $[\,]$, and the multiset containing only one instance of $x \in M_A$ with $[x]$. Addition of multisets is defined pointwise.

Definition 3 (Composition of single-peer automata). *Let $A_1, \ldots, A_n$ be finite single-peer automata ($A_i = [Q_i, \delta_i, q_{0_i}, F_i, \{P_i\}]$ for $i = 1, \ldots, n$) such that their peers form a collaboration. Define the* composition $A_1 \oplus \cdots \oplus A_n$ *as the multi-peer automaton $[Q, \delta, q_0, F, \{P_1, \ldots, P_n\}]$ with $Q := Q_1 \times \cdots \times Q_n \times Bags(M_A)$, $q_0 := [q_{0_1}, \ldots, q_{0_n}, [\,]]$, $F := F_1 \times \cdots \times F_n \times \{[\,]\}$, and, for all $i \neq j$ and $B \in Bags(M_A)$ the transition relation δ contains exactly the following elements:*

- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{\tau} [q_1, \ldots, q_i', \ldots, q_n, B]$, *if and only if $[q_i, \tau, q_i'] \in \delta_i$ (internal move by A_i),*

- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{!x} [q_1, \ldots, q_i', \ldots, q_n, B + [x]]$, *if and only if* $x \in M_A$ *and* $[q_i, !x, q_i'] \in \delta_i$ *(asynchronous send by* A_i*),*
- $[q_1, \ldots, q_i, \ldots, q_n, B + [x]] \xrightarrow{?x} [q_1, \ldots, q_i', \ldots, q_n, B]$, *if and only if* $x \in M_A$ *and* $[q_i, ?x, q_i'] \in \delta_i$ *(asynchronous receive by* A_i*), and*
- $[q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n, B] \xrightarrow{x} [q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n, B]$, *if and only if* $x \in M_S$, $[q_i, x, q_i'] \in \delta_i$, *and* $[q_j, x, q_j'] \in \delta_j$ *(synchronization between* A_i *and* A_j*).*

The composition of two finite service automata may have an infinite number of states, because we consider arbitrary multisets of asynchronous messages. In the remainder, we only consider finite compositions.

2.2 Languages and Traces

The results in the next sections heavily rely on concepts of regular languages and traces.

Definition 4 (Automaton versus language). *Let* $A = [Q, \delta, q_0, F, \mathcal{P}]$ *be a finite peer automaton. For an accepting run* ρ*, define the* event sequence *of* ρ *as* $\rho_{|E}$ *(i.e.,* ρ *without* τ*-steps). The* language *of A, denoted* $\mathcal{L}(A)$*, is the set of the event sequences of all accepting runs of A. The other way round, if* $\mathcal{P}$ *is a set of peers and L is a regular language over the alphabet* $\bigcup_{[I,O]\in\mathcal{P}} I \cup O$ *then* $\mathcal{A}(L)$ *is the minimal (regarding size of Q) finite, deterministic, and* τ*-free peer-automaton that implements* $\mathcal{P}$ *and has* $\mathcal{L}(\mathcal{A}(L)) = L$*.*

Formal languages theory asserts that, for every nonempty regular language L, an automaton $\mathcal{A}(L)$ exists and is unique up to isomorphism. Throughout this paper, we only consider regular languages and choreographies. This accords to many industrial and academic choreography specification languages.

Definition 5 (Choreography). *Let* $\mathcal{P} = \{[I_1, O_1], \ldots, [I_n, O_n]\}$ *be a collaboration. A* conversation *of* $\mathcal{P}$ *is a word over the events of* $\mathcal{P}$*. A* choreography *for* $\mathcal{P}$ *is a nonempty regular set of conversations of* $\mathcal{P}$*.*

The individual realizability notions differ in the amount of conversations which must be realized by the peers. They all have in common that no new conversation must be introduced. Hence, the projected peers need to be *coordinated* at design time such that they do not produce unspecified conversations. The example choreographies in Fig. 1 show that this coordination can already be impossible even if two peers share message channels. To characterize possible and impossible coordination, we first introduce *distant* message events. We call two message events distant if there exists no peer which can observe both, for instance $!x$ and $!y$ in Fig. 1(b):

Definition 6 (Distant message events). *Let* $\mathcal{P}$ *be a collaboration. Two message events* $a, b \in E$ *are* distant *if and only if there exist no peer* $[I, O] \in \mathcal{P}$ *such that* $\{channel(a), channel(b)\} \subseteq (I \cup O)$*.*

Several results in this article shall depend on the observation that no composition of peer automata is able to enforce any order on concurrently activated distant events. That is, if distant events subsequently occur in one order, they can also occur in the reverse

order. An exception are related asynchronous send and receive events. They are distant according to our definition but the send event always precedes the corresponding receive event as long as no other message of this kind is pending. The following definition formalizes this observation. Whether there are pending asynchronous message, can be determined by a simple counting on the prefix of the sequence.

Definition 7 (Message counting, trace). *For $x \in M_A$ and an event sequence w, define $\hat{x}(w)$ by the following induction scheme:*

Base: *For the empty sequence ε, let $\hat{x}(\varepsilon) = 0$.*
Step: *Let $\hat{x}(wa) = \hat{x}(w) + 1$, if $a = !x$, $\hat{x}(wa) = \hat{x}(w) - 1$, if $a = ?x$, $\hat{x}(wa) = \hat{x}(w)$, for all other events a.*

Let $\mathcal{P}$ be a collaboration. For a word w over the alphabet E, define the trace *of w, $\langle w \rangle_{\mathcal{P}}$, by the following induction scheme:*

Base: *Let $w \in \langle w \rangle_{\mathcal{P}}$.*
Step: *For all distant events a, b such that there is no $x \in M$ with $a = !x$ and $b = ?x$, $w_1 abw_2 \in \langle w \rangle_{\mathcal{P}}$ implies $w_1 baw_2 \in \langle w \rangle_{\mathcal{P}}$ and, if $\hat{x}(w_1) > 0$, $w_1 !x?xw_2 \in \langle w \rangle_{\mathcal{P}}$ implies $w_1 ?x!xw_2 \in \langle w \rangle_{\mathcal{P}}$.*

If $\mathcal{P}$ is clear from the context,we simply write $\langle w \rangle$ instead of $\langle w \rangle_{\mathcal{P}}$.

This concept is very similar to local traces [11]. If $M_a = \emptyset$, it coincides with Mazurkiewicz traces [13] which have been intensely investigated [5]. By our definition of composition, the message count functions $\hat{x}$ will always return 0 for event sequences of terminating runs and values greater than or equal to 0 for prefixes of terminating runs.

Proposition 1 (Notation for languages). *Let L_1 and L_2 be regular languages. Then (1) the concatenation of L_1 and L_2, $L_1 L_2$, (2) the complement of L_1, $\overline{L_1}$, (3) the union of L_1 and L_2, $L_1 \cup L_2$, (4) the difference of L_1 and L_2, $L_1 \setminus L_2$, (5) iteration/Kleene star, L_1^*, (6) nonempty iteration, $L_1^+ = L_1 L_1^*$, (7) projection to the letters appearing in the events of $\mathcal{P}$, $L_1|_{\mathcal{P}}$, and (8) the shuffle product of L_1 and L_2, $L_1 \,||\, L_2$, are regular.*

3 Partial Realizability

Definition 8 (Partial realizability). *Let C be a choreography for a collaboration $\{P_1, \ldots, P_n\}$. The finite single-peer automata $A_1, \ldots, A_n$* partially realize *C if, for all i, A_i implements $\{P_i\}$ and $\emptyset \neq \mathcal{L}(A_1 \oplus \cdots \oplus A_n) \subseteq C$.*

Example. The choreography C_1 in Fig. 1 is only partially realizable (e.g., by the peer automata depicted in Fig. 1(b)), because the conversation $!x!y?x?y$ cannot be implemented by peers without also producing the unspecified conversations $!y!x?x?y$ or $!y!x?y?x$. Only the conversation $!x?x!y?y$ is realized.

3.1 The Synchronous Case

In this subsection, we assume $M_A = \emptyset$. We shall show that partial decidability is undecidable for $n \geq 4$ and trivial for $n \leq 3$. We start with some simple observations about partial realizability in presence of distant events.

Proposition 2. *Let $A_1, \ldots, A_n$ be finite single-peer automata whose collection of peers forms a collaboration. Then for all sequences w over E, $w \in \mathcal{L}(A_1 \oplus \cdots \oplus A_n)$ implies $\langle w \rangle \subseteq \mathcal{L}(A_1 \oplus \cdots \oplus A_n)$.*

Proof. Follows directly from the definition of composition and distant events. □

That is, realization cannot be finer than the level of granularity of traces. The other way round, we can realize a single conversation w provided that the choreography contains its whole trace $\langle w \rangle$. This can be done by letting each peer automaton execute, in sequence, those letters of w which occur in its set of peers. Formally:

Proposition 3. *Let w be a sequence over E. Then $\mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(\{w\}|_{\mathcal{P}_i})) = \langle w \rangle$.*

Proof. Follows directly from the definition of composition and distant events. □

Joining these two observations, we obtain a characterization of partial realizability that we shall use throughout the remainder of this section.

Lemma 1. *A choreography C for a collaboration $\mathcal{P}$ is partially realizable if and only if it contains a conversation w with $\langle w \rangle \subseteq C$.*

Proof. Implication: Assume C is partially realizable. Then exists at least one conversation w that is realized by peers and Proposition 2 states that $\langle w \rangle \subseteq C$.
Replication: Assume there exists a conversation w with $\langle w \rangle \subseteq C$. Then Proposition 3 states that the single-peer automata $\mathcal{A}(\{w\}|_{\mathcal{P}_1}), \ldots, \mathcal{A}(\{w\}|_{\mathcal{P}_n})$ realize $\langle w \rangle$ and, as $\emptyset \neq \langle w \rangle \subseteq C$, also partially C. □

We are now ready to consider the case of at most three peers.

Theorem 1. *Let C be a choreography for a collaboration with at most three peers. Then C is partially realizable if and only if $C \neq \emptyset$.*

Proof. Take an arbitrary conversation w in C and apply the construction of Prop. 3. Observe further that there cannot be distant events, because each event is shared by two peers. This means that the realized language is $\{w\}$ which is clearly a nonempty subset of C. □

For the case of four or more peers, we show undecidability.

Theorem 2. *Partial realizability is undecidable for choreographies that involve at least four peers.*

Undecidability is shown by reduction of the famous *Post correspondence problem* using a proof pattern that is inspired by a proof in [15].

Definition 9 (Post correspondence problem (PCP)). *A* Post system *over alphabet X is a finite set $P = \{[u_1, v_1], \ldots, [u_k, v_k]\}$ of ordered pairs of words $u_i, v_i \in X^*$. A* candidate *is a nonempty finite sequence $i_1 \cdots i_n$ of indices $i_j \in \{1, \ldots, k\}$. Candidate $i_1 \cdots i_n$ is a* solution *of Post system P if $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$. The* Post correspondence problem *is to decide for a given Post system P whether it has a solution.*

In other words, the question is whether we can arrange the pairs (in arbitrary copies) such that the concatenation of the left elements yields the same sequence as the concatenation of the right elements. Undecidability of PCP is a classical result in the theory of computable functions.

In the sequel, we show that decidability of partial realizability would imply decidability of PCP. Consequently, we start with a Post system P and construct a choreography C for a collaboration such that P has a solution if and only if C is partially realizable; that is, C includes $\langle w \rangle$ for at least one conversation $w \in C$.

Message channels. Let X be the alphabet used in P and assume that $X' := \{x' \mid x \in X\}$ is another, disjoint alphabet of same size. For a sequence $x_1 \cdots x_m$ in X let $(x_1 \cdots x_m)' = x'_1 \cdots x'_m$. Let k be the number of pairs in P and assume further, without loss of generality, that none of X and X' contain elements from $\{1, \ldots, k\}$. Set $M_S = X \cup X' \cup \{1, \ldots, k\}$ and $M_A = \emptyset$.

Collaboration. We translate P into a collaboration with four peers $P_1 = [I_1, O_1], \ldots, P_4 = [I_4, O_4]$. We set $I_1 = O_2 = X \cup \{1, \ldots, k\}$, $I_3 = O_4 = X'$, and $O_1 = I_2 = O_3 = I_4 = \emptyset$. This means that two messages are distant if and only if one of them is in $X \cup \{1, \ldots, k\}$ and the other is in X'.

Encoding of candidates. Consider the following encoding of an arbitrary candidate $i_1 \cdots i_n$ of P: $w(i_1 \cdots i_n) = i_1 u_{i_1} v'_{i_1} \cdots i_n u_{i_n} v'_{i_n}$. That is, we have a sequence of blocks where each block consists of a pair number, the left side of the pair, and the primed version of the right side of the pair. If $i_1 \cdots i_n$ is a solution, the projection of $w(i_1 \cdots i_n)$ to X yields the same sequence as its projection to X' (up to the "priming" of the letters in X'). Letters in X do not commute, so the projection to X is the same for all members of the trace $\langle w(i_1 \ldots i_n) \rangle$. The same is true for the projection to X'. On the other hand, letters of X' commute arbitrarily with letters in X and in $\{1, \ldots, k\}$. This leads us to the core observation for our construction.

Proposition 4. *The sequence $i_1 \cdots i_n$ is a solution of the Post system P if and only if the trace $\langle w(i_1 \cdots i_n) \rangle$ contains a word of the language defined by the regular expression $(x_1 x'_1 \mid \cdots \mid x_n x'_n \mid 1 \mid \cdots \mid k)^*$.*

In other words, the letters of X and X' can be adjusted such that they can be compared letter by letter (and the pair numbers occur somewhere in between).

Example. As an example, consider the Post system $P = \{[a, baa], [ab, aa], [bba, bb]\}$ with $k = 3$ pairs over the alphabet $X = \{a, b\}$. Define $X' = \{a', b'\}$ and the peers $P_1 = [\{a, b, 1, 2, 3\}, \emptyset]$, $P_2 = [\emptyset, \{a, b, 1, 2, 3\}]$, $P_3 = [\{a', b'\}, \emptyset]$, and $P_4 = [\emptyset, \{a', b'\}]$. Consider the candidate 3 2 3 1 and define the word $w(3\,2\,3\,1) = 3bbab'b'\,2aba'a'\,3bbab'b'\,1ab'a'a'$. The letters in this word can be reordered, and in the trace $\langle w(3\,2\,3\,1) \rangle$ we can find the word $3bb'bb'aa'2aa'bb'3bb'bb'aa'1aa'$. By Prop. 4, we can conclude that the candidate 3 2 3 1 is a solution, and indeed *bba ab bba a = bb aa bb baa*.

Choreography. We want the choreography C to contain all sequences w on $X \cup X' \cup \{1, \ldots, k\}$ except

(1) at least one sequence of the trace $\langle w \rangle$ if the word w cannot be reshuffled to the encoding of some candidate of P and
(2) at least one sequence of the trace $\langle w \rangle$ if the word projections of w to X and X' lead to different sequences.

At the same time we need to assure that,

(3) for any solution $i_1 \cdots i_n$ of P, the trace of its encoding, $\langle w(i_1 \ldots i_n) \rangle$, is indeed fully contained in C.

The recognition of the faulty sequence w is facilitated by the fact that the respective trace $\langle w \rangle$ contains all possible reshufflings of w. These reshufflings contain normal forms for which the characterization of fault sequences is straightforward.

It is easy to see that a choreography that satisfies (1), (2), and (3) is indeed partially realizable if and only if P has a solution. Hence, it remains to show that there are regular languages L_1 and L_2 such that

- L_1 contains at least one sequence of $\langle w \rangle$ if w cannot be reshuffled to the encoding of some candidate of P (1), but no sequence of $\langle w(i_1 \cdots i_n) \rangle$, for any solution $i_1 \cdots i_n$ of P (3) and
- L_2 contains at least one sequence of $\langle w \rangle$ if the projections of w to X and X' lead to different sequences (2), but no sequence of $\langle w(i_1 \cdots i_n) \rangle$, for any solution $i_1 \cdots i_n$ of P (3).

We present L_1 and L_2 as expressions using the operations mentioned in Def. 1 which proves regularity. Concerning L_1, there can be two reasons for the incapability to reshuffle a conversation to the encoding of any candidate. First, the projection of a word to $X \cup \{1, \ldots, k\}$ may not correspond to a sequence of pair numbers and corresponding left elements of pairs. As this projection is invariant under reshuffling (no pair of letters in $X \cup \{1, \ldots, k\}$ is distant to each other), removal of such words cannot compromise condition (3). This is reflected in

$$L_1 = L_{11} \cup L_{12} \quad \text{with} \quad L_{11} = \overline{(1u_1 \mid \cdots \mid ku_k)^*} \,||\, X'^*.$$

Second, the projection to X' may not deliver the (unique) sequence that fits to the projection to $X \cup \{1, \ldots, k\}$. This, in turn, may be caused by (a) excess letters from X' after having served all pairs or (b) the incapability to complement some iu_i with the unique fitting v'_i. We model L_{12} such that we detect the problem immediately subsequent to the largest prefix that can be shuffled into the correct encoding. Consequently, let

$$L_{12} = (1u_1v'_1 \mid \cdots \mid ku_kv'_k)^* \; (L_a \cup L_b).$$

The two languages in the tail of this expression correspond to the mentioned problems. Thus,

$$L_a = X'^+ \quad \text{and} \quad L_b = \bigmid_{j=1}^{k} \Big(ju_j((X'^* \setminus v'_jX'^*) \,||\, (X \cup \{1, \ldots, k\})^*) \Big).$$

For language L_2, we only need to detect a single mismatch or excess letters in either subalphabet. Let $X = \{x_1, \ldots, x_m\}$. Then we set

$$L_2 = (x_1x'_1 \mid \cdots \mid x_mx'_m \mid 1 \mid \cdots \mid k)^* \, (X^+ \mid X'^+ \mid \mid_{i \neq j} x_ix'_j(X \cup X' \cup \{1, \ldots, k\})^*).$$

For both languages L_1 and L_2, the construction transparently shows that they satisfy the specified conditions. Hence, we may come to our final conclusion that

$$C := (X \cup X' \cup \{1, \ldots, k\})^* \setminus (L_1 \cup L_2)$$

contains a word w with $\langle w \rangle \subseteq L$ if and only if the Post system P has a solution. This concludes the proof of Theorem 2.

Example (cont.). For the example Post system, the following words are examples for the defined languages:

- $1aba'\,2ab'a' \in L_{11}$ — this word uses pairs $[ab, a], [a, ba] \notin P$.
- $3bbab'b'a' \in L_{12}$ — the letter a' after pair $[bba, bb]$ is too much.
- $2aba'b' \in L_{12}$ — this word uses a pair $[ab, ab] \notin P$.
- $3bb'bb'aa'1ab'a'a' \in L_2$ — no solution, because one a is not matched.

3.2 The Asynchronous Case

Assume now $M_S = \emptyset$. We show that the arguments used in the synchronous case extend to asynchronous communication. First, Propositions 2 and 3 hold in the asynchronous case as well. For the latter proposition, observe in particular that events $!x$ and $?x$ commute only if a message x is pending before the execution of the considered event $!x$. This is reflected both in the definition of composition and the definition of traces.

For the PCP reduction of a PCP instance P with k pairs, a topology with three peers P_1, P_2, and P_3. is sufficient. Having distinct events for sending and receiving, we can use $M_A = X \cup \{1, \ldots, k\}$. While the sending events take the role of X in the previous subsection, the corresponding receiving events replace the primed letters above. Then $O_1 = !X \cup \{!1, \ldots, !k\}$, $I_2 = ?X$, $I_3 = \{?1, \ldots, ?k\}$, and $I_1 = O_2 = O_3 = \emptyset$. We use the same choreography as above, except for the fact that we assume P_3 to receive messages arbitrarily; that is, we shuffle the choreography used above with $(\{?1, \ldots, ?k\})^*$. Send and corresponding receive events are distant except for the case that no message of shape x is pending. However, we can exploit that already a "monotonous" version of PCP is undecidable:

> Given a Post system $\{[u_1, v_1], \ldots, [u_k, v_k]\}$, is there a candidate $i_1 \cdots i_n$ such that $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$ and, for all $j < n$, $v_{i_1} \cdots v_{i_j} \sqsubseteq u_{i_1} \cdots u_{i_j}$.

Thereby, $\sqsubseteq$ denotes the prefix operator: the left pairs are always a prefix of the right pairs. Undecidability can be observed from the standard reduction of the halting problem for Turing machines to PCP. In this proof, the difference between the u-sequence and the v-sequence is used for coding configurations of the Turing machine. That is, the u-sequence is always ahead of the v-sequence which can only catch up after having passed a terminating configuration of the machine.

In the monotonous setting, the coding of a PCP solution satisfies the condition that every receive event is preceded by sufficiently many send events. The reshuffling to a form where send and corresponding receive events are immediate neighbors is also not blocked by inactivated receive events, Thus, the argument used in the synchronous case extends to the asynchronous case.

Corollary 1. *Partial realizability under the asynchronous communication model is undecidable if at least three peers are involved.*

4 Distributed Realizability

Definition 10 (Distributed realizability). *Let C be a choreography for a collaboration $\{P_1, \ldots, P_n\}$. The set of tuples of finite single-peer automata $\{[A_{1j}, \ldots, A_{nj}] \mid j \in \mathbb{N}^+\}$ is* distributedly realize *C if, for $i = 1, \ldots, n$ and all j, (i) A_{ij} implements $\{P_i\}$, (ii) $\emptyset \neq \mathcal{L}(A_{1j} \oplus \cdots \oplus A_{nj}) \subseteq C$, and (iii) $\bigcup_j \mathcal{L}(A_{1j} \oplus \cdots \oplus A_{nj}) = C$.*

Example. The choreography C_2 in Fig. 1 is distributedly realizable: There exist two tuples of peers (cf. Fig. 1(c)) such that every conversation of the choreography is implemented. As the distant events $!x$ and $!y$ cannot be coordinated, C_2 is not completely realizable.

4.1 The Synchronous Case

Again assume $M_A = \emptyset$. Distributed realizability can be rephrased using traces.

Theorem 3. *A choreography C for a collaboration $\mathcal{P}$ is distributedly realizable if and only if $C = \bigcup_{w \in C} \langle w \rangle$.*

Proof. If $C = \bigcup_{w \in C} \langle w \rangle$, Prop. 3 proves distributed realizability. The other way round, if there is some $w \in C$ with $\langle w \rangle \not\subseteq C$, Prop. 2 shows that w cannot be covered by any realization. □

It remains to find an effective way to check whether $C = \bigcup_{w \in C} \langle w \rangle$. This problem has, however, already been solved in trace theory [5]:

Proposition 5. *Let C be a choreography for a collaboration $\mathcal{P}$ and $A = \mathcal{A}(C)$ the minimal deterministic automaton that accepts the language of C. $C = \bigcup_{w \in C} \langle w \rangle$ if and only if, for all states q_1, q_2 of A and all distant events x and y, $q_1 \xrightarrow{xy} q_2$ implies $q_1 \xrightarrow{yx} q_2$.* □

Although we imported the result, we present the sketch of the proof for reasons of self-containedness. Minimal deterministic automata are linked to the Nerode relation $\sim_L$. For a language L, let $w_1 \sim_L w_2$ if, for all w, it holds that $w_1 w \in L$ if and only if $w_2 w \in L$. The main observation on the Nerode relation is that, in any automaton accepting L, $q_0 \xrightarrow{w_1} q$ and $q_0 \xrightarrow{w_2} q$ implies $w_1 \sim_L w_2$. For the minimal deterministic automaton accepting L, the reverse holds as well: If $w_1 \sim_L w_2$, $q_0 \xrightarrow{w_1} q_1$ and $q_0 \xrightarrow{w_2} q_2$ then $q_1 = q_2$. Applying this observation to our problem, we see that, for all sequences w and distant events a and b, we have $wab \sim_C wba$ thus proving the above result.

4.2 The Asynchronous Case

As Propositions 2 and 3 extend to the asynchronous case (i. e., $M_S = \emptyset$), so does Thm. 3.

Corollary 2. *A choreography C for a collaboration $\mathcal{P}$ that uses only asynchronous communication is distributedly realizable if and only if $C = \bigcup_{w \in C} \langle w \rangle$.*

The actual decision procedure requires some additional considerations, though. We start with reminding that no messages are pending after termination. That is, for all terminating runs w and all messages x, $\hat{x}(w) = 0$. This observation can be used for extending the $\hat{}$-notation to states of any automaton A that accepts C.

Lemma 2. *Let C be a distributedly realizable choreography. Let A be an automaton that accepts C. Assume that A does not have trap states; that is, states from which no final state of A is reachable. For all sequences w_1 and w_2, if $q_0 \xrightarrow{w_1} q$ and $q_0 \xrightarrow{w_2} q$ then, for all $x \in M_A$, $\hat{x}(w_1) = \hat{x}(w_2)$.*

Proof. Assume the contrary. As a final state is reachable from q, say by executing w, both $w_1 w$ and $w_2 w$ are accepted in A. One of these sequences has an unbalanced number of send and receive events for some x, violating the termination condition for compositions and thus contradicting distributed realizability of C. □

This observation yields a simple necessary condition for distributed realizability:

Corollary 3. *Let C be a choreography and A an automaton that has no trap states and accepts C. Then C is realizable only if the following system of equations has a unique and nonnegative solution. In the system, for each state q of A and $x \in M_A$, $\hat{q}(x)$ is a distinct variable and we impose the following equations.*

- $\hat{q}(x) = 0$, *for all $x \in M_A$ and all $q \in \{q_0\} \cup F$;*
- $\hat{q}(x) + 1 = \hat{q}'(x)$, *if $q \xrightarrow{!x} q'$;*
- $\hat{q}(x) - 1 = \hat{q}'(x)$, *if $q \xrightarrow{?x} q'$;*
- $\hat{q}(x) = \hat{q}'(x)$, *if $q \xrightarrow{y} q'$, $y \neq !x$, and $y \neq ?x$.*

In the following considerations, we assume that C passed this sanity check and thus employ the solution $\hat{q}(x)$ of the presented system of equations. Reflecting the restrictions for commutation of $!x$ and $?x$, we propose the following modification of Prop. 5.

Lemma 3. *Let C be a choreography for a collaboration $\mathcal{P}$ using asynchronous communication satisfying the condition established in Cor. 3 and let $A = \mathcal{A}(C)$ the minimal deterministic automaton that accepts the language of C. $C = \bigcup_{w \in C} \langle w \rangle$ if and only if, for all states q_1, q_2 of A and all distant events a and b:*

- *If there is no message x with $a = !x$ and $b = ?x$ then $q_1 \xrightarrow{ab} q_2$ implies $q_1 \xrightarrow{ba} q_2$*
- *If, for some message x, $a = !x$ and $b = ?x$, and $\hat{q_1}(x) > 0$ then $q_1 \xrightarrow{ab} q_2$ implies $q_1 \xrightarrow{ba} q_2$*

Proof. Again, the proof relies on the relation between the Nerode equivalence and the minimal deterministic automaton accepting C. Indeed, in all situations where the conditions for a and b are satisfied, we have $wab \sim_C wba$ thus justifying the stated diamond property in the automaton. In particular, condition $\hat{q_1}(x) > 0$ asserts that $!x$ and $?x$ commute. □

4.3 Complexity

Complexity depends on the assumptions to be imposed on the original representation of C. From most relevant choreography description languages we are aware of, it is easy to derive a finite automaton model for the choreography. Thus, we assume such an automaton for C to be given. On the other hand, we do not assume this automaton to be deterministic, let alone minimal. Thus, the costs for checking distributed realizability comprise the efforts for:

- transforming the given automaton into a minimal deterministic one. This involves the well known power set construction for transforming a nondeterministic automaton into a deterministic one which may cause exponential blow-up in the number of states;
- checking the diamond property of Prop. 5 or Lemma 3 which can be done in linear time with respect to the number of states of the automaton.

In the asynchronous case, we additionally need to solve the linear system of equations of Cor. 3 which requires, for tis particular system of equations, only linear time as well.

As the most costly step, transformation into a deterministic automaton, is well studied in the area of compiler construction, we believe that existing standard solutions will be sufficiently efficient for practice.

5 Complete Realizability

Definition 11 (Complete realizability). *Let C be a choreography for a collaboration $\{P_1, \ldots, P_n\}$. The finite single-peer automata $A_1, \ldots, A_n$* completely realize *C if, for all i, A_i implements $\{P_i\}$ and $\mathcal{L}(A_1 \oplus \cdots \oplus A_n) = C$.*

Example. The choreography C_3 in Fig. 1 is completely realizable: Every specified conversation is implemented by the peers in Fig. 1(d).

5.1 The Synchronous Case

Assume $M_A = \emptyset$.

Theorem 4. *A choreography C for a collaboration $\{P_1, \ldots, P_n\}$ is completely realizable if and only if it is completely realized by $\mathcal{A}(C|_{P_1}), \ldots, \mathcal{A}(C|_{P_n})$,*

Proof. If the automata $\mathcal{A}(C|_{P_i})$ $(1 \leq i \leq n)$ completely realize C, nothing remains to be shown. So assume C is completely realizable, say, by finite single-peer automata $B_1, \ldots, B_n$. We show that C is also realized by the $\mathcal{A}(C|_{P_i})$ $(i = 1, \ldots, n)$.

We show first $C \subseteq \mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$. Let $w \in C$. Every automaton $\mathcal{A}(C|_{P_i})$ has $w|_{P_i}$ as one of its accepting runs. Consequently, w can be realized using a suitable scheduling of the events in $\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i})$.

Next, we show $C \supseteq \mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$. Let $w \in \mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$. When realizing w, automaton $\mathcal{A}(C|_{P_i})$ executes $w|_{P_i}$ $(i = 1, \ldots, n)$. By construction of these automata, this is only possible if there are conversations $w_i \in C$ $(i = 1, \ldots, n)$ such that $w_i|_{P_i} = w|_{P_i}$.

As the composition of the B_i realizes at least the conversations in C, they realize all the words w_i ($i = 1, \ldots, n$). In a run that produces w_i, automaton B_i executes the event sequence $w_i|_{P_i} = w|_{P_i}$. Consider now a run where each of the B_i executes the event sequence $w|_{P_i}$ in the order given by w. Globally, this run produces w. As the composition of the B_i realizes at most the conversations specified in C, we finally conclude $w \in C$. □

In contrast to the simplistic automata used in the previous sections, the composition of automata $\mathcal{A}(C|_{P_i})$ may contain deadlocks; that is, runs which cannot be extended in a nonfinal state. We show, however, that this is the case only if all complete realizations contain deadlocks.

Theorem 5. *A choreography C for the collaboration $\{P_1, \ldots, P_n\}$ is completely and deadlock freely realizable if and only if it is completely and deadlock freely realized by $\mathcal{A}(C|_{P_1}), \ldots, \mathcal{A}(C|_{P_n})$,*

Proof. In addition to the arguments in Thm. 4, it remains to be shown that every partial run in $\mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$ can be extended to a terminating run if that is possible in any complete realization $B_1 \oplus \cdots \oplus B_n$ of C. Let w_1 be a partial run in $\mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$ that does not end in a final state, thus $w_1 \notin C$. Using the same argument as for Thm. 4, we can show that w_1 is also the event sequence produced by some run in $B_1 \oplus \cdots \oplus B_n$ which cannot be a terminating run, because $w_1 \notin C$. Thus, $B_1 \oplus \cdots \oplus B_n$ is able to extend the run to a terminating run by executing an additional event sequence w_2 (i.e., $w_1 w_2 \in C$). As $\mathcal{L}(\bigoplus_{i=1}^{n} \mathcal{A}(C|_{P_i}))$ completely realizes C, $w_1 w_2$ is also executable here. Since all the $\mathcal{A}(C|_{P_i})$ are deterministic by definition, there is only one state that can be reached after having executed w_1. Hence, the unique state reached by the partial run w_1 enables the continuation w_2 and thus cannot be a deadlock. □

5.2 The Asynchronous Case

Under the asynchronous communication model (i. e., $M_S = \emptyset$), it is clear that, as for distributed realizability, the number of send and receive events must be balanced in terminating runs. Hence, we may import Cor. 3 from the previous section as a necessary condition for complete realizability. Assuming a choreography that meets this condition, the partial synchronization between send and corresponding receive events is fully reflected in the choreography. This means that, repeating the arguments for Thm. 4, a receive event is always activated in the composition of automata if that is locally the case. Other than this, there are no significant differences in the argument, and we may state:

Corollary 4. *A choreography C for a collaboration $\{P_1, \ldots, P_n\}$ using asynchronous communication is completely realizable if and only if the conditions established in Cor. 3 is satisfied and it is completely realized by $\mathcal{A}(C|_{P_1}), \ldots, \mathcal{A}(C|_{P_n})$,*

The same is true for the case of deadlock free realizability:

Corollary 5. *A choreography C for the collaboration $\{P_1, \ldots, P_n\}$ using asynchronous communication is completely and deadlock freely realizable if and only if he condition established in Cor. 3 is satisfied and it is completely and deadlock freely realized by $\mathcal{A}(C|_{P_1}), \ldots, \mathcal{A}(C|_{P_n})$,*

5.3 Complexity

Checking the condition of Cor. 3 can be done in linear time on any automaton representing C. The projection of C to an individual peer P_i amounts to replacing all events distant to P_i by τ and requires linear time for each P_i. The size of the composition is at most the product of the sizes of the components. Checking language equivalence is PSPACE-complete [7]. We have to leave open whether language equivalence can be done more efficiently in the case where C is checked against the composition of its projections.

For the deadlock free case, the resulting components must be determinized and minimized, with potential exponential blow-up, and the resulting composition must be checked for deadlock freedom which requires linear time in the size of the composition.

6 Related Work

Realizability received much attention in recent literature, see [17] for a survey.

Complete realizability. Alur et al. [1] present necessary and sufficient criteria to deadlock freely realize a choreography specified by a set of message sequence charts (MSCs). Both synchronous and asynchronous communication is supported. Their proposed algorithms are very efficient, but are limited to acyclic choreography specifications, because the used MSC model does not support arbitrary iteration. Salaün and Bultan [16] investigate complete realizability of choreographies specified by collaboration diagrams. The authors express the realizability problem in terms of LOTOS and present a case study conducted with a LOTOS verification tool. Their approach tackles both synchronous and asynchronous communication (using bounded FIFO queues). Collaboration diagrams, however, provide only limited support for repetitive behavior (only single events can be iterated) and choices (events can be skipped, but complex decisions cannot be modeled). Hence, the reduction of the PCP is not applicable. These restrictions also apply to the results of Bultan and Fu [3] in which sufficient conditions for complete realizability of collaboration diagrams are elaborated. A tool to check the sufficient criteria of [3,6] is presented by Bultan et al. [2]. Using this tool, the authors showed that many collaboration diagrams in literature are not completely realizable. In fact, most of these models are, however, distributedly realizable. Realizability of conversation protocols by asynchronously communicating Büchi automata is examined by Fu et al. [6]. The authors define a necessary condition for complete realizability. One of the prerequisites, *synchronous compatibility*, heavily restricts asynchronous communication. Kazhamiakin and Pistore [9] study a variety of communication models and their impact on realizability. They provide an algorithm that finds the "simplest" communication model under which a given choreography can be completely realized.

Other realizability notions. Decker and Weske [4] study realizability of interaction Petri nets. To the best of our knowledge, it is the only approach in which (complete and partial) realizability is not defined in terms of languages. Instead, the authors require the peer implementations and the choreography to be branching bisimilar. This results in a stronger realizability notion which needs further investigations with respect to decidability issues.

We defined distributed realizability in an earlier paper [12], and to the best of our knowledge, this notion was not yet subject of other work. In the same paper, we showed that complete, distributed, and partial realizability can be approached using an algorithm to check for *distributed controllability* [18]. However, undecidability has been shown for this problem recently [19]. This result as such did, however, not directly imply undecidability of partial realizability.

7 Conclusion and Open Problems

We showed that partial realizability is undecidable if at least four (synchronous communication), respectively three (asynchronous communication) peers are involved. The result relies on the capability of expressing arbitrarily large chunks of distant, noninterfering events. This observation could lead, in future work, to decidable subproblems. Furthermore, the case of only two asynchronously communicating peers is left open. Also the case of mixed communication models requires further investigation.

Distributed realizability is decidable. Realizability only depends on the question whether the choreography is closed under the commutation of distant events. An apparent follow-up question would be whether it is possible to cover all specified sequences with *finitely many* implementations.

For complete realizability, we found the choreography projections to the respective peers to be a canonical realization. If that projection does not realize, no one else does. If the projections are transformed into deterministic automata, this result extends to the problem of deadlock free complete realizability.

The decision procedures suggested by our arguments depend on automata minimization, checking language equivalence, and other, trivially implementable checks. Hence, we assume that the decision procedures can be turned into tools with acceptable behavior on relevant instances.

Acknowledgment. We would like to thank Dietrich Kuske for a very helpful briefing in trace theory.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. 29(7), 623–633 (2003)
2. Bultan, T., Ferguson, C., Fu, X.: A tool for choreography analysis using collaboration diagrams. In: ICWS 2009, pp. 856–863. IEEE (2009)
3. Bultan, T., Fu, X.: Specification of realizable service conversations using collaboration diagrams. SOCA 2(1), 27–39 (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. Diekert, V.: The Book of Traces. World Scientific Publishing Co., Inc., River Edge (1995)
6. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)

7. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Inf. Comput. 86(1), 43–68 (1990)
8. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation (November 2005), http://www.w3.org/TR/ws-cdl-10
9. Kazhamiakin, R., Pistore, M.: Analysis of realizability conditions for Web service choreographies. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 61–76. Springer, Heidelberg (2006)
10. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in Web service compositions. In: WWW 2006, pp. 267–276. ACM (2006)
11. Kleijn, H.C.M., Morin, R., Rozoy, B.: Event structures for local traces. Electr. Notes Theor. Comput. Sci. 16(2) (1998)
12. Lohmann, N., Wolf, K.: Realizability is controllability. In: Laneve, C., Su, J. (eds.) WS-FM 2009. LNCS, vol. 6194, pp. 110–127. Springer, Heidelberg (2010)
13. Mazurkiewicz, A.W.: Trace Theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 279–324. Springer, Heidelberg (1987)
14. OMG: Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0, Object Management Group (2009), http://www.omg.org/spec/BPMN/2.0
15. Sakarovitch, J.: The "last" decision problem for rational trace languages. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 460–473. Springer, Heidelberg (1992)
16. Salaün, G., Bultan, T.: Realizability of choreographies using process algebra encodings. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 167–182. Springer, Heidelberg (2009)
17. 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)
18. Wolf, K.: Does my service have partners? In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 152–171. Springer, Heidelberg (2009)
19. Wolf, K.: Decidability issues for decentralized controllability of open nets. In: AWPN 2010. pp. 124–129. CEUR Workshop Proceedings Vol. 643, CEUR-WS.org (2010)
20. Zaha, J.M., Barros, A.P., 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)

Conformance Testing for Asynchronously Communicating Services

Kathrin Kaschner

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
kathrin.kaschner@uni-rostock.de

Abstract. We suggest a black box testing approach to examine conformance for stateful services. Here, we consider asynchronous communication in which messages can overtake each other during their transmission. For testing, we generate partner services that exchange messages with the implementation under test (IUT). From the observations made during testing, we are then able to infer whether the IUT conforms to its specification. We study how partner services need to be designed to serve conformance testing in an asynchronous setting and present an algorithm which generates a complete test suite.

1 Introduction

Modern software systems are more and more composed of a set of loosely-coupled *services*. Thereby, each service implements an encapsulated, self-contained functionality and communicates via message exchange with its *partner services*. For a proper interaction, the *behavior* of the involved services plays a central role; e.g., a whole composition of services may deadlock if a single service fails to send an expected message. To avoid failures, the behavior of a service should be tested thoroughly before it is deployed.

To face this issue, we propose a black box testing approach. That means, we examine the behavior of the implementation under test (IUT) from the partner's perspective without accessing the inner structure of the IUT. But usually, a service is not bound to a fixed set of partner services. Instead, partners may change frequently and can even be created after deployment of the IUT. Since they are unknown at testing time, real partners or their abstract behavioral description (public view) cannot be used for testing. Instead, we *synthesize* partner services and use them as test cases. Their intended purpose is to imitate the behavior of real partners with which the IUT will be potentially confronted in practise. Moreover, they observe the IUT's reactions during a test run such that we are able to conclude whether the interactions are conform to the IUT's specification.

Throughout the paper, we consider services with *stateful* interaction; i.e., the communication follows a more or less complex protocol in which several messages are exchanged between the services. In addition, we assume an *asynchronous* message passing, in which messages can overtake each other. This is motivated by the fact that services usually communicate over the internet, which does not preserve the message order during transmission. In contrast to *synchronous* communication, the sending of an asynchronous message cannot be blocked by the partner, but is executed independently from the receiving. Further, a message may be sent in advance if it is ensured that the partner (if it follows the protocol) will eventually receive it; i.e., between sending and receiving a message,

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 108–124, 2011.

other messages can be exchanged. All these non-trivial aspects need to be taken into account when test partners are synthesized for substituting real partner services.

To automate the test case generation procedure, a formal model of the specification is required. The test partners are then derived from the model. With our tests we focus on the specified behavior only (conformance testing) and do not examine whether the implementation is robust against undesired messages (robustness testing). To offer exhaustiveness, we design the test partners such that each possible behavior defined by the formal specification can be triggered during testing *and* thereby the asynchronous characteristics are taken into consideration (e.g., test partners may send messages in advance for imitating real partners adequately). Moreover, we present a selection algorithm to limit the number of test cases. The resulting test suite is complete in the sense that (1) each detected failure indicates an error in the implementation (soundness) and (2) each failure that can be discovered by any partner derivable from the formal specification can also be detected by a partner of the test suite (exhaustiveness). That way, the behavior of any potential real partner is considered best possible by the created test suite, and we can be confident that the IUT will interact correctly in practice – if it passes the test suite successfully.

This paper is organized as follows: In Sect. 2, we define when an observation made during testing is considered as correct. In Sect. 3, we study how the test partners for conformance testing need to be designed. Based on these considerations, Sect. 4 shows how a complete test suite can be generated. Section 5 summarizes related work and Sect. 6 concludes the paper.

2 Correct Behavior

While the specification can be assumed to be given in a formal description, the implementation is a physical, real object which is not amenable to formal reasoning. However, to deal with implementations in a formal way, we assume for our theory that for any implementation there is a formal model. But we only require its existence, not the model itself. This is common usage when the implementation is seen as a black box and formal testing is conducted [1]. Then, during conformance testing we infer from the observations made during testing, the (unknown) formal model of the IUT and decide with the help of the testing theory whether the implementation complies to its specification.

Formalizing Behavior. To formally reason about service behavior, we use *service automata* [2,3]. They are related to input output transitions systems (IOTS) [1] and I/O automata [4], but perform communication asynchronously via unidirectional *message channels*. When modeling behavior by service automata, we abstract from data. The set of the abstract messages is denoted by $\mathbb{M}$ and is assumed to be finite. The interface of a service is formed by a set of *pins* $\Pi \subseteq \mathbb{M} \times \{\mathbf{i}, \mathbf{o}\}$. Pin $\pi = [m, \mathbf{i}]$ is an *inbound pin* and $\pi = [m, \mathbf{o}]$ is an *outbound pin*. The *dual pin* of $\pi = [m, z]$ is defined as $\overline{\pi} = [m, z']$ with $z' \neq z$. From a set of pins Π we derive a set of *communicating events* $\mathbb{E}^{\Pi}$ such that $!m \in \mathbb{E}^{\Pi}$ iff there is a pin $\pi = [m, \mathbf{o}] \in \Pi$ (sending a message m) and $?m \in \mathbb{E}^{\Pi}$ iff there is a pin $\pi = [m, \mathbf{i}] \in \Pi$ (receiving a message m). *Internal events* (i.e., non-communicating events) are pooled in a set $\mathbb{E}^{\tau}$. The behavior itself is expressed by a finite state machine whose transitions are labeled with communicating events and internal events.

Definition 1 (service automaton [2,3]) *A* service automaton $A = [Q, q_0, \Omega, \Pi, \mathbb{E}^\tau, \delta]$ *consists of a finite set of* states Q*, an* initial state $q_0 \in Q$*, a set of* final states $\Omega \subseteq Q$*, a finite set of* pins $\Pi \subseteq \mathbb{M} \times \{\mathbf{i}, \mathbf{o}\}$ *(with* $\pi \in \Pi$ *implies* $\overline{\pi} \notin \Pi$*), a finite set of* internal events $\mathbb{E}^\tau$ $(\mathbb{E}^\tau \cap \mathbb{E}^\Pi = \emptyset)$ *and a* transition relation $\delta \subseteq Q \times (\mathbb{E}^\Pi \cup \mathbb{E}^\tau) \times Q$.

We write $q \xrightarrow{e} q'$ for $[q, e, q'] \in \delta$. For a state q, we express with $q \xrightarrow{e}$ that there is a state q' with $q \xrightarrow{e} q'$. If q has no successors, we write $q \not\rightarrow$. A state q' is *reachable from a state* q iff there are events $e_1, \ldots, e_n \in \mathbb{E}^\Pi \cup \mathbb{E}^\tau$ and states $q_1, \ldots, q_n \in Q$ with $q \xrightarrow{e_1} q_1 \xrightarrow{e_2} \ldots \xrightarrow{e_{n-1}} q_{n-1} \xrightarrow{e_n} q_n = q'$ or $q = q'$. A state is *reachable* iff it is reachable from the initial state. A non-final state q ($q \notin \Omega$) is a *deadlock state* iff $q \not\rightarrow$ holds. Since specifications with deadlocks are seen as ill-designed, we consider deadlock-free specifications in our theory only.

As an example, service automaton A in Fig. 1(a) is initially waiting for message a or message c. In case c is received, message z is sent back. In case message a is received, A decides non-deterministically (modeled by a branch of two internal τ steps) whether it sends message y and is then waiting for message b, or it sends message x after receiving message b. The set of pins of B, C and D are dual to the set of pins of A. Some pins of C and D are not used by the internal process of C and D.

Formalizing Communication. To formalize the interplay of two services we use the concept of *composition*. In the model, communication is realized by connecting dual pins with a unidirectional channel. That way, a message of shape m is sent via an outbound pin $[m, \mathbf{o}]$ through the corresponding channel. There, m is pending until the service on the other channel's side receives it via its inbound pin $[m, \mathbf{i}]$. Throughout the paper, we consider those services as *composable*, whose pin sets are dual to each other. Note, that the criterion is merely syntactically and independent of the actual behavior. In particular, it does not guarantee that a sent message is indeed received on the other channel's side.

Two composable service automata are also called *partners*. In Fig. 1, service automata B, C and D are partners of A.

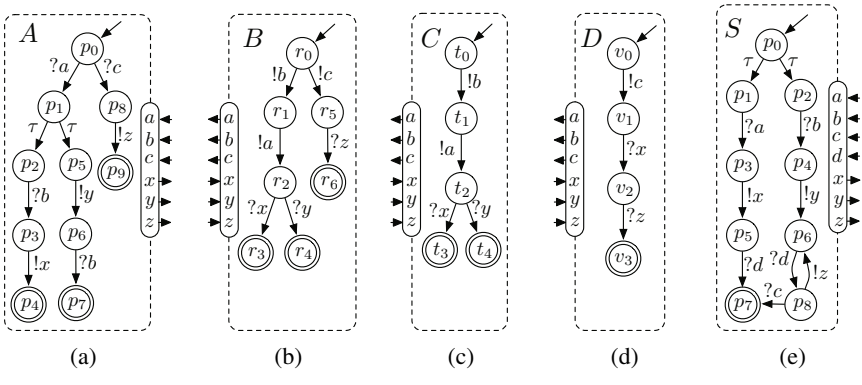


Fig. 1. Five service automata. In the graphical representation initial states are denoted by an incoming arc from nowhere and final states are circled twice. The interface with its inbound and outbound pins is depicted on the dashed box.

Formally, composition of services (see Def. 2) is a product automaton construction, adjusted to the characteristics of asynchronous message passing between the involved services. A state comprises the states of the involved services and the messages pending in the channels, represented by a multiset $\mathcal{B}$. A state $[q_A, q_B, \mathcal{B}]$ with $[q_A, q_B, \mathcal{B}] \not\rightarrow$ is declared as a final state iff $q_A \in \Omega_A$, $q_B \in \Omega_B$ and $\mathcal{B} = [\,]$. Whereas the first two conditions are obvious, the requirement for empty channels at termination is motivated by the fact that messages may contain important information such as payment details. Hence, the sender of such a message wants to ensure that it is actually received rather than ignored. In Def. 2, $Bags(\mathbb{M})$ denotes the set of all multisets over set $\mathbb{M}$, $[\,]$ denotes the empty multiset, and $+$ is the elementwise addition of multisets.

Definition 2 (composition of service automata [2,3]). *The* composition *of two partners A and B ($E_A^{\Pi}, E_B^{\Pi}, E_A^{\tau}$ and E_B^{τ} are pairwise disjoint) is the service automaton $A \oplus B = [Q, q_0, \Omega, \Pi, \mathbb{E}^{\tau}, \delta]$ consisting of $Q := Q_A \times Q_B \times Bags(\mathbb{M})$, $q_0 := [q_{0A}, q_{0B}, [\,]]$, $\Omega := \Omega_A \times \Omega_B \times \{[\,]\}$, $\Pi := \emptyset$, $E^{\tau} := E_A^{\Pi} \cup E_A^{\Pi} \cup E_B^{\tau} \cup E_B^{\tau}$, and δ containing exactly the following elements:*
for all $m \in \mathbb{M}$, $\tau \in E_A^{\tau} \cup E_B^{\tau}$ and $\mathcal{B} \in Bags(\mathbb{M})$,

- $[q_A, q_B, \mathcal{B}] \xrightarrow{!m} [q'_A, q_B, \mathcal{B} + [m]]$, *iff* $q_A \xrightarrow{!m}_A q'_A$ *(send event in A),*
- $[q_A, q_B, \mathcal{B}] \xrightarrow{!m} [q_A, q'_B, \mathcal{B} + [m]]$, *iff* $q_B \xrightarrow{!m}_B q'_B$ *(send event in B),*
- $[q_A, q_B, \mathcal{B} + [m]] \xrightarrow{?m} [q'_A, q_B, \mathcal{B}]$, *iff* $q_A \xrightarrow{?m}_A q'_A$ *(receive event in A),*
- $[q_A, q_B, \mathcal{B} + [m]] \xrightarrow{?m} [q_A, q'_B, \mathcal{B}]$, *iff* $q_B \xrightarrow{?m}_B q'_B$ *(receive event in B),*
- $[q_A, q_B, \mathcal{B}] \xrightarrow{\tau} [q'_A, q_B, \mathcal{B}]$, *iff* $q_A \xrightarrow{\tau}_A q'_A$ *(internal step in A),*
- $[q_A, q_B, \mathcal{B}] \xrightarrow{\tau} [q_A, q'_B, \mathcal{B}]$, *iff* $q_B \xrightarrow{\tau}_B q'_B$ *(internal step in B).*

Figure 2 shows the composition of service automata A and B of Fig. 1. It is free of deadlocks. In contrast, the composition of A and D deadlocks in state $[p_9, v_1, [z]]$ because A never sends message x after receiving message c.

A composition is *k-bounded* iff in all reachable states the number of identical messages in $\mathcal{B}$ does not exceed a value k. This property is motivated by the middleware, that realizes the message exchange between services. Since it has only finite storage available, we limit the capacity of messages in each channel to k, in our theory. The value of k is either known by the middleware's characteristics or chosen carefully. Then, a k-bounded composition does not overflow the channels. As an example, the composition of Fig. 2 is 1-bounded.

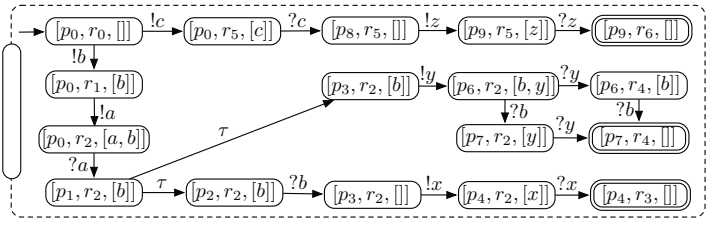


Fig. 2. Composition of A and B of Fig. 1 (only the reachable states are depicted)

Formalizing Observations. As mentioned earlier, we aim at generating partner services for the IUT. During the testing procedure, each generated partner service and IUT are executed in isolation within a test environment. Thereby, we assume that the tester can always observe the IUT's execution status; i.e., still in execution or terminated. A failure is detected as soon as a partner observes an unforeseen reaction of the IUT; i.e., the absence of specified message, sending wrong message or a wrong execution status. Due to the asynchronous setting it is not observable whether the IUT actually receives a message or not. In the following, we define when observations are classified as correct or incorrect. To this end, we give a formal definition of a *(test) run*.

Definition 3 (run). *For a service automaton A, a finite or infinite sequence $\sigma = q_0 e_1 q_1 e_2 q_2 \ldots$ of states and events is called* run *iff q_0 is the initial state of A and $q_{i-1} \xrightarrow{e_i} q_i$ for all $i \in \{1, 2, ...\}$. A run is* maximal *iff it is infinite or it ends in a state q_n with $q_n \not\rightarrow$. The set of all possible runs in A is denoted by Σ_A.*

The notion of a run can be applied to both, a single service automaton and a composition. For example, $\sigma_1 = p_0\, ?a\, p_1\, \tau\, p_5\, !y\, p_6\, ?b\, p_7$ and $\sigma_2 = p_0\, ?a\, p_1$ are runs of service automaton A in Fig. 1(a), and $\varphi_1 = [p_0, r_0, []]\, !c\, [p_0, r_5, [c]]\, ?c\, [p_8, r_5, []]\, !z [p_9, p_5, [z]]\, ?z\, [p_9, r_6, []]$ and $\varphi_2 = [p_0, r_0, []]\, !b\, [p_0, r_1, [b]]\, !a\, [p_0, r_2, [a, b]]$ are runs in the composition $A \oplus B$ of Fig. 2. Thereby, σ_1 and φ_1 are maximal runs.

For a finite run $\sigma = [q_{0_A}, q_{0_B}, []] \ldots [q'_A, q'_B, \mathcal{B}]$ in a composition $A \oplus B$, we define $msg(\sigma)$ as the function that returns the messages pending in the channels after σ; i.e., $msg(\sigma) = \mathcal{B}$. The pending messages after φ_2 are $msg(\varphi_2) = [a, b]$.

During the interaction between an implementation and a test partner, a run is executed in their composition. But, the part proceeded in the implementation is hidden since we cannot access the implementation's structure in black box testing. Consequently, only the part executed in the test partner can be observed and used for the judgment of correctness. Therefore, we establish the notion of a *projected run*. Assume σ is a run in a composition $A \oplus B$. Then, σ *projected* on A, denoted by $\sigma_{\downarrow A}$, reflects the events and states of A during σ. It can easily be seen that $\sigma_{\downarrow A}$ is a run in A. As an example, consider composition $A \oplus B$ in Fig. 2 and run φ_1, mentioned above. Then $\varphi_{1_{\downarrow A}} = p_0\, ?c\, p_8\, !z\, p_9$ and $\varphi_{1_{\downarrow B}} = r_0\, !c\, r_5\, ?z\, r_6$.

With the help of an observed projected run in a test partner, we are able to make assumptions about the implementation's behavior. Thereby, we exploit that (1) each message received by the test partner has been sent by the implementation and (2) the implementation can only receive messages of the test partner. Whether a sent message is indeed received by the implementation cannot be observed in black box testing.

When judging correctness, not only the observed (projected) runs are considered but also the implementation's execution *status*. Thereby, we distinguish three values: `fi`, for terminating in a final state; `ex`, still in execution (i.e., waiting for a message, being before sending or in deadlock) and `inf`, for infinite runs. The latter is only for theoretical considerations because infinite runs cannot be identified in practise. We consciously do not define separate statuses for "receiving message(s)" and "running internal step(s)", since these statuses are transient and will eventually lead to the status `ex` or `fi`. We assume that it is waited long enough, when querying the implementation's execution status.

The function γ (see Def. 4) returns the statuses that can be reached after a run σ without sending a message. Thereby, the messages pending in the channels after σ can be used to proceed σ. Thus, the status after a run may vary depending on the messages that are already sent by the environment.

Definition 4 (status). *Let A be a* service automaton*, let $\sigma \in \Sigma_A$ and let $\mathcal{B}$ be a multiset of messages. In case σ is a finite run, let it end in state q. The* status *after σ is defined by the function $\gamma_A^{\mathcal{B}} : \Sigma_A \to 2^{\{\texttt{fi},\texttt{ex},\texttt{inf}\}}$ as follows:*

- $\texttt{fi} \in \gamma_A^{\mathcal{B}}(\sigma)$*, iff σ is finite and there is a final state q' reachable from q via internal steps and receiving steps $?a$ with $a \in \mathcal{B}$,*
- $\texttt{ex} \in \gamma_A^{\mathcal{B}}(\sigma)$*, iff σ is finite and there is a state q' reachable from q via internal steps or receiving steps $?a$ with $a \in \mathcal{B}$ such that $q' \xrightarrow{!x}$ or $q' \xrightarrow{?b}$ $(b \notin \mathcal{B})$,*
- $\texttt{inf} \in \gamma_A^{\mathcal{B}}(\sigma)$*, iff σ is infinite.*

For example, $\gamma_B^{\mathcal{B}}(\varphi_{1_{\downarrow B}}) = \{\texttt{fi}\}$ and $\gamma_B^{\mathcal{B}}(\varphi_{2_{\downarrow B}}) = \{\texttt{ex}\}$ regardless of the content of $\mathcal{B}$. Now, we are ready to define *observations about an implementation I*.

Definition 5 (recognizable behavior, observation). *For an implementation I and a test partner P, the* behavior of I recognizable by P *is defined by the set $\mathcal{O}_P^I := \{[\sigma, t] \mid$ there is a run $\varphi \in \Sigma_{I \oplus P}$ with $\varphi_{\downarrow P} = \sigma$ and $t = \gamma_I^{msg(\varphi)}(\varphi_{\downarrow I})\}$. The elements of $\mathcal{O}_P^I$ are* observations *about I recognizable by P.*

As an example, we consider A of Fig. 1(a) as an implementation and B of Fig. 1(c) as a test partner. Two possible observations about A recognizable by B are $\text{obs}_1 = [r_0\,!b\,r_1\,!a\,r_2\,?x\,r_3, \texttt{fi}]$ and $\text{obs}_2 = [r_0\,!b\,r_1\,!a\,r_2\,?y\,r_4, \texttt{fi}]$.

An implementation may own decisions that cannot be influenced by the test partner. This means, a test partner is not able to enforce a certain decision. Consequently, during testing not all observations recognizable by a test partner indeed occur. In the example above, B may induce only obs_1 when sending message b - even though the test is repeated. For this purpose, we distinguish two classes of correctness: For *weak correctness* it is sufficient that any observation made of the implementation can be explained by the specification. For *strong correctness* we additionally claim that every possible observation (regarding a given specification) has indeed occurred during testing.

Definition 6 (correctness of recognizable behavior). *Let I be an implementation and S a specification. The behavior of I recognizable by a partner P*

- *is* weak correct *regarding S iff $\mathcal{O}_P^I \subseteq \mathcal{O}_P^S$, and*
- *is* strong correct *regarding S iff $\mathcal{O}_P^I = \mathcal{O}_P^S$.*

The behavior of I recognizable by a (possibly infinite) set $\mathbb{P}$ of partners

- *is* weak correct *regarding S iff for every $P \in \mathbb{P}$ holds: $\mathcal{O}_P^I \subseteq \mathcal{O}_P^S$, and*
- *is* strong correct *regarding S iff for every $P \in \mathbb{P}$ holds: $\mathcal{O}_P^I = \mathcal{O}_P^S$.*

Definition 6 gives a correctness criterion depending on a given set of partners. In the next section, we define a set of partners that is suitable for conformance testing for a given specification S.

3 Conformance Partner

As mentioned above, a deadlock-free composition is fundamental to guarantee a proper interaction between services. It makes no sense to bind services if their composition may deadlock. Consequently, we can assume that in practise the IUT will only be confronted with partners with which a deadlock-free interaction is ensured a priori. In another context, we already proposed deadlock-freely interacting partners as test cases [5]. These partners communicate deadlock-freely per construction with the IUT (supposed it is implemented correctly). However, if a deadlock occurs during testing, a failure is detected. Deadlock-freely interacting partners can be generated automatically from the formal specification.

But in general, this approach is not applicable for conformance testing without further ado. Due to the abstraction, the formal specification may own non-communicated decisions. Since deadlock-freedom after non-communicated decisions cannot be guaranteed, deadlock-freely interacting partners never cover such decisions and the behavior after them. Consequently, the implementation would be tested insufficiently.

This issue is illustrated by specification S in Fig. 1(e): There is a non-communicated decision in state p_0. Thus, a partner does not know whether S is expecting message a or b. Whatever the partner assumes, its communication with S can deadlock. Even though both messages are sent, deadlock-freedom is not guaranteed. For example, partner P_1 (P_2) in Fig. 3 guesses message a (b) is expected. Consequently, the communication can deadlock in state $[p_2, v_1, a]$ ($[p_1, t_1, b]$). In contrast, P_0 in Fig. 3 sends both, a and b. Here, the composition with S deadlocks in state $[p_7, r_5, a]$ and $[p_7, r_7, b]$ since for deadlock-freedom empty channels are required at termination.

As it can easily be seen, each possible partner can deadlock with S. Thus, no test cases could be generated for S using the existing approach [5]. In general, if non-communicated decisions belong to the model, parts of the implementation are not tested using deadlock-freely interacting partners.

Nevertheless, such specifications make sense. Non-communicated decisions are usually caused by a too coarse abstraction when creating the formal model from an

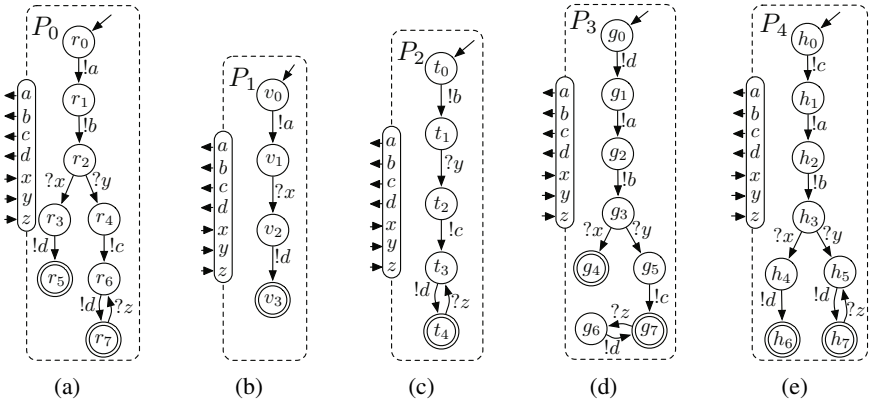


Fig. 3. Partners of the specification S in Fig. 1(e). P_0 and P_3 are conformance partners of S.

(informal) specification. Abstraction is essential to obtain a manageable model. The model in turn is required for automated test case generation. Indeed, non-communicated decisions can be eliminated by refinement. But finding the right level of abstraction is a non-trivial task: If the refinement is too high, the size of the model increases dramatically and makes testing inefficient. Moreover, non-communicated decisions are not as obvious as in the example above. Usually, an extensive analysis is required.

To obviate the problem of refinement, we introduce a new class of partners - the *conformance partners*. With them, non-communicated decisions in the formal specification can be handled and need not be eliminated before test case generation. Thus, thorough testing is possible even though there are non-communicated decisions in the formal specification. In contrast to a deadlock freely interacting partner, a conformance partner is allowed to risk pending messages in the channels at termination time if they are necessary to guarantee the continuation after a non-communicated decision and this non-communicated decision is reached for sure at the point of sending. That means, when a non-communicated decision is inevitable, a conformance partner sends the messages required for all alternatives until it is clear (by receiving a message) how the decision was made. For example, P_0 in Fig. 3(a) is a conformance partner for S in Fig. 1(e). It sends both, a message a and b. Thus, continuation in q_0 is guaranteed, regardless of how S decides. After P_0 has received x or y it knows how the decision was made and behaves appropriately. Since not both alternatives can be executed, either a or b will stay in the channels at termination. That is now permitted thanks to the relaxation of the termination criterion. P_3 is a conformance partner for S, too. It sends message d in advance. That is possible, because d is received independently of the decision and stays in the channel until S reaches state p_5 or p_6 where d is consumed. It is essential to include such partners to test the sending of messages in advance adequately. Similar to P_3, partner P_4 sends message c before it knows which alternative is chosen by S. But in contrast to message d, the receiving of message c is only possible if the right-hand side is chosen. Otherwise, c stays in the channel. That can be avoided, if c is sent after the result of the decision is communicated by message x or y. That is why, c must not stay in the channels at termination. In contrast to a and b, it is not necessary to risk that c is pending in the channels for securing continuation after a non-communicated decision. Thus, we do not classify P_4 as a conformance partner.

These considerations are formalized in the following two definitions. By the notion of *weak receivable* messages we determine whether messages of a channel are allowed to be pending at termination time. In contrast, a channel m is empty after a test run σ if all messages that have been transmitted via m during σ are *strong receivable*. Finally, Def. 8 constitutes the conformance partner.

Definition 7 (weak receivable, strong receivable). *Let A and B be partners. Let σ be a maximal run in B and $\pi = [m, o]$ an outbound pin of B. The messages sent during σ via channel m are* weak receivable *by A iff for each m-event in σ there is a run σ^* with:*

- *σ^* is a prefix of σ containing (at least) the m-event currently considered, and*
- *there is a run φ^* in $A \oplus B$ with $\varphi^*_{\downarrow B} = \sigma^*$ ending in state $[q_A, q_B, \mathcal{B}]$ with $\mathcal{B}(m) = 0$.*

The messages sent during σ via channel m are strong receivable *by A iff after all φ with $\varphi_{\downarrow B} = \sigma$ a state $[q_A, q_B, \mathcal{B}]$ in $A \oplus B$ is reachable such that $\mathcal{B}(m) = 0$.*

As an example: for the run $h_0\,!c\,h_1!a\,h_2\,!b\,h_3\,?x\,h_4\,!d\,h_6$ of P_4 the messages in channel a and d are strong receivable by S, the messages in channel b are weak receivable and the messages in channel c are neither strong receivable nor weak receivable. For the infinite run $h_0\,!c\,h_1!a\,h_2\,!b\,h_3\,?y\,h_5\,!d\,h_7\,?zh_5\,!d\,h_7$ ... of P_4 the messages in channel b, c and d are strong receivable by S and the messages in channel a are weak receivable.

Definition 8 (conformance partner). *Let A and B be partners (A is free of deadlocks). B is a* conformance partner *of A iff*

1. *for all maximal runs σ in B and all m with $[m, \mathbf{o}] \in \Pi_B$ holds: The messages sent during σ via channel m are strong or weak receivable by A,*
2. *for all maximal runs ψ in A and all n with $[n, \mathbf{o}] \in \Pi_A$ holds: The messages sent during ψ via channel n are strong receivable by B,*
3. *for every state $[q_A, q_B, \mathcal{B}]$ in $A \oplus B$ with $[q_A, q_B, \mathcal{B}] \not\rightarrow$ holds: $q_A \in \Omega_A$ and $q_B \in \Omega_B$.*

The set of all conformance partner of A is denoted by Conf(A).

Note, if B is a conformance partner for A then A is not necessarily a conformance partner for B. This asymmetry can be easily derived from the definition by comparing the requirement (1) and (2).

A run σ in a service automaton A is *covered* by a partner B iff there is a run φ in $A \oplus B$ such that $\varphi_{\downarrow A} = \sigma$. We then also say, σ is covered by $\varphi_{\downarrow B}$. Finally, we constitute in Thm. 1 that a specification is fully covered by conformance partners.

Theorem 1. *Every run σ in a service automaton A is* covered *by (at least) one conformance partner of A.*

Proof: Assuming $\sigma \in \Sigma_A$ is the shortest path not covered by any conformance partner; i.e., $\sigma = \sigma^* e q'$ with σ^* is still covered. Consequently, there is a partner $P \in Conf(A)$ such that there is a run $\varphi \in A \oplus P$ with $\varphi_{\downarrow P} \in \Sigma_P$ and $\varphi_{\downarrow A} = \sigma^*$. If e is an internal event or a sending event then σ is also covered by $\varphi_{\downarrow P}$. If e is a receiving event and $e \in msg(\varphi)$ then σ is also covered by $\varphi_{\downarrow P}$. Finally, if e is a receiving event and possibly $e \notin msg(\varphi)$ then we can extend P to a partner P' such that P' sends a message e after σ^* is covered. This does not violate Def. 8 since e is obviously receivable by A. Let ψ be the run in P' that covers σ. As it can be easily seen there is a continuation for ψ such that P' does not violate the requirements of Def. 8. Thus, σ is covered by the conformance partner P'. *q.e.d.*

By requirement (1) in Def. 8 we ensure that the sending of messages in advance is adequately considered by the conformance partners - even if non-communicated decisions are covered. Requirement (2) guarantees that no messages from the specification are pending in the channels at termination time. Thus, specified messages are not ignored by a conformance partner. That is essential for the evaluation of a test run. Requirement (3) ensures, that a finite interaction with a conformance partner always terminates in a final state - if the implementation acts correctly. Based on these considerations together with Thm. 1 we can define *conformance* as follows:

Definition 9 (conformance). *Let S be a specification and I be an implementation. I is* weak (strong) conform *to S iff for every $P \in Conf(S)$ holds: $\mathcal{O}_P^I \subseteq \mathcal{O}_P^S$ ($\mathcal{O}_P^I = \mathcal{O}_P^S$).*

As already mentioned, each channel is able to buffer only a limited number of k messages. For this reason, we restrict ourself to specifications whose composition with each existing conformance partner is k-bounded. Other specifications are not reasonable since the channels can overflow and then the following behavior is not defined.

4 Test Case Generation

As stated in Def. 9, we propose conformance partners as test cases. In this section, we describe how conformance partners are derived systematically from a formal specification S. Therefore, we create a representation - we call it *test guidelines* - that characterizes the set of *all* conformance partners of S. Its construction follows the ideas of the operating guidelines [2] that characterize deadlock-freely interacting partners.

Using all conformance partners as test cases would be too time consuming. Thus, we show in the second part of this section, how a limited number of partners can be extracted from the test guidelines. The resulting test suite is exhaustive in the sense that it is still able to detect each failure that could be discovered by any conformance partner.

Synthesizing Conformance Partners. To generate the test guidelines for a specification S, we first construct a partner that overapproximates the behavior of the conformance partners. Intuitively, a service B has *more behavior* than another service A if it is able to imitate A's behavior such that the environment cannot observe differences between the two services. We formalize this property by the notion of *weak simulation*.

Definition 10 (weak simulation). *Let A and B be service automata. A relation $\varrho \subseteq Q_A \times Q_B$ is a* weak simulation *relation iff*

- $[q_{0_A}, q_{0_B}] \in \varrho$,
- *for all $q_A, q'_A \in Q_A$, $q_B \in Q_B$, and $m \in \mathbb{E}_A^\Pi$ holds: if $[q_A, q_B] \in \varrho$ and $q_A \xrightarrow{m} q'_A$, then there exists a state $q'_B \in Q_B$ with $q_B \xrightarrow{m} q'_B$ and $[q'_A, q'_B] \in \varrho$,*
- *for all $q_A, q'_A \in Q_A$, $q_B \in Q_B$ and $e \in \mathbb{E}_A^\tau$ holds: if $[q_A, q_B] \in \varrho$ and $q_A \xrightarrow{e} q'_A$, then $[q'_A, q_B] \in \varrho$,*
- *for all $q_A, q'_A \in Q_A$, $q_B \in Q_B$ and $e \in \mathbb{E}_B^\tau$ holds: if $[q_A, q_B] \in \varrho$ and $q_B \xrightarrow{e} q'_B$, then $[q_A, q'_B] \in \varrho$,*
- *$[q_A, q_B] \in \varrho$ implies that $q_A \in \Omega_A$ iff $q_B \in \Omega_B$.*

B weakly simulates A *iff there exists a weak simulation relation $\varrho \subseteq Q_A \times Q_B$.*

As an example, the service automaton depicted in Fig. 4(a) weakly simulates (or has more behavior than) P_0, P_1, P_2 and P_3 of Fig 3. The weak simulation relation with P_1 is $\varrho = \{[v_0, q_0], [v_1, q_2], [v_2, q_9], [v_3, q_{19}]\}$.

To synthesize a partner that has more behavior than *any* conformance partner, we consider specification S and the message channels as a black box. That means, their state is only estimated by considering the communication with the partner. Usually, there are several possibilities. They are calculated by the $closure$.

Definition 11 (situation, closure [2,3]). *Let $S = [Q, q_0, \Omega, \Pi, \mathbb{E}^\tau, \delta]$ be a specification and $X \subseteq (Q \times Bags(\mathbb{M}))$. The elements of X are called* situations*. The set $closure_S(X)$ is the smallest set satisfying:*

1. $X \subseteq closure_S(X)$.
2. *If* $[q, \mathcal{B}] \in closure_S(X)$ *and* $q \xrightarrow{e} q'$ *with* $e \in \mathbb{E}^\tau$, *then* $[q', \mathcal{B}] \in closure_S(X)$.
3. *If* $[q, \mathcal{B}] \in closure_S(X)$ *and* $q \xrightarrow{!m} q'$ *with* $!m \in \mathbb{E}^\Pi$, *then* $[q', \mathcal{B} + [m]] \in closure_S(X)$.
4. *If* $[q, \mathcal{B} + [m]] \in closure_S(X)$ *and* $q \xrightarrow{?m} q'$ *with* $?m \in \mathbb{E}^\Pi$, *then* $[q', \mathcal{B}] \in closure_S(X)$.

For a set X of situations, $closure_S(X)$ comprises all situations that can be reached by S without the help of the partner; i.e., by doing internal steps, sending messages or receiving messages already pending in the channels. For example, for S in Fig. 1(e) and $X = \{[p_0, [a]]\}$ we obtain $closure_S(X) = \{[p_0, [a]], [p_1, [a]], [p_2, [a]], [p_3, [\,]], [p_5, x]\}$.

In the following, we define the partner $TS^0(S)$ that has more behavior than any conformance partner of S. Each state q of $TS^0(S)$ consists of those situations that are possible after the events leading to q have occurred.

Definition 12 (conformance partner overapproximation). *Let $S = [Q_S, q_{0_S}, \Omega_S, \Pi_S, \mathbb{E}^\tau_S, \delta_S]$ be a specification. We define the service automaton $TS^0(S) = [Q, q_0, \Omega, \Pi, \mathbb{E}^\tau, \delta]$ with $\Pi = \overline{\Pi_S}$, $E^\tau = \emptyset$, and Q, q_0 and δ inductively as follows:*

base: $q_0 := closure_S(\{[q_{0_S}, [\,]] \mid q_{0_S} \in Q_S\})$ *and* $q_0 \in Q$.
step: For all $q \in Q$ *and* $m \in \mathbb{M}$*:*

1. *There is a transition* $q \xrightarrow{!m} q'$ *and* $q' \in Q$ *iff* $!m \in \mathbb{E}^\Pi$ *and* $q' := closure_S(\{[q_S, \mathcal{B} + [m]] \mid [q_S, \mathcal{B}] \in q\})$ *and* $[q^*, \mathcal{B}] \in q'$ *implies* $\mathcal{B}(m) \leq k$.
2. *There is a transition* $q \xrightarrow{?m} q'$ *and* $q' \in Q$ *iff* $?m \in \mathbb{E}^\Pi$ *and* $q' := closure_S(\{[q_S, \mathcal{B}] \mid [q_S, \mathcal{B} + [m]] \in q\})$.
3. $q \in \Omega$ *iff* *there is* $[q_S, \mathcal{B}] \in q$ *with* $q_S \in \Omega_S$ *and* $m \in \mathcal{B}$ *implies there is* $[m, \mathbf{i}] \in \Pi_S$.

A state q in $TS^0(S)$ has (1) a leaving edge labeled with a sending event $!m$ iff the given message-bound k is not exceeded by the sending of message m and (2) a leaving edge for any receiving event $?m$ (possibly the closure of the successor state q' is empty). Further, state q is defined as a final state (3) iff it contains a situation where S is in a final state and the input channels of $TS^0(S)$ are empty. That means, if $TS^0(S)$ reaches q, S could be terminated and no more messages can be received by $TS^0(S)$.

$TS^0(S)$ overapproximates the behavior of any conformance partner.This can be directly proofed by comparing the conditions of Def. 12 and Def. 8 (conformance partner). To fulfill condition (1) in Def. 12, $TS^0(S)$ sends in every state all possible messages $m \in \mathbb{M}$ (as long as the message-bound is not exceeded). In contrast, a conformance partner is more restricted: Its sent messages must be strong or weak receivable by S (see (1) in Def. 8). Caused by condition (2) in Def. 12, $TS^0(S)$ is *input complete*; i.e., every state has a leaving edge for any receive event. Thus, a conformance partner cannot

take more receive events into account. Finally, the last conditions of Def. 12 and Def. 8 coincide each other. Consequently, $TS^0(S)$ weakly simulates any each conformance partner of S.

To actually construct the test guidelines, we now create a conformance partner from $TS^0(S)$. Currently, $TS^0(S)$ only violates condition (1) of Def. 8 whereas condition (2) and (3) are already fulfilled (see the argumentation above). The behavior of $TS^0(S)$ is complete in the sense that due to the message bound no more edges can be added. Thus, edges must be removed to obtain a conformance partner of S. Since only condition (1) of Def. 8 is violated, $TS^0(S)$ simply owns some send events that are forbidden for conformance partners. They can be identified by the following *labeling function*. Thereby, a *strongly connected component* (SCC) is a maximal set of mutually reachable states, and a *terminal* strongly connected component (TSCC) is an SCC with no leaving eges.

Definition 13 (labeling function). *For $TS^i(S)$ $(i = \{0, 1, \ldots\})$ of a service automaton S we define a labeling function l that assigns to each SCC C in $TS^i(S)$ a set $\mathcal{M} \subseteq Bags(\mathbb{M})$ $(l(C) = \mathcal{M})$ such that $\mathcal{M}$ is maximal according to the following two conditions:*

1. *for each state $q \in C$ and each $[p, \mathcal{B}] \in q$: $\mathcal{M} \leq \mathcal{B}$, and*
2. *for all messages m: $\mathcal{M}(m) \leq max\{l(C')(m) \mid C' \text{ is successor of } C\}$.*

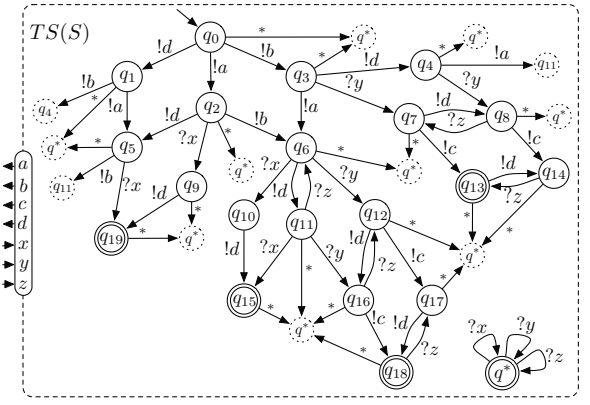
We use the labeling function to analyze which messages sent by $TS^i(S)$ are neither weak nor strong receivable by S. Then, the corresponding edges have to be eliminated to fulfill condition (1) of Def. 8. Thereby, condition (2) and (3) remain fulfilled since only edges labeled with send events are removed. In particular, input completeness is preserved. The procedure is formalized in Def. 14.

Definition 14 (conformance partner synthesis). *Given $TS^i(S)$ $(i \geq 0)$, the service automaton $TS^{i+1}(S)$ is obtained by removing an arc labeled with $!m$ and leading to an SCC C with $l(C)(m) > 0$. Thereby, the removal of an arc includes the removal of all states that become unreachable from the initial state q_0.*

Let $TS(S)$ be $TS^j(S)$ for the smallest j with $TS^j(S) = TS^{j+1}(S)$.

Fig. 4(a) shows $TS(S)$ of the specification S in Fig. 1(e). From the argumentation above follows directly that $TS(S)$ is a conformance partner of S *and* it is the one with the most behavior; i.e., $TS(S)$ weakly simulates any other conformance partner of S. But, not every weakly simulated service automaton is a conformance partner. For example, P_1 and P_2 are simulated by $TS(S)$ in Fig. 4(a). But they are not conformance partners of S as they violate condition (3) of Def. 8. For example, in the composition $S \oplus P_1$, state $[p_2, v_1, [a]]$ has no successors, but neither p_2 nor v_1 is a final state.

To determine whether a weakly simulated automaton P is really a conformance partner, we annotate $TS(S)$ with Boolean formulae. They express which states and transitions of $TS(S)$ must be present in the simulation relation and which parts are allowed to be absent. By the resulting *annotated service automaton* the set of all conformance partners of S is characterized. We call it the *test guidelines* of specification S ($TG(S)$ for short) as it represents all eligible test cases. Due to page limit, we omit the details of



(a)

$\phi(q_0) = !a \vee !b \vee !d$
$\phi(q_1) = !a \vee !b$
$\phi(q_2) = !b \vee !d$
$\phi(q_3) = !a \vee !d$
$\phi(q_4) = !a$
$\phi(q_5) = !b$
$\phi(q_6) = ?x \wedge ?y \vee !d$
$\phi(q_7) = !c \vee !d$
$\phi(q_8) = !c$
$\phi(q_9) = !d$
$\phi(q_{10}) = !d$
$\phi(q_{11}) = ?x \wedge ?y$
$\phi(q_{12}) = !d \vee !c$
$\phi(q_{13}) = !d$
$\phi(q_{14}) = ?z$
$\phi(q_{15}) = true$
$\phi(q_{16}) = !c$
$\phi(q_{17}) = !d$
$\phi(q_{18}) = ?z$
$\phi(q_{19}) = true$
$\phi(q^*) = true$

(b)

Fig. 4. Test guidelines $TG(S)$ of specification S in Fig. 1(e) consisting of (a) conformance partner $TS(S)$ and (b) Boolean formulae. In the graphical representation, dotted states are placeholders for states with the same number. $TS(S)$ is input complete. For not depicted receiving events there is implicitly an edge to state q^*, indicated by an edge labeled with "*". The $closure$ of some states is as follows: $q_0 = \{[p_0, []], [p_1, []], [p_2, []]\}$, $q_1 = \{[p_0, [a]], [p_1, [a]], [p_2, [a]], [p_3, []], [p_5, x]\}$ and $q^* = \emptyset$.

the construction of the Boolean formulae. It follows exactly the ideas of generating operating guidelines [2]. There, Boolean formulae are used to characterize the set of the deadlock-freely interacting partners for a given service.

Finally, a service automaton P is a conformance partner for S iff it is weakly simulated by $TG(S)$ *and* in every state pair $[q_P, q_{TG(S)}]$ of the weak simulation relation state q_P evaluates the boolean formula of $q_{TG(S)}$ to true. The latter means, $\phi(q_{TG(S)})$ is satisfied in the assignment β_{q_P} where $\beta_{q_P}(e) = true$ iff there exists $q'_P \in Q_P$ with $[q_P, e, q'_P] \in \delta_P$.

As an illustrating example, Fig. 4(b) shows the Boolean formulae for $TS(S)$ of the specification S in Fig. 1(e). The formula $\phi(q_0) = !a \vee !b \vee !c$ of the initial state q_0 determines that a conformance partner must have a leaving edge labeled with $!a$, $!b$ or $!c$ (or combinations thereof) at the beginning. Due to the structure of $TS(S)$ a conformance partner is also allowed to have additionally a leaving edge labeled with $?x$, $?y$ or $?z$ (or combinations thereof).

As already mentioned, $[v_0, q_0]$ and $[v_1, q_2]$ are elements of the weak simulation relation between P_1 and $TS(S)$. State v_0 has a leaving edge labeled with $!a$. Thus, v_0 evaluates the formula $!a \vee !b \vee !c$ of q_0 to true. In contrast, v_1 does not fulfill the formula $!b \vee !d$ of state q_2 as it has neither a leaving edge labeled with $!b$ nor a leaving edge labeled with $!d$. Thus, P_1 is not characterized by $TG(S)$. This coincides with the statement above where P_1 was identified as non-conformance partner of S using Def. 8.

Note, the literals in the Boolean formulae of the test guidelines are not always connected by disjunctions. Conjunctions are also possible. For example, the Boolean formula $\phi(q_{11}) = ?x \wedge ?y$ of $TS(S)$ in Fig. 4(a) indicates that a conformance partner must provide both, a leaving edge labeled with $?x$ *and* a leaving edge labeled with $?y$ if the

weak simulation relation "touches" state q_{11}. A conjunction in a Boolean formula is caused by a decision of the specification that cannot be influenced by the partner. Thus, a conformance partner is required to deal with the messages of all possible alternatives to avoid deadlocks in the interaction. In the example, a partner cannot influence whether S in Fig. 1(e) sends x or y.

We are aware that the complexity of the generation of $TG(S)$ is more or less moderate. But due to the strong links, it is comparable with the construction of operating guidelines. With the tool Wendy[1] [6] it was shown that operating guidelines can be calculated in a reasonable time.

Test Case Selection. The test guidelines $TG(S)$ of a specification S characterize all conformance partners of S. Thus, $TG(S)$ can already be seen as a complete test suite for S. However, there are some redundant partners which can be left out without reducing the quality of the test suite.

The behavior of the IUT is evaluated based on the observations made during testing. Consequently, to imitate each conformance partner, we only need to ensure that each observation that is possible by any conformance partner is also possible by one partner of the test suite. This is already fulfilled when selecting the conformance partner $TS(S)$ exclusively as it weakly simulates any conformance partner of S. But $TS(S)$ contains many branches. Thus, it needs to be executed repeatedly for thorough testing. To enforce the different paths, manual adjustments are required before every execution. This is not desirable in automated testing. Instead, we split $TS(S)$ into several small partners $T_1, ..., T_n$ such that

(1) each T_i is a conformance partner,
(2) each run of $TS(S)$ is also considered by (at least) one T_i,
(3) each T_i contains a run that is not considered by any T_j ($i \neq j$), and
(4) each T_i contains as less branches as possible.

Condition (1) and (2) guarantee that the resulting test suite contains only sound test cases (i.e., conformance partners) and is exhaustive regarding Def. 9. By Condition (3), we exclude redundancies among the test cases in the sense that each test partner could make an observation that cannot be made by any other test partner in the test suite. Definition (4) ensures that test partners do not need to be adjusted before execution. Note, to preserve the conformance partner properties not all branches can be eliminated by splitting. However, the remaining branches do not require adjustments. Basically, they are caused by decisions of the implementation that cannot be influenced by the test partner. Thus, a certain alternative cannot be enforced during testing and manually adjustments can be omitted.

The generation of the test suite can be realized by one depth-first search through the test guidelines. Basically, the splitting of the branches is triggered by the Boolean formulae. Due to page limit, we sketch the procedure by the running example only. Consider the test guidelines in Fig. 4(a). The formula "$!a \vee !b \vee !d$" of the initial state demands that any conformance partner has to start with an $!a$, $!b$ or $!d$ event - or combinations thereof. Thereby, the partners containing more than one action can choose which message to send on (test) runtime. In such a case, we can move this runtime decision to the

[1] `http://service-technology.org/wendy`

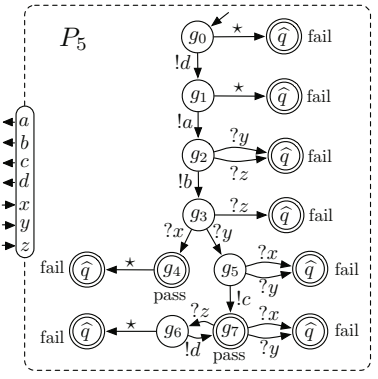


Fig. 5. A test partner synthesized from the test guidelines in Fig. 4. For better readability state $\widehat{q}$ is depicted multiply. An edge labeled with "$\star$" is a placeholder for all receiving events of P_5.

design time of the test partners and only consider those partners consisting of one of the respective events. Consequently, we can split and have three kinds of test cases: either starting with $!a$ or $!b$ or $!d$. By following the $!d$ edge, we reach a state related to the formula "$!a \vee !b$". Thus, we can refine the last kind of test partners to those which either start with $!d$ followed by $!a$ or start with $!d$ followed by $!b$. With the remaining states we proceed in the same manner. Note, we do not split conjunctions. Otherwise, the conformance partner property is not preserved. As an example, see again conformance partner P_3 in Fig. 3(d).

State q^* in $TS(S)$ is a special state as its closure is the empty set (cf. Fig.4(a)). That is, a conformance partner P may contain a state $\widehat{q}$ that is related to q^* in the weak simulation relation, but the specification cannot cover transitions in P leading to $\widehat{q}$. If, however, such a state is reached during testing, the implementation has sent an unspecified message. To observe the implementation best possible, we add such a state $\widehat{q}$ to the test partners and equip each "regular" state with all possible transitions leading to $\widehat{q}$. As soon as $\widehat{q}$ is reached during testing, a failure is detected and we can stop the test run. Thus, we classify $\widehat{q}$ as a final state. During testing, in each state, a test partner tries to execute a transition to $\widehat{q}$ first. If no unspecified message could be received it continues with a "regular" transition. As an example, Fig. 5 depicts a test partner. It is the conformance partner P_3 of Fig. 3(d) enriched with state $\widehat{q}$ and the respective transitions to discover unspecified messages.

To distinguish the intended behavior from unspecified messages easily, we introduce the labels `pass` and `fail` into the test partner. State $\widehat{q}$ is labeled with `fail`. The other final states q_Ω indicate that the specification could reach a final state if a test partner is in q_Ω. Consequently, we label each q_Ω with `pass`. These are the only states where the implementation is allowed to terminate.

For testing conformance we execute the selected partners one after another together with the implementation. A failure is detected if during a test run the implementation behaves unforeseen in any point of time; i.e., a test partner reaches a state labeled with `fail`, the implementation terminates but the test partner is not in a `pass` state, or the

interaction deadlocks (i.e., the test partner gets stuck in a state not labeled with `pass`). For weak conformance it is sufficient to execute each test partner once. In contrast, to establish strong conformance, the test partners are executed repeatedly until every expected observation is recognized. The implementation passes the test suite successfully, if every test run completes with `pass`. On success, we can be confident that it will interact correctly in practice as well.

5 Related Work

There exists a variety of approaches for testing services. A detailed overview is given by Bozkurt et al. [7] and Baresi and Nitto [8]. Test case generation for *synchronous* communicating components was studied in detail by Tretmans (e.g., [9,1]). But due to the different characteristics of message passing, these approaches are not applicable for our asynchronous setting. Others consider asynchronous communication, but with limitations: The approaches of Dranidis et al. [10] and Keum et al. [11] are restricted to services with a communication exclusively consisting of request-response pairs; that is, the sending of a message is directly followed by receiving a message. In our approach, we are more liberal; that is, messages can be sent and received in an arbitrary order. Other approaches (e.g., [12,13,9]) assume asynchronous message passing through an input and an output queue. Here, the sending and receiving is independent, but the messages cannot overtake each other during their transmission. Consequently, these approaches are also not applicable for our setting. To our knowledge, there exists no related work that considers the overtaking of messages adequately when generating test cases for asynchronous communicating components.

6 Conclusion

In this paper, we presented a black-box testing approach for stateful asynchronously communicating services. We formalized correct (asynchronous) communication and introduced conformance partners as test cases. Further, we studied how a limited number of conformance partners can be selected such that the resulting test suite is still complete. That way, we avoid testing with all possible conformance partners without reducing the quality of the test suite.

The introduced theory is independently from a specific language since we use automata as underlying formalism. Thus, the presented test case generation approach is not restricted to services, but also applicable for asynchronous communicating components in other domains; e.g., components of reactive systems or participants in telecommunication systems.

References

1. Tretmans, G.J.: Conformance testing with labelled transition systems: Implementation relations and test generation. Computer networks and ISDN systems 29, 49–79 (1996)
2. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)

3. Lohmann, N.: Correctness of services and their composition. PhD thesis, Universität Rostock / Technische Universiteit Eindhoven, Rostock, Germany / Eindhoven, The Netherlands (2010)
4. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly 2 (1989)
5. Kaschner, K., Lohmann, N.: Automatic test case generation for interacting services. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 66–78. Springer, Heidelberg (2009)
6. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 297–307. Springer, Heidelberg (2010)
7. Bozkurt, M., Harman, M., Hassoun, Y.: Testing web services: A survey. Technical report (2010)
8. Baresi, L., Nitto, E.D. (eds.): Test and Analysis of Web Services. Springer, Heidelberg (2007)
9. Tretmans, G.J.: A Formal Approach to Conformance Testing. PhD thesis, University of Twente, Enschede (1992)
10. Dranidis, D., Kourtesis, D., Ramollari, E.: Formal verification of web service behavioural conformance through testing (2007)
11. Keum, C., Kang, S., Ko, I.-Y., Baik, J., Choi, Y.-I.: Generating Test Cases for Web Services Using Extended Finite State Machine. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 103–117. Springer, Heidelberg (2006)
12. Weiglhofer, M., Wotawa, F.: Asynchronous input-output conformance testing. In: COMPSAC (1), pp. 154–159 (2009)
13. Simao, A., Petrenko, A.: From test purposes to asynchronous test cases. In: ICSTW 2010. IEEE Computer Society (2010)

Programming Services with Correlation Sets

Fabrizio Montesi and Marco Carbone

IT University of Copenhagen, Denmark
{fmontesi,carbonem}@itu.dk

Abstract. Correlation sets define a powerful mechanism for routing incoming communications to the correct running session within a server, by inspecting the content of the received messages. We present a language for programming services based on correlation sets taking into account key aspects of service-oriented systems, such as distribution, loose coupling, open-endedness and integration. Distinguishing features of our approach are the notion of correlation aliases and an asynchronous communication model. Our language is equipped with formal syntax, semantics, and a typing system for ensuring desirable properties of programs with respect to correlation sets. We provide an implementation as an extension of the JOLIE language and apply it to a nontrivial real-world example of a fully-functional distributed user authentication system.

1 Introduction

Correlation sets, introduced by WS-BPEL [17] (BPEL for short), are used to program routing policies for delivering incoming messages to the correct running session within a server. A message is relayed to an internal session whenever a part of its data content matches a part of the session state. These parts are defined by the correlation sets. Correlation sets are widely used in Service-Oriented Computing (SOC) and in web technologies, from complex multiparty interactions to simple client-server protocols between a web browser and a web server. Their role resembles that of unique keys in relational databases: they uniquely identify a session from a portion of their data. Considered in isolation there is little difference between the two concepts. The interesting aspect lies in the interplay with key aspects of SOC, such as distribution and loose coupling. The aim of this paper is to investigate this interplay, in order to gain insight on correlation sets as a programming methodology.

We develop a language for programming correlation-based services. Features of our approach are the direct manipulation of correlation data in programs and the notions of correlation aliasing. The former allows the programmer to write custom policies for instantiating correlation sets from within sessions, whereas the latter defines where correlation data is retrieved inside message content.

We start by analysing some prominent characteristics of SOC. The analysis is used as a foundation for reasoning on the basic constructs of our language and its semantics, whose structure takes inspiration from the π-calculus [14] and SOCK [7]. We establish a typing discipline that prevents the occurrence of some

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 125–141, 2011.

run-time errors, for example ensuring that a service does not break the property that each session is uniquely identifiable through a correlation set. Our results show how to discipline message routing programming based solely on data for obtaining a determinism similar to that of π-calculus-like channels. We demonstrate applicability by providing an implementation of our language – in the form of an extension of the service-oriented language JOLIE [16] – and a nontrivial real-world example showing a fully-functional distributed user authentication system, inspired by the OpenID Authentication specifications [18].

1.1 Key Concepts in Service-Oriented Computing

Distribution. Service-oriented architectures such as Web Services are in most cases distributed over a wide area network, e.g., the Internet. Distribution often features two main aspects: locality and communication asynchrony. The former means that processes run locally at different sites and use the network for communicating with each other. The latter means that messages are received at later points in time wrt when they were sent.

Sessions. Services may engage in multiple, concurrent interactions that may be complex and long-running, each one maintaining a separate execution state. Services support these interactions with *sessions*: stateful instances of workflows. Interactions between sessions are, in practice, dealt with correlation sets, which establish the session a message must be delivered to.

Loose Coupling. Services maintain minimal dependencies among each other, abstracting from internal implementation details. Particular emphasis is put on the types of the functionalities exposed by a service, each one defined by an *operation* and the *structure* of the data to be exchanged through it. The latter is usually defined as a tree using, e.g., the XML language.

Integration. SOC promotes reuse since new services are often implemented by composing already existing ones. Therefore, it is important to offer flexible mechanisms for adapting to the interfaces (e.g. APIs) of preexisting, legacy services and to recent technologies.

Open-Endedness. Service-oriented systems can be open-ended: participants may join or leave the system at run-time. For this reason, it is important to design languages and tools that allow safe execution of systems regardless of the environment they are executed in.

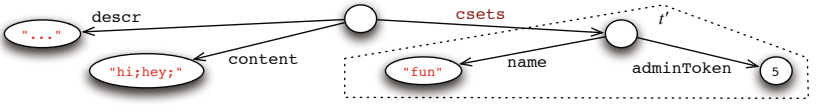
2 Language Overview

In this section, we outline with an example the main ideas of our language against the key concepts of SOC. Formal syntax and semantics will be given in § 3.

Our example is a common distributed scenario with a chat service supporting the management of chat rooms. Chat rooms are identified by name, as in IRC

servers [1]. The service allows users to: create new chat rooms, publish a message in a chat room, retrieve published messages from existing chat rooms, and close chat rooms. When a client requests the creation of a chat room, the service checks that no other room with the same name exists. It then sends an *administration token* back to the invoker. Any client can publish messages in an open chat room or retrieve the history of published messages. The initial creator can close the chat room at any point by using the administration token.

Data Structures. Each chat room has a data structure representing its local state where its name, description, published messages, and administration token are stored. In our language, we represent data as trees where nodes are values of basic data types such as strings and integers. For instance, the state of a chat room is represented by the tree:



(1)

The root has three children pointed to by labels `descr`, `content` and `csets`. Subnode `csets` has two other children, `name` and `adminToken`. Data trees are accessed in programs by means of *paths*. Paths are sequences of edge names separated by dots, and can be used for traversing a tree starting from its root. Paths can be used in assignments and expressions. For example, the tree above could be initialised in our language with the following assignments:

```
descr = "..."; content = "hi;hey;";
csets.name = "fun"; csets.adminToken = 5
```

For brevity, we refer to a path as a variable, and the node it points to as its value. So in this case variable `content` would have value `"hi;hey;"`.

Communication Behaviour. In our language, data is exchanged between services by means of message passing. As in Web Services, messages are labelled by *operations*. Given operations `create`, `publish`, `read` and `close`, we could program the chat service behaviour as:

```
create(name)(csets.adminToken) { csets.adminToken = new };
run = 1; while( run ) {
    [publish(msg)] { content = content + msg.content + ";" }
    [read(req)(content) { 0 }] { 0 }
    [close(req)] { run = 0 }
}
```

The first instruction is an input on operation `create`. The content of the received message (a data tree) will be stored as a subtree of `name` which is a path in the local state. We call this input instruction a session start since its execution will start a new chat. Moreover, it is also a Request-Response (as in WSDL [20]): the client will wait for the server to reply with the content of `csets.adminToken` that is sent back once the local code in curly brackets `{ csets.adminToken = new }` is

executed. `new` is a primitive that returns a locally-fresh token. After invocation, the service enters a loop containing a choice of three inputs with operations `publish` (for publishing in the chat room), `read` (for reading already published messages), and `close` (for closing the chat room). The inputs with operations `publish` and `close` are standard inputs called One-Way while the one with operation `read` is a Request-Response.

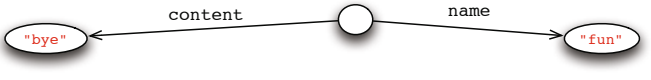
Dually to the server, we can give a sample code for a client:

```
roomName = "MyRoom"; create@Chat(roomName)(adminToken);
msg1.roomName = roomName; msg1.content = "hi";
msg2.roomName = roomName; msg2.content = "hey";
{ publish@Chat(msg1) | publish@Chat(msg2) };
read@Chat(roomName)(chatContent); close@Chat(adminToken)
```

This client sample performs a Solicit-Response output (dual of Request-Response) on operation `create`. The message is sent at location `Chat`, the location of the chat server. Locations (cf. URIs) define where services are deployed, modeling locality. The instruction is completed when the response from the server is received and assigned to `adminToken`. Thereafter, the client sends messages to the chat with two Notification outputs (dual to One-Way) executed in parallel by means of the `|` operator. Finally, the client reads the content of the chat room through operation `read` and closes it by means of operation `close`.

In our language, messages are delivered asynchronously to sessions. After a message is sent, it is guaranteed that the receiving service has buffered it, but not that a session has consumed it. This can lead to bad behaviour. For this reason, the semantics of our language in § 3 preserves ordering of buffered messages.

Correlation Sets and Aliasing. The chat service may have many running sessions executing in parallel, each one representing a chat room. How can it identify the session an incoming message is for, when it receives one from the network? Correlation sets address this issue. In our language, a correlation set is a set of paths, called correlation variables, that define which nodes of a session state identify the session. A correlation set is defined by means of the keyword `cset`. Our chat service has two correlation sets: `cset {name}` and `cset {adminToken}`. For example, if the chat server receives a message carrying the tree:



the first correlation set will then associate the message to the session running with the state shown in (1), since both message and session share the same value for correlation variable `name`, and route the message to it. We call this association *correlation*, and we say that the message *correlates* with the session. The value for correlation variable `name` is stored in the subtree `csets` in the session state. More generally, in our language every correlation value must be put in that subtree. This makes modifications to data that influences correlation explicit. We exploit this aspect in the definition of our type system, in § 4.

Correlation sets are specified by the receiver: the client does not need to be aware of the correlation sets of the invoked service but needs only to send

messages with the expected data structures, enabling loose coupling. Correlation sets are also prone to integrate with existing technologies. For example, a web server session can be identified by the correlation set $c = \{\texttt{sid}\}$, the session id usually stored in a browser cookie.

Above, we associated a message to a session by matching the value of the same path `name` in the message tree and the session state. Such a mechanism is limiting, because the fact that the two paths must be the same means that there is tight coupling between the service implementation and its interface. This could be even completely unfeasible. Consider, for instance, the case in which a programmer must write a service that interacts with a legacy application. The interface of the service will have to be in accordance to what the legacy application expects. Let us assume now that the legacy application will send two different kinds of message to our new chat service, on different operations. The first contains the room name under path `roomName` and the other in the root of the message data tree; this is the behaviour of the client that we showed before. How can we relate both values to the same correlation variable? We address this issue with a notion of aliasing: a correlation variable may be defined together with a list of aliases that tell where to retrieve, in a message, the value to be compared with that of the session, depending on the type of the incoming message (aliasing can be looked at as a type itself). Hence, the correlation set definitions for the chat service become:

```
cset { name:Create Publish.roomName Read }   cset { adminToken:Close }
```

where, for brevity, we assume the input message type of each operation has the same name with an uppercase initial. Data types will be presented in detail in § 5. Correlation aliasing is a key feature for meeting the requirements of integration.

3 Data Structures, Syntax and Semantics

In this section, we formalise data, syntax and give the semantics of our language.

Data Trees and Correlation. Let t range over a set of *data trees* $\mathcal{T}$, with edges denoted by $\mathtt{x}$,$\mathtt{y}$,$\mathtt{z}$,... and nodes denoted by v. v is a value, which can be a string, an integer, a *location* or the undefined value $v_\perp$. Values is the set of all values. In programs, data trees are accessed by *paths*. A path $\mathtt{p}$ is a sequence of tree edges $\mathtt{x}_1.\ \ldots\ .\mathtt{x}_n$ denoting an endofunction on data trees defined as:

$$\mathtt{p}(t) = \begin{cases} t & \text{if } \mathtt{p} = \epsilon \\ \mathtt{p}'(t') & \text{if } \mathtt{p} = \mathtt{x}.\mathtt{p}' \text{ and } \mathtt{x} \text{ is an edge from the root of } t \text{ to } t\text{'s subtree } t' \\ t_\perp & \text{if } \mathtt{p} = \mathtt{x}.\mathtt{p}' \text{ and there is no edge } \mathtt{x} \text{ from } t \text{ to a subtree } t' \end{cases}$$

where ϵ denotes the empty sequence and $t_\perp$ a tree with a single node with value $v_\perp$. We denote the set of possible paths with Paths. Furthermore, we require paths written in programs to be nonempty. We extract the value of the root of a tree by using the function $^\dagger : \mathcal{T} \to \mathsf{Values}$.

A *correlation set* c is a set of paths corresponding to those values that identify a running session of a service: $c \subseteq \mathsf{Paths}$. A service may define more than one correlation set: we denote with C a set of correlation sets, $C \subseteq \mathcal{P}(\mathsf{Paths})$.

We model correlation aliasing by means of an *aliasing function*, α_C, that establishes where to retrieve correlation values in a message received for an operation. Let $\mathcal{O}$ be the set of possible operations, ranged over by $\mathtt{o}$. An aliasing α_C is a function that given an operation $\mathtt{o}$ returns a correlation set $c \in C$ and a function from paths contained in c to paths in the incoming message:

$$\alpha_C : \mathcal{O} \rightharpoonup C \times (\mathsf{Paths} \rightharpoonup \mathsf{Paths})$$

The aliasing function α_C bases aliases on operations, and not on message types like in § 2. This is a matter of convenience: in our language implementation, aliases are defined on message types and are converted exactly to an aliasing function as described in this section.

We now present our definition of correlation in terms of the relation $\vdash_{\alpha_C}$:

Definition 1 (Correlation $\vdash$). *A data tree t' received for operation* $\mathtt{o}$ correlates *with a data tree t with respect to an aliasing α_C, written $t', \mathtt{o} \vdash_{\alpha_C} t$, whenever*

$$\exists c, f.\ c \neq \emptyset \ \wedge\ \alpha_C(\mathtt{o}) = (c, f) \ \wedge\ \forall \mathtt{p} \in c.\ f(\mathtt{p})(t')^\dagger = \mathtt{csets.p}(t)^\dagger \neq v_\perp$$

Syntax. The syntax of programs is structured in three layers. The *behavioural layer* models actions performed by a service session, the *service layer* handles the definition of correlation sets, state and session instantiation and the *network layer* deals with deployment and communication. This layering is reflected in the language implementation, presented in § 5.

Behavioural layer. Behavioural terms are given as processes, ranged over by $P, Q, \ldots$ and defined by the grammar below where r denotes a *channel name*, $l, l', \ldots$ *locations* and $e, e', \ldots$ unspecified first-order expressions that include locations and an operator `new` for generating locally-fresh values:

$$\begin{array}{rlll}
P, Q, \ldots & ::= \sum_i \mathtt{[}\eta_i\mathtt{]}\{P_i\} & \text{(choice)} & \\
& \mid\ \eta & \text{(input)} & \\
& \mid\ \overline{\eta} & \text{(output)} & \\
& \mid\ \mathtt{if(}e\mathtt{)}\,\{P\}\,\mathtt{else}\,\{Q\} & \text{(cond)} & \\
& \mid\ \mathtt{while(}e\mathtt{)}\,\{P\} & \text{(loop)} & \\
& \mid\ \mathtt{p} = e & \text{(assign)} & \\
& \mid\ P; Q & \text{(seq)} & \\
& \mid\ P \mid Q & \text{(par)} & \\
& \mid\ \mathbf{0} & \text{(inact)} &
\end{array}
\qquad
\begin{array}{rll}
\eta & ::= \mathtt{o(p)} & \text{(one-way)} \\
& \mid\ \mathtt{o(p)(p')}\,\{P\} & \text{(request-response)} \\
\overline{\eta} & ::= \mathtt{o@p(p')} & \text{(notification)} \\
& \mid\ \mathtt{o@p(p')(p'')} & \text{(solicit-response)} \\
e & ::= \mathtt{new} & \text{(new)} \\
& \mid\ l & \text{(location)} \\
& \mid\ \ldots & \text{(first-order expr)}
\end{array}$$

Input-guarded branching is available through (choice). Communications can be unidirectional (one-way) or bidirectional (request-response). `o(p)` reads an incoming message for operation `o` and places the received tree in the local state tree under path `p`. Dually, `o@p(p')` sends a message for operation `o` to the location stored in the state node `p` points to, carrying the data in the local state pointed by `p'`. Alternatively, (request-response) and (solicit-response) allow for Request-Response communications. All other constructs are standard.

Service layer. Services, denoted by S, consist of a service behaviour definition (replicated process) and an aliasing α_C, defining correlation sets and aliasings:

$$S ::= \sum_i \texttt{[}\eta_i\texttt{]}\{P_i\} \triangleright_{\alpha_C} \mathbf{0} \quad \text{(service)} \qquad | \qquad \mathbf{0} \triangleright_{\alpha_C} P \cdot t_\perp \cdot \epsilon \quad \text{(starter)}$$

Normally, services become active only after they are invoked. For this reason, a system needs at least one service to spontaneously start invoking other services. We call such a service a *starter*. A starter specifies a single session, which will start its execution without the need to be triggered.

Network layer. Services are *deployed* on locations and composed in parallel to form networks:

$$N, M ::= [S]_l \quad | \quad \boldsymbol{\nu} r\, N \quad | \quad N \mid N \quad | \quad \mathbf{0} \quad \text{(network)}$$

We assume that, for any network, it is never the case that two services are deployed with the same location.

Semantics. We extend the language syntax with run-time terms (as in [8]):

$$\begin{array}{lll} S ::= & P \triangleright_{\alpha_C} I & \text{(running service)} \\ I ::= & P \cdot t \cdot \tilde{m} \quad | \quad I \mid I & \text{(running sessions)} \\ P ::= & \ldots \mid \mathsf{Wait}(r, \mathtt{p}) \mid \mathsf{Exec}(r, \mathtt{p}, P) & \text{(running processes)} \end{array}$$

Services are extended to support multiple locally running sessions (denoted by I). Each session consists of the currently executing run-time process, a state t and a FIFO queue $\tilde{m}$ [8], with ϵ representing the empty queue. m is a message of the form $(r, \mathtt{o}, t)$ where r is a channel, $\mathtt{o}$ an operation and t the content. The terms $\mathsf{Wait}(r, \mathtt{p})$ and $\mathsf{Exec}(r, \mathtt{p}, P)$ model Request-Response communications.

We equip our model with a structural congruence defined as the least congruence relation on P, I and N such that $(\mid, \mathbf{0})$ is a commutative monoid, it supports alpha-conversion, $\mathbf{0}; P \equiv P$, $P \equiv P'$ and $I \equiv I'$ imply $[P \triangleright_{\alpha_C} I]_l \equiv [P' \triangleright_{\alpha_C} I']_l$, $\boldsymbol{\nu} r\, \boldsymbol{\nu} r'\, N \equiv \boldsymbol{\nu} r'\, \boldsymbol{\nu} r\, N$ and such that $(\boldsymbol{\nu} r\, N) \mid N' \equiv \boldsymbol{\nu} r\, (N \mid N')$ if $r \notin \mathtt{cn}(N')$, where $\mathtt{cn}$ is a function that returns the set of channel names in a term.

We give the semantics in terms of a labeled transition system (lts). The labels, ranged over by μ, are standard and their domain is omitted. The *behavioural layer* defines the semantics of service sessions. A selection of the rules is reported below (all rules can be found in the online appendix [2]).

$$\text{(P-Choice)}\ \ j \in J \ \ \eta_j \xrightarrow{\mu} Q_j \ \ \Rightarrow \ \ \textstyle\sum_{i \in J} \texttt{[}\eta_i\texttt{]}\{P_i\} \xrightarrow{\mu} Q_j; P_j$$

$$\text{(P-Solicit)}\ \ \mathtt{o@p(p')(p'')} \xrightarrow{\boldsymbol{\nu} r\ \mathtt{o@p(p')}} \mathsf{Wait}(r, \mathtt{p''}) \qquad \text{(P-Notify)}\ \ \mathtt{o@p(p')} \xrightarrow{r:\mathtt{o(p)}} \mathbf{0}$$

$$\text{(P-Req)}\ \ \mathtt{o(p)(p')}\{P\} \xrightarrow{r:\mathtt{o(p)}} \mathsf{Exec}(r, \mathtt{p'}, P) \qquad \text{(P-OneWay)}\ \ \mathtt{o(p)} \xrightarrow{r:\mathtt{o(p)}} \mathbf{0}$$

$$\text{(P-EndExec)}\ \ \mathsf{Exec}(r, \mathtt{p}, \mathbf{0}) \xrightarrow{\overline{r}\,\mathtt{p}} \mathbf{0} \qquad \text{(P-Wait)}\ \ \mathsf{Wait}(r, \mathtt{p}) \xrightarrow{r\,\mathtt{p}} \mathbf{0}$$

$$\text{(P-Exec)}\ \ P \xrightarrow{\mu} P' \ \Rightarrow\ \mathsf{Exec}(r, \mathtt{p}, P) \xrightarrow{\mu} \mathsf{Exec}(r, \mathtt{p}, P') \qquad \text{(P-Asgn)}\ \ \mathtt{p} = e \xrightarrow{\mathtt{p}=e} \mathbf{0}$$

Rules `P-OneWay` and `P-Notify` allow, respectively, for the receiving and sending of asynchronous one-way messages. Rules `P-Req` and `P-Solicit` do similarly for Request-Response patterns, handling also the subsequent response computation and sending. The computation of the response is handled by rule `P-Exec`; when the response computation terminates, the caller and the callee communicate again by means of the private channel that they established in their interaction. The modeling of Request-Response replies through private channels supports classic client-server communications, where the client could be unable to expose inputs of its own due to external restrictions, e.g. firewalls.

The *service layer* interfaces a session behaviour with the hosting service. Below, we give a selection of the rules (complete table in the appendix [2]):

$$(\texttt{S-Get}) \quad P \xrightarrow{r:\mathtt{o}(\mathtt{p})} P' \Rightarrow P \cdot t \cdot (r, \mathtt{o}, t') :: \tilde{m} \xrightarrow{\tau} P' \cdot t \leftarrow_{\mathtt{p}} t' \cdot \tilde{m}$$

$$(\texttt{S-Send}) \quad P \xrightarrow{\boldsymbol{\nu} r\ \mathtt{o}@\mathtt{p}(\mathtt{p'})} P' \Rightarrow P \cdot t \cdot \tilde{m} \xrightarrow{\boldsymbol{\nu} r\ \mathtt{o}@\mathtt{p}(t)^{\dagger}(\mathtt{p'}(t))} P' \cdot t \cdot \tilde{m}$$

$$(\texttt{S-SR}) \quad P \xrightarrow{\overline{r}\,\mathtt{p}} P' \Rightarrow P \cdot t \cdot \tilde{m} \xrightarrow{\overline{r}\,\mathtt{p}(t)} P' \cdot t \cdot \tilde{m}$$

$$(\texttt{S-RR}) \quad P \xrightarrow{r\,\mathtt{p}} P' \Rightarrow P \cdot t \cdot \tilde{m} \xrightarrow{r\,t'} P' \cdot t \leftarrow_{\mathtt{p}} t' \cdot \tilde{m}$$

$$(\texttt{S-Asgn}) \quad P \xrightarrow{\mathtt{p} = e} P' \Rightarrow P \cdot t \cdot \tilde{m} \xrightarrow{\tau} P' \cdot t \twoheadleftarrow_{\mathtt{p}} e(t) \cdot \tilde{m}$$

$$(\texttt{S-Corr}) \quad t', \mathtt{o} \vdash_{\alpha_C} t \Rightarrow P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} \xrightarrow{\boldsymbol{\nu} r\ \mathtt{o}(t')} P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')$$

$$(\texttt{S-Start}) \quad \frac{t, \mathtt{o} \nvdash_{\alpha_C} I \qquad P \xrightarrow{r:\mathtt{o}(\mathtt{p})} P' \qquad t' = \mathsf{init}(t, \mathtt{o}, \alpha_C)}{P \triangleright_{\alpha_C} I \xrightarrow{\boldsymbol{\nu} r\ \mathtt{o}(t)} P \triangleright_{\alpha_C} I \mid P' \cdot t_{\perp} \leftarrow_{\mathtt{p}} t \leftarrow_{\mathtt{csets}} t' \cdot \epsilon}$$

$$\mathsf{init}(t, \mathtt{o}, \alpha_C) = \begin{cases} t_{\perp} \twoheadleftarrow_{\mathtt{p}_1} f(\mathtt{p}_1)(t) \ldots \twoheadleftarrow_{\mathtt{p}_n} f(\mathtt{p}_n)(t) & \text{if } \alpha_C(\mathtt{o}) = (\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, f) \\ t_{\perp} & \text{if } \mathtt{o} \notin \mathit{Dom}(\alpha_C) \\ \text{undefined} & \text{otherwise} \end{cases}$$

In all rules but `S-Corr` and `S-Start` we have omitted the whole service structure (it is irrelevant for those rules). Rule `S-Start` implements the spawning of a new local session by receiving a message that does not correlate with any running session (thus giving precedence to existing sessions), initialising its `csets` subtree if there is an aliasing definition for operation `o`. Note that the initialisation function $\mathsf{init}(t, \mathtt{o}, \alpha_C)$ is partial and undefined if the message does not contain all the correlation data specified in α_C for `o`; in this case, rule `S-Start` can not be applied. The relation $t', \mathtt{o} \nvdash_{\alpha_C} I$ is defined whenever there is no state t in I such that $t', \mathtt{o} \vdash_{\alpha_C} t$. Moreover, $t \leftarrow_{\mathtt{p}} t'$ is a function that returns a new tree obtained from t by replacing the subtree pointed by `p` with t'; the function automatically creates the missing nodes for traversing t with `p`, initializing them with $v_{\perp}$. Function $t \twoheadleftarrow_{\mathtt{p}} e(t')$ does the same but replaces only $\mathtt{p}(t)$'s root with the value that results from the evaluation of e on t', $e(t')$. Rule `S-Get` allows a running process to fetch the first element from the message queue of its session. Rule `S-Send` propagates the label for a sending, which will be used by the network layer for performing the actual message transmission; the rule substitutes the paths `p` and `p'` in the original label with, respectively, the location pointed by `p` and the data tree pointed by `p'` stored in the session state. Rules `S-Send-Resp` and `S-Recv-Resp` close a Request-Response communication by exchanging the final

reply. Rule `S-Asgn` models variable assignment. Rule `S-Corr` allows a running session to receive a correlating message and store it in its local queue (we omit the condition for handling the special case of an empty queue $\tilde{m} = \epsilon$).

The outer layer of our semantics, the *network layer*, deals with inter-service interactions. The rules are standard and can be found in the appendix [2].

4 Properties and Types

In this section we discuss some desirable properties of services that can be captured with our language. Some of them are based on conditions that need to be guaranteed through the use of a typing system.

Properties. Our properties focus on integrity of sessions and communications.

Property 1 (Message delivery atomicity). Let $N \equiv \boldsymbol{\nu}\tilde{r}\ ([S_1]_{l_1} \mid M)$ *such that* $S_1 \xrightarrow{\boldsymbol{\nu}r'\ \mathtt{o}@l_2(t_M)} S_1'$ *and* $N \xrightarrow{\tau} \boldsymbol{\nu}\tilde{r}\ \boldsymbol{\nu}r'\ ([S_1']_{l_1} \mid M')$. *Then,* $M \equiv [S_2]_{l_2} \mid M''$, $M' \equiv [S_2']_{l_2} \mid M''$ *and either (i)* $S_2 \equiv P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} \wedge S_2' \equiv P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t_M)$ *or (ii)* $S_2 \equiv P \triangleright_{\alpha_C} I \ \wedge \ S_2' \equiv P \triangleright_{\alpha_C} I \mid P' \cdot t \leftarrow_{\mathtt{p}} t_M \cdot \epsilon$ *for some* $t, \mathtt{p}$.

Property 1 states that if a service successfully executes a message sending then there is another service in the network that either (i) put the message in the queue of a correlating session or (ii) started a new session with a state containing the message data. This is guaranteed by our semantics since a message sending is completed only by synchronising with the receiver by means of rule `S-Start` or rule `S-Corr`.

Property 2 (No session ambiguity). For each t', $\mathtt{o}$ *and service* $P \triangleright_{\alpha_C} I$, *there is at most one running session* $P' \cdot t \cdot \tilde{m}$ *in* I *such that* $t', \mathtt{o} \vdash_{\alpha_C} t$.

Our second property states that a service can never have more than one running session that correlates with the same message. Such a situation would lead to non-deterministic assignment of incoming messages, which goes against the principle that a session is *uniquely* identifiable under correlation.

Property 3 (Possible inputs). Let $S \equiv P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m}$. *If* $P' \xrightarrow{r:o(p)} P''$ *then* $\tilde{m} = \tilde{m}' :: (r, \mathtt{o}, t') :: \tilde{m}'' \ \vee \ S \xrightarrow{\boldsymbol{\nu}r\ \mathtt{o}(t')} P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')$.

Property 3 says that if a session needs to perform an input, then a message for that input is in a queue and/or the enclosing service is able to receive a message for the session by correlation. I.e., whenever a session tries to perform an input its state has the related correlation set fully instantiated.

Properties 2 and 3 depend on the states of the sessions running in a service. Bad programming can lead to executions for which the properties do not hold. For example, for $\alpha_C = [\mathtt{join} \mapsto (\{\mathtt{x}\}, [\mathtt{x} \mapsto \epsilon])]$, if the service with behaviour `start(a);csets.x = 5;join(b)` gets invoked twice on `start`, it will spawn two sessions which will both execute `csets.x = 5`. After that, by α_C, both sessions can correlate with a message for operation `join` with value `5` as root node. This

situation breaks property 2 and leads to the non-deterministic routing. Also, if $\alpha_C = [\texttt{join} \mapsto (\{\texttt{x}, \texttt{y}\}, [\texttt{x} \mapsto \epsilon, \texttt{y} \mapsto \texttt{y}])]$, we break property 3: the two sessions would be stuck forever waiting for a message for `join`, because rule `S-Corr` could never be applied due to the lack of a value for `y` in the sessions.

Typing System. We present a type system that focuses on the manipulation of correlation data. Our typing performs an initialisation analysis for correlation variables. Although this is a well-established technique, our setting requires particular attention to the concurrent execution of multiple sessions, and the interplay between session behaviour and the aliasing function.

Typing judgments have the form $\Gamma \vdash P : \Delta_N | \Delta_P$, where $\Delta_N \subseteq \mathsf{Paths}$ and $\Delta_P \subseteq \mathsf{Paths} \times \{\circ, \bullet\}$. Δ_N says which correlation paths *need* to be initialised before P executes and Δ_P contains the correlation paths initialised (*provided*) by P. In Δ_P each correlation path is flagged telling if it carries a fresh value ($\circ$) or not ($\bullet$). The main typing rules follow (all rules and an extended discussion can be found in the appendix [2]).

$$(\texttt{T-CSets-New})\ \frac{}{\Gamma \vdash \texttt{csets.p=new} : \emptyset | \{\texttt{p}^{\circ}\}} \qquad (\texttt{T-CSets-Expr})\ \frac{e \text{ not undefined}}{\Gamma \vdash \texttt{csets.p=}e : \emptyset | \{\texttt{p}^{\bullet}\}}$$

$$(\texttt{T-Seq})\ \frac{\Gamma \vdash P : \Delta_{N_1} | \Delta_{P_1} \quad \Gamma \vdash Q : \Delta_{N_2} | \Delta_{P_2} \quad \Delta' = (\Delta_{N_2} \setminus \Delta_{P_1}) \cup \Delta_{N_1}}{\Gamma \vdash P;Q : \Delta' | \Delta_{P_1} \uplus \Delta_{P_2}}$$

$$(\texttt{T-OneWay})\ \frac{\Gamma(\texttt{o}) = c \neq \emptyset \quad \texttt{p} \neq \texttt{csets.p}'}{\Gamma \vdash \texttt{o(p)} : c | \emptyset} \qquad (\texttt{T-OneWay-Start})\ \frac{(\Gamma \vdash \texttt{o(p)} : c | \emptyset \ \vee\ c = \emptyset) \ \wedge\ \texttt{p} \neq \texttt{csets.p}'}{\Gamma \vdash_s \texttt{o(p)} : c | \emptyset}$$

$$(\texttt{T-Service})\ \frac{\forall j \in J.\ \begin{cases} \Gamma_j \vdash_s \eta_j : c_j | \Delta_{P_j} \ \wedge\ \Gamma_j \vdash P_j : \Delta'_{N_j} | \Delta'_{P_j} \ \wedge\ \Delta'_{N_j} \subseteq c_j \uplus \Delta_{P_j} \ \wedge \\ \forall \texttt{o} \in Dom(\Gamma_j).\, \alpha_C(\texttt{o}) = (\Gamma_j(\texttt{o}), f) \ \wedge\ Dom(f) = \Gamma_j(\texttt{o}) \ \wedge\ \Delta_{P_j} \uplus \Delta'_{P_j} \ltimes C \ \wedge \\ \forall i \in J.\ i \neq j \Rightarrow \Gamma_i \vdash_s \eta_i : c_i | \Delta_{P_i} \ \wedge\ \Gamma_i \vdash P_i : \Delta'_{N_i} | \Delta'_{P_i} \ \wedge\ (\Delta_{P_i} \cup \Delta_{P_j}) \cap c_j = \emptyset \end{cases}}{\emptyset \vdash \sum_{i \in J} [\eta_i]\{P_i\} \triangleright_{\alpha_C} I : \emptyset | \emptyset}$$

The first two rules check the freshness of a correlation variable initialisation. In `T-CSets-Expr` we require e to be defined, i.e. that its evaluation will not yield $v_\perp$. This is a simple (but omitted) definite assignments analysis. Rule `T-Seq` checks that a same correlation variable is not defined multiple times – $\Delta_{P_1} \uplus \Delta_{P_2}$ – and propagates the set of variables that need initialisation before executing $P;Q$. The disjoint union operator $\uplus$ behaves as the union operator $\cup$, but is defined only if the two sets are disjoint. The operator ignores the freshness flag; as such, it is never the case that a same path `p` can be in $\Delta_P \uplus \Delta'_P$ more than once (either with the same flag or with two different flags). We assume the same behaviour for the other sets operators (union, intersection and subtraction). Rule `T-OneWay` uses the environment Γ to register a requirement for the aliasing function of the enclosing service, i.e. that a nonempty correlation set c is associated to operation `o`. It sets $\Delta_N = c$ for the input statement, meaning that the latter requires all the nodes pointed by the paths in c to be initialised in the state before being executed. These requirement are relaxed if the operation is used as a guard for starting a new session, in rule `T-OneWay-Start`, for allowing starting operations to have an empty associated correlation set. Rule `T-Service` checks that a service is well-typed. First, it checks that every branch is well-typed and that the aliasing function α_C complies to the requirements stored in the environment Γ_j of each branch. Then, it checks that for each correlation set $c \in C$ that will be completely defined at least one path in c will point to

a fresh value in the session state, ensuring that sessions will be distinguishable by correlation. This check is performed through $\Delta_P \ltimes C$; relation $\ltimes$ is formally defined below. Finally, the rule forbids different initialisation methods of a same correlation set, i.e. when two different session branches use the same correlation set they must agree if that correlation set is initialised through local assignments or through the message that started the session.

Relation $\ltimes$ captures that, for the sake of being uniquely identifiable by correlation, a session needs at least one correlation variable to be fresh for every correlation set that is completely initialised.

Definition 2 (Correlation set freshness relation $\ltimes$)

$$\Delta_P \ltimes C \quad \textit{iff} \quad \forall c \in C \,.\, (\nexists \mathtt{p} \in c \,.\, \mathtt{p} \notin \Delta_P) \;\Rightarrow\; (\exists \mathtt{p} \in c \,.\, \mathtt{p}^\circ \in \Delta_P)$$

We introduce now a notion of error in the semantics adding two rules. Both use a `wrong` label that carries the location of the originating service. The first rule requires that a service has two running sessions that correlate with the same message t_M for the same operation `o`, the negation of property 2. The second rule, instead, is active when a running session wishes to input on an operation for which there is no message in the queue and the correlation mechanism can route no new message to such a session, the negation of property 3.

$$\text{(S-Wrong-Corr)}\quad \frac{P \triangleright_{\alpha_C} I \mid P' \cdot t' \cdot \tilde{m}' \xrightarrow{\nu\, r\, \mathtt{o}(t_M)} P \triangleright_{\alpha_C} I \mid P' \cdot t' \cdot \tilde{m}' : (r, \mathtt{o}, t_M) \qquad P \triangleright_{\alpha_C} I \mid P'' \cdot t'' \cdot \tilde{m}'' \xrightarrow{\nu\, r'\, \mathtt{o}(t_M)} P \triangleright_{\alpha_C} I \mid P'' \cdot t'' \cdot \tilde{m}'' : (r', \mathtt{o}, t_M)}{[P \triangleright_{\alpha_C} I \mid P' \cdot t' \cdot \tilde{m}' \mid P'' \cdot t'' \cdot \tilde{m}'']_l \xrightarrow{\mathtt{wrong}\, l} [P \triangleright_{\alpha_C} I \mid P' \cdot t' \cdot \tilde{m}' \mid P'' \cdot t'' \cdot \tilde{m}'']_l}$$

$$\text{(S-Wrong-Input)}\quad \frac{P' \xrightarrow{r : \mathtt{o}(p)} P'' \quad \tilde{m} \neq \tilde{m}' : (r, \mathtt{o}, t') : \tilde{m}'' \quad P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} \overset{\nu\, r\, \mathtt{o}(t'')}{\nrightarrow} P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m} : (r, \mathtt{o}, t'')}{[P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m}]_l \xrightarrow{\mathtt{wrong}\, l} [P \triangleright_{\alpha_C} I \mid P' \cdot t \cdot \tilde{m}]_l}$$

We can finally show the main results of our type system:

Theorem 1 (Subject Reduction)
$\Gamma \vdash N : \Delta_N | \Delta_P \;\wedge\; N \xrightarrow{\mu} N' \;\Rightarrow\; \Gamma \vdash N' : \Delta'_N | \Delta'_P$.

Theorem 2 (Safety). *Let l be a location in N.*

1. $\Gamma \vdash N : \Delta_N | \Delta_P \;\Rightarrow\; N \overset{\mathtt{wrong}\, l}{\nrightarrow} N'$
2. $\Gamma \vdash N : \Delta_N | \Delta_P \;\Rightarrow\; N | N' \overset{\mathtt{wrong}\, l}{\nrightarrow} N''$

Note that Theorem 2.2 ensures *local safety*, i.e. that a well-typed network will respect our properties regardless of its context.

5 Language Implementation in JOLIE

We have implemented the techniques described in the previous sections by extending the JOLIE programming language. Our solution is now the official mechanism for programming correlation with JOLIE.

The JOLIE Language. We give a brief description of JOLIE [16,11], an open source [5] fully-fledged service-oriented programming language.

JOLIE programs are composed by a *behavioural part* and a *deployment part*. The syntax of the behavioural part is a superset of the syntax of our behavioural layer. When it comes to correlation, the only relevant difference is that output primitives refer to output ports (described below) rather than using paths for pointing to locations. The deployment part defines *communication ports* and *interfaces*. *Output ports* are used for invoking external services, whereas *input ports* are used to expose locations on which the service can receive messages. A port specifies a location and the data protocol to use (e.g. SOAP [19]). Some JOLIE protocol implementations allow for configuring protocol-specific headers. For example, we can connect cookie values in HTTP to paths in message data trees. As a consequence, programmers can store correlation values as cookies in web browsers that invoke a JOLIE service, thus seamlessly integrating our approach with common web technologies. Interfaces describe the (types of the) operations used in the behavioural part. Each element in an interface couples an operation to its message types, which are structured as trees. For example, the following defines an interface with a Request-Response operation `sum` that takes a tree with two subnodes – `x` and `y` – and returns an integer:

```
type SumRequest:void { .x:int .y:int }
interface SumInterface {RequestResponse:sum(SumRequest)(int)}
```

Correlation Set Definitions. We have implemented the syntax exposed in § 2 for defining correlation sets based on message types. A `cset` block corresponds to the definition of a correlation set c in our model and its related aliases in α_C:

$$\texttt{cset}\ \{\ \textit{list of}\ V\ \} \qquad V ::= \texttt{p} : \textit{list of}\ \texttt{p}_T$$

where $\texttt{p}$ is a path and $\texttt{p}_T$ a path that starts with a message type reference. In our implementation, these definitions are converted to an aliasing function α_C for the interpreter; this is a convenient shortcut: if two operations share a same message type, then the aliasing function generated for the interpreter will have an entry for both operations with the same aliasing specified for that type by the programmer. Furthermore, we implemented a check for verifying that an alias is compatible with the related message type – i.e. if the message type contains the node of interest pointed by $\texttt{p}_T$. Thus we can ensure that a correlation set definition is compatible with the interface provided by the service.

Primitive `new` has been implemented using the Java standard library for generating secure Universally Unique Identifiers (UUID).

Implementation. The JOLIE interpreter is developed in the Java language and is structured in four modules. The *Parsing* module reads a program, produces its related AST (Abstract Syntax Tree), analyzes it and generates an OOIT. An OOIT (*Object-Oriented Interpretation Tree*) executes a session behaviour. The *Runtime Environment* handles the creation of sessions and their execution states. The *Communication Core* handles communications.

We updated the Parsing module for handling the syntax for defining correlation sets, applying our type system and converting `cset` definitions into an aliasing function α_C for the Runtime Environment. The Java class used by the Runtime Environment for controlling the execution of a session has been augmented with a message queue. When a node from the OOIT asks for a message input, the Runtime Environment checks the message queue of its session as spec-

ified by rule `S-Get`. Message queues are filled with incoming messages received by the Communication Core, looking for correlation as defined in rule `S-Corr`. If no correlating session is found, the Runtime Environment is asked to start a new session with the message, cf. `S-Start`. If the session can not be started, a fault `CorrelationError` is sent to the invoker and the message is discarded[1].

Request-Response interactions are supported by means of communication channel objects. The OOIT nodes implementing rules `P-Request` and `P-Solicit` (and their continuation) are given access to the channel of interest and can use it for sending or receiving responses as specified by rules `P-End-Exec` and `P-Wait`, abstracting from the underlying details.

6 Example: A Decentralised Authentication Protocol

We present now an example inspired by the OpenID Authentication specifications [18]. OpenID is a largely adopted Single Sign-On solution based upon a decentralised authentication protocol that allows a service, called *relying party*, to authenticate a user, the *client*, by relying on another external service that is responsible for handling identities, the *identity provider*. When the client requests access to the relying party, the latter opens an authentication session in the identity provider. The client can then send its authentication credentials to the session in the identity provider, which will inform the relying party on the result of the authentication attempt.

We implemented the protocol in the updated version of JOLIE. The example can be downloaded at [3], where we support web browser clients by means of the JOLIE integration with HTTP.

The code below is a sketch of the relying party service:

```
cset { clientToken: ... }
cset { secureToken: AuthMessage.secureToken }
interface RelyingPartyInterface {
OneWay: authSucceeded(AuthMessage), authFailed(AuthMessage)
RequestResponse: login(LoginRequest)(Redirection) }
main {
    login( loginRequest )( redirection ) {
        clientToken = new; secureToken = new;
        openRequest.relyingPartyIdentifier = MY_IDENTIFIER;
        openRequest.clientToken = csets.clientToken;
        openRequest.secureToken = csets.secureToken;
        openAuth@IdentityProvider( openRequest );
        /* ... build redirection message for client ... */
    };  [ authSucceeded( message ) ] { /* ... */ }
        [ authFailed( message ) ]    { /* ... */ }
}
```

[1] For space reasons, we do not report fault semantics in this paper. The implementation is in line with the fault handling semantics of JOLIE, reported in [15,6].

First, the service receives a request on the Request-Response operation `login` from the client for initiating the protocol. The body of `login` generates two fresh tokens: `clientToken`, referred by the first correlation set[2], and `secureToken`, referred by the second one. We will use `clientToken` for receiving messages from the client and `secureToken` for receiving messages from the identity provider. The client is not informed about `secureToken`, preventing it to maliciously act as the identity provider. The body of `login` performs a call to the identity provider, opening an authentication session and communicating `secureToken`. We can now safely reply to the client that invoked operation `login`: property 1, from § 4, guarantees that the session in identity provider has been opened at this point and that the client will therefore find it ready. The reply will redirect the client to the identity provider. The relying party will then wait for a notification about the result of the authentication attempt, hence the input choice on the operations `authSucceeded` and `authFailed`, which correlate through `secureToken`.

We now show the identity provider behavioural code sketch omitting the interface definitions: we assume input types to have their operation name with an initial uppercase letter.

```
cset { relyingPartyIdentifier:
          OpenAuthentication.relyingPartyIdentifier
          Authenticate.relyingPartyIdentifier,
       token: OpenAuthentication.token Authenticate.token }
main {
    openAuth( openRequest ); authenticate( authRequest );
    /* ... verify authentication ... */
    message.secureToken = openRequest.secureToken;
    if ( verified ) { authSucceeded@RelyingParty( message ) }
    else            { authFailed@RelyingParty( message ) }
}
```

The service can start a session with an input on `openAuth` (to be called by the relying party). The operation receives the values for initialising the correlation set, which is composed by two variables: `relyingPartyIdentifier` and `token`. We need both variables because there may be multiple active sessions for handling requests from different relying parties: two relying parties may generate a same value for `token`. We solve this issue by adding the identifier, e.g. a URL, of the relying party to the correlation set. After the session has been opened, we wait for the user credentials on operation `authenticate`. The credentials are verified and the result sent to the relying party.

7 Related Work and Conclusions

Related Work. Previous versions of JOLIE (including SOCK [7]) feature correlation sets where correlation data is manipulated within sessions. However, they

[2] Aliasings for `clientToken` are left unspecified in the relying party implementation sketch, since it will only be used after establishing whether the user can log in.

support no correlation aliasing and no static analysis for identifying bad correlation programming. Even though SOCK features the Request-Response pattern, its semantics does not meet our requirement of integration since the reply is not routed through a private channel. Instead, it is correlated again to the session of the invoker, thus making it similar to an interaction performed by means of two One-Way operations. Moreover, they do not feature multiple correlation sets. All correlation variables are, instead, put in one single correlation set, which does not act as unique session identifier: sessions may be ambiguous under correlation and the related message routing non-deterministic.

Our approach takes inspiration from BPEL [17], which supports multiple correlation sets for identifying sessions. In BPEL, correlation programming is mixed with that of behaviours. Correlation sets are scoped in specific code blocks, and different input activities can use different correlation sets for receiving even if they use the same operation. This makes BPEL programming more error-prone than in our approach, where correlation sets are based on the service interface (its operations) and defined independently from the behaviour. Our language expressiveness is still high, due to correlation data manipulation inside sessions. BPEL does not support correlation programming with a typing discipline, but relies on run-time faults for signaling undesired situations that the programmer specifies manually. Finally, BPEL does not come with formal specifications, leaving much of the burden in handling the complexity of a distributed system to the programmer and the interpreter implementations.

Blite [13] is a model for service orchestration, whose programs can be compiled to BPEL processes [4]. The model is formal, but the final compilation to BPEL makes the approach suffer from the unpredictable behaviour of the execution engine, due to the lack of formality of BPEL specifications. Similarly, the calculus for web services COWS [12] allows to correlate sessions based on channel usage. COWS features several tools for static analyses and an interpreter, however it lacks a fully-fledged language implementation.

[10] provides an implementation of channel-based sessions relying on session types [9,8]. In this setting, message routing does not rely on data transmission.

Conclusions. We have presented a language for programming services with correlation sets, investigating the interplay between some key aspects of SOC and correlation-based programming. Our approach features a direct manipulation of correlation data in programs and a notion of correlation aliasing. We have shown how both aspects can be disciplined by means of a type system. The applicability of our work has been demonstrated by exposing implementations of real-world scenarios where correlation sets can be successfully employed. Our solution has replaced the previous correlation mechanism in the JOLIE language. The features guaranteed by properties 2 and 3 are similar to those provided by private channels in the π-calculus. In our approach different sessions use different instances of correlation sets, much like in the π-calculus replications of a same process use different private channels.

Our semantics for message queues can lead to deadlocks, because a session must consume messages in the same order in which they are received. There are various potential solutions to this problem. For instance, each session could manage a separate queue for each correlation set, or for each operation. Another issue in our model is that it does not handle session garbage collection, i.e. terminated sessions are not removed from their executing service. Handling this aspect is nontrivial, because a terminated session may have some messages left in its queue which must be dealt with. We leave the investigation of these issues to future work, as they are not relevant for the results presented in this paper.

More complex forms of analysis may be developed for correlation. An interesting aspect would be to analyze the behaviour of service networks by introducing behavioural types for participants such as session types [8]. Another topic to be explored is that of security. Programs may be checked to establish that correlation values are not *compromised*.

References

1. Internet Relay Chat Protocol, `http://tools.ietf.org/html/rfc1459`
2. On-line appendix, `http://www.itu.dk/people/fabr/icsoc2011`
3. OpenID implementation, `http://www.jolie-lang.org/files/icsoc2011/openid.zip`
4. Cesari, L., Lapadula, A., Pugliese, R., Tiezzi, F.: A Tool for Rapid Development of WS-BPEL applications. In: SAC, pp. 2438–2442 (2010)
5. Free Software Foundation (FSF). GNU Lesser General Public License, `http://www.gnu.org/licenses/lgpl.html`
6. Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: Dynamic Error Handling in Service Oriented Applications. Fundamenta Informaticae 95(1), 73–102 (2009)
7. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: SOCK: A Calculus for Service Oriented Computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
8. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proc. of POPL 2008, vol. 43(1), pp. 273–284. ACM Press (2008)
9. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
10. Hu, R., Yoshida, N., Honda, K.: Session-based Distributed Programming in Java. In: Ryan, M. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 516–541. Springer, Heidelberg (2008)
11. JOLIE. JOLIE: Java Orchestration Language Interpreter Engine, `http://www.jolie-lang.org/`
12. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 33–47. Springer, Heidelberg (2007)
13. Lapadula, A., Pugliese, R., Tiezzi, F.: A Formal Account of WS-BPEL. In: Wang, A.H., Tennenholtz, M. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 199–215. Springer, Heidelberg (2008)

14. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I and II. Information and Computation 100(1), 1–40, 41–77 (1992)
15. Montesi, F., Guidi, C., Lanese, I., Zavattaro, G.: Dynamic Fault Handling Mechanisms for Service-Oriented Applications. In: Proceedings of ECOWS 2008, pp. 225–234 (2008)
16. Montesi, F., Guidi, C., Zavattaro, G.: Composing Services with JOLIE. In: Proceedings of ECOWS 2007, pp. 13–22 (2007)
17. OASIS. Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/
18. OpenID. OpenID Specifications, http://openid.net/developers/specs/
19. World Wide Web Consortium (W3C). SOAP Specifications, http://www.w3.org/TR/soap/
20. World Wide Web Consortium (W3C). Web Services Description Language, http://www.w3.org/TR/wsdl

Verification of Deployed Artifact Systems via Data Abstraction

Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi

Department of Computing,
Imperial College London, UK

Abstract. Artifact systems are a novel paradigm for specifying and implementing business processes described in terms of interacting modules called *artifacts*. Artifacts consist of *data* and *lifecycle* models, accounting for the relational structure of the artifact state and its possible evolutions over time. We consider the problem of verifying artifact systems against specifications expressed in quantified temporal logic. This problem is in general undecidable. However, when artifact systems are deployed, their states can contain only a bounded number of elements. We exploit this fact to develop an abstraction technique that enables us to verify deployed artifact systems by model checking their bounded abstraction.

1 Introduction

Artifact systems (AS) are a recent paradigm in business process modelling and development [9]. Artifacts are a response to criticisms [4] in the workflow and services literature regarding the fact that services are typically represented and reasoned about by mostly neglecting the data involved in the system. In artifact systems the underlying databases on which services operate are treated with similar emphasis to the processes operating on them. While this responds to the need of more accurate and natural modelling, it results in significant difficulties from a theoretical point of view.

A considerable problem arises when trying to verify formally services specified using artifacts. While verification through model checking and its applications (including orchestration and choreography) are relatively well-understood in the plain process-based modelling, the presence of data makes these problems much harder. Put succinctly, infinite domains in the underlying databases lead to infinite state-spaces for the model checking problem. Infinite domains also call for specification languages that support quantification, e.g., first-order temporal logic. This setting is known to generate an undecidable model checking problem [12], thereby reducing, at least theoretically, the possibility of verifying ASs in their most general setting.

The starting point for the work presented in this paper is that any AS, when deployed, can operationally have only size-bounded (though infinitely many) underlying database states, due to the finiteness of the machine running the system. It follows that any verification problem for ASs deployed on concrete machines can be abstracted by considering bounded abstractions of the theoretically infinite domains. This bound can either be obtained from the size of the memory of the machine running the system, or can be iteratively refined to generate finer abstractions of the concrete system. In the

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 142–156, 2011.

paper we show that under these assumptions the verification problem for ASs is decidable and its complexity, while high, is not vastly dissimilar from that of other applications.

The rest of the paper is organised as follows. In Section 2 we introduce database schemas and a formalisation of AS suitable for verification. Section 3 illustrates the formalisation through a use-case thereby validating the formal machinery. In Section 4 we introduce the verification problem in its general form and point to its undecidability. Section 5 introduces the notion of deployed artifacts and bounded AS, and presents abstraction results leading to the decidability of the verification problem. We conclude by applying the results to the scenario discussed and pointing to further work.

Related Work

Although different approaches to infinite-state model checking have been proposed [15,7,6], we are not aware of results addressing properties that involve the relational structure of the data in each state.

Our approach is largely inspired by [11] and [10], where decidability is achieved by adding syntactic restrictions on both the system description and the specification to verify. These impose a form of *guarded quantification* on variables and limit the access to the values stored as the system evolves. Our work differs from these in that we do not allow quantifiers to occur out of the scope of modal operators in the property to verify; we do not impose any restriction on the system specification; we verify an infinite fragment of the original system; and we consider a branching-time specification language. We do not perceive exploring a fragment of the original system as a limitation, as the explored fragment is in fact *the* deployed system. Moreover, our technique enables a form of (incomplete) verification based on generating successively more refined abstractions of the original system.

Our abstraction technique is based on replacing the actual values of the concrete system with a finite set of *symbolic* values. This approach was previously adopted in [3] in the context of service composition.

Our work is also related to [1], which addresses the orthogonal problem of checking whether an AS introduces only a finite number of new values. The conditions put forward in this work guarantee the underlying model to be finite-state, which is a tighter restriction than in our case. While we focus on properties specified in a first-order extension of CTL, [1] considers a quantified version of the μ-calculus. Although not formally addressed here, we expect our technique to be able to deal with μ-calculus properties, too, thus making our framework a generalisation of [1].

2 Artifacts and Artifact Systems

Broadly speaking, *artifacts* are abstract models of the atomic entities that, by mutually interacting, give rise to a *business process* [9]. They are structures consisting of two parts: a *data model* and a *lifecycle model*. The former captures a fragment of the structure of the information relevant to the process. The latter is a specification of the possible ways such information evolves in response to external or internal events, and

how new events for other artifacts are generated. For our purposes, artifacts can be simply thought of as possibly nested records (i.e., they may contain sets of records as attribute values), equipped with actions that enable changes on their attributes. A characterizing feature of artifacts is the presence of an id and some status attributes. The id field identifies a particular artifact instance, the status fields encode the advancement of the artifact with respect to its lifecycle. Adopting an approach similar to [1], in this paper we formalise artifact systems by means of a database and a set of actions, which account for the artifact data models and the artifact lifecycles, respectively.

Definition 1 (Database schema). *A* (relational) database schema *is a set* $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ *of* relation (or predicate) symbols P_i*, each associated with its arity* $a_i \in \mathbb{N}$.

Definition 2 (Database interpretation). *Given a database schema* $\mathcal{D}$*, a* $\mathcal{D}$-interpretation (or $\mathcal{D}$-instance) over an interpretation domain U *is a mapping* D *associating each relation symbol* P_i *with a* finite a_i*-ary relation over* U*, i.e.,* $D(P_i) \subseteq U^{a_i}$.

The set of all $\mathcal{D}$-interpretations over a given domain U is denoted by $\mathcal{I}_{\mathcal{D}}(U)$. The *active domain* of D, $adom(D)$, is the set of all U-elements occurring in some tuple of some predicate interpretation $D(P_i)$. By definition the active domain $adom(D)$ is finite.

Definition 3 (First-order formulas over $\mathcal{D}$, U, and V). *Given a database schema* $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$*, and two sets* U *and* V *of constant and variable symbols, respectively, the language* $\mathcal{L}_{\mathcal{D},U,V}$ *of first-order formulas* φ *over* $\mathcal{D}$*,* U *and* V *is inductively defined as follows:*

$$\varphi ::= t = t' \mid P_i(\boldsymbol{t}) \mid (\varphi) \mid \neg\varphi \mid \forall x\varphi \mid \varphi \rightarrow \varphi,$$

where $\boldsymbol{t}$ *is an* a_i*-tuple of* terms*, and* t, t' *are* terms*, i.e., elements in* $U \cup V$.

In the rest of the paper we assume V fixed, thus omitting the corresponding subscript. When no ambiguity arise, we also omit U. We use the standard abbreviations $\exists$, $\top$, $\bot$, $\wedge$, $\vee$, and $\neq$. Free and bound variables are defined as standard. For a formula φ we denote the set of its variables as $vars(\varphi)$, the set of its free variables as $free(\varphi)$, and the set of constants occurring in φ as $const(\varphi)$. We write $\varphi(\boldsymbol{x})$ with $\boldsymbol{x} = \langle x_1, \ldots, x_\ell \rangle$ to list explicitly in arbitrary order all the free variables in φ. By slight abuse of notation, we treat $\boldsymbol{x}$ as a set, thus we write $\boldsymbol{x} = free(\varphi)$. A *sentence* is a formula with no free variables. Given a FO-formula $\varphi(\boldsymbol{x})$, and a list of terms $\boldsymbol{t}$ s.t. $|\boldsymbol{x}| = |\boldsymbol{t}| = \ell$, $\varphi(\boldsymbol{t})$ represents the formula obtained from φ by replacing every occurrence of the i-th element in $\boldsymbol{x}$ with the i-th element of $\boldsymbol{t}$, for $i = 1, \ldots, \ell$. Obviously, if $\boldsymbol{t}$ contains only constants, $\varphi(\boldsymbol{t})$ is a sentence. A U*-assignment* is a total function $\sigma : V \mapsto U$. For technical convenience, we implicitly assume that every U-assignment σ is extended to the whole U and is the identity on it, i.e., $\forall u \in U, \sigma(u) = u$. Given an assignment σ, we denote by σ^x_u the U-assignment s.t. $\sigma^x_u(x) \doteq u$ and $\sigma^x_u(x') \doteq \sigma(x')$, for every $x' \in V$ s.t. $x' \neq x$.

Definition 4 (Active-Domain Semantics of FO-formulas). *Given a database schema* $\mathcal{D}$*, an interpretation domain* U*, a* $\mathcal{D}$*-interpretation* D *over* U*, a* U*-assignment* σ*, and a FO-formula* $\varphi \in \mathcal{L}_{\mathcal{D},U}$ *over* $\mathcal{D}$ *and* U *(with* V *being fixed), we inductively define whether* D *satisfies* φ *under* σ*, written* $(D, \sigma) \models \varphi$*, as follows:*

$(D, \sigma) \models t = t'$ *iff* $\sigma(t) = \sigma(t')$;
$(D, \sigma) \models P_i(t_1, \ldots, t_\ell)$ *iff* $\langle \sigma(t_1), \ldots, \sigma(t_\ell) \rangle \in D(P_i)$;
$(D, \sigma) \models (\varphi)$ *iff* $(D, \sigma) \models \varphi$;
$(D, \sigma) \models \neg\varphi$ *iff* $(D, \sigma) \not\models \varphi$;
$(D, \sigma) \models \varphi \rightarrow \psi$ *iff* $(D, \sigma) \not\models \varphi$ *or* $(D, \sigma) \models \psi$;
$(D, \sigma) \models \forall x \varphi$ *iff for every* $u \in adom(D)$, $(D, \sigma^x_u) \models \varphi$.

A formula φ *is* true *in* D, *written* $D \models \varphi$, *iff* $(D, \sigma) \models \varphi$ *for all* U*- assignments* σ.

It can easily be seen that the satisfaction of a formula does not depend on the values that σ assigns to non-free variables. Observe that we are adopting an *active-domain* semantics, i.e., all quantified variables range over the active domain of D. Also, notice that constants are *uninterpreted*, i.e., two constants are equal iff they are the same constant. We can now formally introduce the notion of *artifact system*.

Definition 5 (Artifact System). *An* artifact system *is a tuple* $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, *where:*

- $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ *is a database schema;*
- U *is a (possibly infinite) interpretation domain;*
- D_0 *is an* initial database instance*;*
- Φ *is a finite set of parametric* artifact actions *of the form* $\alpha(\boldsymbol{x}) = \langle \pi(\boldsymbol{y}), \psi(\boldsymbol{z}) \rangle$, *where:*
  - $\boldsymbol{x} = \boldsymbol{y} \cup \boldsymbol{z}$;
  - $\alpha(\boldsymbol{x})$ *is the* action signature *and* $\boldsymbol{x}$ *the set of its* (formal) parameters*;*
  - $\pi(\boldsymbol{y})$ *is the* action precondition*, i.e., a FO-formula over* $\mathcal{D}$ *and* U*;*
  - $\psi(\boldsymbol{z})$ *is the* action postcondition*, i.e., a FO-formula over* $\mathcal{D} \cup \mathcal{D}'$ *and* U*, with* $\mathcal{D}' \doteq \{P'_1/a_1, \ldots, P'_n/a_n\}$*;*

For an action $\alpha(\boldsymbol{x})$ we let $const(\alpha(\boldsymbol{x})) = const(\pi(\boldsymbol{x})) \cup const(\psi(\boldsymbol{x}))$. Moreover, if $|\boldsymbol{x}| = \ell$, an *execution* of $\alpha(\boldsymbol{x})$ with *actual parameters* $\boldsymbol{u} \in U^\ell$, is the *ground* action $\alpha(\boldsymbol{u}) = \langle \pi(\boldsymbol{v}), \psi(\boldsymbol{w}) \rangle$, where $\boldsymbol{v}$ (resp., $\boldsymbol{w}$) is obtained by replacing each $\boldsymbol{y}$ ($\boldsymbol{z}$) component y_i (z_i) with the value occurring in $\boldsymbol{u}$ at the same position as y_i (z_i) in $\boldsymbol{x}$ (observe that such replacements make both $\pi(\boldsymbol{v})$ and $\psi(\boldsymbol{w})$ sentences).

The semantics of an artifact system is given in terms of its possible executions, captured by a Kripke structure, whose states are instances of the database schema and whose transitions correspond to the execution of some action.

Definition 6 (Model of an artifact system). *Given an artifact system* $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, *the* model of $\mathcal{S}$ *is the Kripke structure* $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$, *where:*

- $\Sigma \subseteq \mathcal{I}_\mathcal{D}(U)$ *is the* set of states*;*
- $D_0 \in \Sigma$ *is the* initial state*;*
- $\tau \subseteq \Sigma \times \Sigma$ *is the* transition relation *such that* $\tau(D, D')$ *iff for some action* $\alpha(\boldsymbol{x}) \in \Phi$, *there exists an execution* $\alpha(\boldsymbol{u}) = \langle \pi(\boldsymbol{v}), \psi(\boldsymbol{w}) \rangle$ *such that:*
  - $adom(D') \subseteq adom(D) \cup \{u_1, \ldots, u_\ell\} \cup const(\psi)$;
  - $D \models \pi(\boldsymbol{v})$*; in this case we say that the action is* enabled*;*
  - $D \oplus D' \models \psi(\boldsymbol{w})$, *where* $D \oplus D'$ *is the* $(\mathcal{D} \cup \mathcal{D}')$*-interpretation over* U *s.t. for every* $i \in \{1, \ldots, n\}$, $D \oplus D'(P_i) = D(P_i)$, *and* $D \oplus D'(P'_i) = D'(P_i)$.

As usual, preconditions represent the requirements that a state needs to fulfil in order for an action to be enabled, and postconditions define the possible successors of the state where the action is executed. The former are simply $\mathcal{L}_{\mathcal{D},U}$ FO-sentences to be evaluated against the current state, while the latter are FO-sentences using *unprimed* and *primed* relational symbols from $\mathcal{D}$, that refer to relations in the current and the successor state. Intuitively, given two states (i.e., $\mathcal{D}$-interpretations) D and D', the operator $\oplus$ constructs a new "joint" interpretation, namely $D \oplus D'$, interpreting unprimed relational symbols in D, and primed in D'. Notice that the active domain of $D \oplus D'$ may include values from $adom(D)$, $const(\psi)$, and $\boldsymbol{u}$ only. Thus D' may contain additional values with respect to D', i.e., those from $const(\varphi)$ and $\boldsymbol{u}$, and is necessarily finite. For this reason, given D and $\alpha(\boldsymbol{u})$ the whole set of D-successors is computable.

Since the actual parameters come from an infinite domain, the set of $\mathcal{K}$-states Σ is in general infinite. This may happen even in presence of a fixed bound on the active domain of each state, which, although bounded, may correspond to any of the infinitely many finite subsets of U not exceeding the bound.

3 The Order-to-Cash Business Process

In this section we formalise a business process inspired by an IBM customer example [13] as an AS. Specifically, the *Order-to-Cash* scenario describes the process a product undergoes from order to delivery. It involves a *manufacturer*, some *customers*, and some *suppliers*. The process starts when a customer prepares and submits a *customer purchase order* (CPO), i.e., a list of products the customer needs.

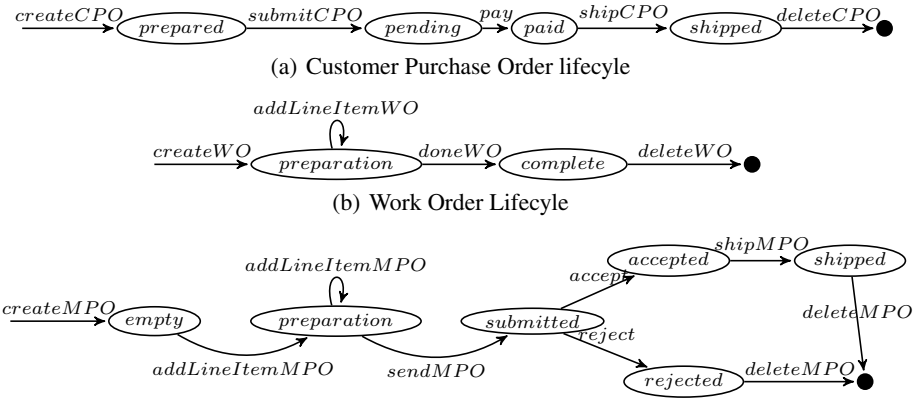
Upon receiving a CPO, the manufacturer first prepares a *work order* (WO), i.e., a list of the components needed to assemble the requested products. Then she selects a possible supplier for each component, prepares one *material purchase order* (MPO) per selected supplier, and submits each MPO to the corresponding supplier.

A supplier can either accept or reject a received MPO. In the former case he delivers the requested components to the manufacturer. In the latter case he notifies the manufacturer of his rejection. If an MPO is rejected, the manufacturer can delete it and prepare and submit new MPOs for the rejected components. When all the components required by a product have been delivered to the manufacturer, she assembles the product and, provided the order has been paid for, delivers it to the customer. Any order (directly on indirectly) related to a CPO can be deleted only after the CPO is deleted.

It is natural to identify 3 classes of artifacts in this process, each corresponding to some of the orders manipulated by the participants (CPO, WO and MPO). An intuitive representation of the artifact lifecycles, capturing only the dependence of actions from the artifact statuses, is shown in Fig. 1. We stress that this is an incomplete representation of the business process, as the interaction between actions and the artifact data content is not represented.

Next, we provide a formal model of the process as an artifact system, where the artifact data models are represented as a relational database schema, and the corresponding lifecycles are formally characterised by an appropriate set of actions.

As to the data model, we reserve a distinguished relation for each artifact class, as well as some auxiliary relations necessary to model the line items present in MPOs



(a) Customer Purchase Order lifecyle

(b) Work Order Lifecyle

(c) Material Purchase Order lifecyle

Fig. 1. Lifecycles of the artifacts involved in the order-to-cash scenario

and WOs. In addition, we introduce *static* relations to store customer, supplier, product, and material information. The resulting database schema $\mathcal{D}$ is shown in Table 1. $R(n_1, \ldots, n_k)$ defines the relation symbol R of arity k (R/k), where n_i is the name of the i-th component, or *attribute*, of its tuples. The relations $Customers$, $Suppliers$, $Products$, and $Materials$, as well as CPO, WO, and MPO, are self-explanatory. Observe the presence of the attribute $status$ in the relations corresponding to artifacts. As to WO_LI and MPO_LI, they contain the line items occurring in WOs and MPOs, respectively. For instance, the fact that two line items containing materials with codes 5 and 6, and quantities 20 and 15, respectively, occur in the WO with $id = 10$, is captured by the presence of tuples $\langle 10, 5, 20\rangle$ and $\langle 10, 6, 15\rangle$ in WO_LI.

Table 1. Database schema $\mathcal{D}$ of the artifact system for the cash-to-order scenario

$$
\begin{gathered}
Customers(id, name),\\
Suppliers(id, name),\\
Products(code, descr),\\
Materials(code, descr),\\
CPO(id, customer\_id, product\_code, status),\\
WO(id, cpo\_id, status),\\
WO\_LI(wo\_id, mat\_code, qty),\\
MPO(id, wo\_id, supplier, status),\\
MPO\_LI(mpo\_id, mat\_code, qty).
\end{gathered}
$$

As interpretation domain, we consider the infinite set U of alphanumeric strings. In the initial database instance D_0 the only non-empty relations are $Customers$, $Suppliers$, $Products$, and $Materials$, which contain background information, such as the possible customers, or a catalogue of available products.

System actions capture *legal* operations on the underlying database and, thus, on artifacts. In Table 2 we report some of their specifications. Variables (from V) and constants (from U) are distinguished by fonts v and c, respectively. We adopt the convention that an action affects only those relations whose name occurs in ψ.

Table 2. Specification of the actions affecting the artifact WO in the order-to-cash scenario

- $createWO(id, cpo) = \langle \pi(id, cpo), \psi(id, cpo) \rangle$, where:
  - $\pi(id, cpo) \equiv \exists code, cid, st\ CPO(cpo, code, cid, st) \wedge$
    $\forall id', c, s\ (WO(id', c, s) \rightarrow id \neq id')$
  - $\psi(id, cpo) \equiv WO'(id, cpo, \mathsf{preparation}) \wedge$
    $\forall id', c, s\ (id \neq id' \rightarrow (WO(id', c, s) \leftrightarrow WO'(id', c, s)))$
- $addLineItemWO(wo, mat, qty) = \langle \pi(wo, mat), \psi(wo, mat, qty) \rangle$, where:
  - $\pi(wo, mat) \equiv \exists cpo\ WO(wo, cpo, \mathsf{preparation}) \wedge \exists desc\ Materials(mat, desc) \wedge$
    $\neg \exists q\ WO\_LI(wo, mat, q)$
  - $\psi(wo, mat, qty) \equiv WO\_LI'(wo, mat, qty) \wedge$
    $\forall w, m, q\ \big((WO\_LI(w, m, q) \rightarrow WO\_LI'(w, m, q)) \wedge$
    $(WO\_LI'(w, m, q) \rightarrow (WO\_LI(w, m, q) \vee (w = wo \wedge m = mat \wedge q = qty)))\big)$
- $doneWO(wo) = \langle \pi(wo), \psi(wo) \rangle$, where:
  - $\pi(wo) \equiv \exists cpo\ WO(wo, cpo, \mathsf{preparation})$
  - $\psi(wo) \equiv \forall w, c, s\ \big((w \neq wo \rightarrow (WO(w, c, s) \leftrightarrow WO'(w, c, s))) \wedge$
    $(WO(wo, c, s) \rightarrow (WO'(wo, c, \mathsf{complete}) \wedge (s \neq \mathsf{complete} \rightarrow \neg WO'(wo, c, s)))))\big)$

Consider the action $createWO$, whose purpose is the creation of a WO-artifact instance. Its precondition requires that cpo is the identifier of some existing CPO, and that id in the new WO is unique with respect to those present when the action is executed. Its postcondition states that, upon execution, the WO relation contains exactly one additional tuple, with identifier attribute set to id, and attribute $status$ set to preparation.

The action $addLineItem$ adds a line item, i.e., a component, to an existing WO. It takes in input the identifier of the WO-artifact (wo), that of the material to add (mat), and the needed quantity (qty). The precondition requires that such parameters correspond to some existing WO-artifact and material, and that the WO-artifact is in state preparation. Moreover, it is required that the material being added is not already present in the WO. The postcondition states that the new line item is added to WO_LI.

As an example of action triggering an artifact's status transition, consider $doneWO$. It is executable only if the WO-artifact is in status preparation and its effect is to set the status attribute to complete.

Notice that although actions are typically conceived to manipulate artifacts of a specific class, e.g., $createWO$ manipulates WO-artifacts, their preconditions and postconditions may depend on artifact instances of different classes, e.g. $createWO$'s precondition depends on CPO-artifacts. We stress that action executability depends not only on the status attribute of an artifact, but on the data content of the whole database, i.e., of all other artifacts. Similarly, action executions do not only affect status attributes.

As actions are executed, the database content, hence the state of each artifact, changes. Obviously, after executing an action, other actions become executable, their executions change the database state, thus make other actions executable, and so on.

4 Verification of Artifact Systems

We focus on the problem of verifying an artifact system against a temporal specification of interest. Since the states of an artifact are characterised by their data content, the atomic components of the specifications need to capture relational properties pertaining

to the states. This, together with the fact that the domain of data may be infinite, makes the problem substantially more challenging than standard model checking [8].

We first introduce syntax and semantics of our specification language.

Definition 7 (Sentence-atomic FO-CTL formulas (over a system $\mathcal{S}$)). *Given an artifact system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, the language $\mathcal{L}_\mathcal{S}$ of* sentence-atomic *FO-CTL formulas over $\mathcal{S}$ is inductively defined as follows:*

$$\varphi ::= \phi \mid (\varphi) \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid AX\varphi \mid A\varphi\mathcal{U}\varphi \mid E\varphi\mathcal{U}\varphi,$$

where ϕ is an FO-sentence from $\mathcal{L}_{\mathcal{D},U}$.

The notions of free and bound variables extend in the obvious way to $\mathcal{L}_\mathcal{S}$, as well as functions $vars$, $free$, and $const$. Observe that formulas in $\mathcal{L}_\mathcal{S}$ are in fact sentences, as all of their atomic components are FO-sentences. We use the standard abbreviations $EX\varphi \equiv \neg AX\neg\varphi$, $AF\varphi \equiv A\top\mathcal{U}\varphi$, $AG\varphi \equiv \neg E\top\mathcal{U}\neg\varphi$, $EF\varphi \equiv E\top\mathcal{U}\varphi$, and $EG\varphi \equiv \neg A\top\mathcal{U}\neg\varphi$.

In order to define the semantics of $\mathcal{L}_\mathcal{S}$, we first define *runs* on a Kripke structure.

Definition 8 ($\mathcal{K}$-runs). *Given a Kripke structure $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$ of an artifact system $\mathcal{S}$, a* $\mathcal{K}$-run *r from a $\mathcal{K}$-state $D \in \Sigma$ is an infinite sequence of $\mathcal{K}$-states $r = D^0 \rightarrow D^1 \rightarrow \cdots$ such that $D^0 = D$ and $\tau(D^i, D^{i+1})$, for $i \geq 0$. For every run r and $i \geq 0$, we define $r(i) \doteq D^i$.*

The semantics of $\mathcal{L}_\mathcal{S}$ formulas is provided in terms of the model $\mathcal{K}$ of $\mathcal{S}$.

Definition 9 (Semantics of $\mathcal{L}_\mathcal{S}$). *Consider a system $\mathcal{S}$ and its model $\mathcal{K}$. Given a formula $\varphi \in \mathcal{L}_\mathcal{S}$ and a $\mathcal{K}$-state $D \in \Sigma$, the satisfaction relation $\models$ is inductively defined as follows:*

- $(\mathcal{K}, D) \models \varphi$ *iff* $D \models \varphi$, *if φ is an FO-sentence;*
- $(\mathcal{K}, D) \models (\varphi)$ *iff* $(\mathcal{K}, D) \models \varphi$;
- $(\mathcal{K}, D) \models \neg\varphi$ *iff* $(\mathcal{K}, D) \not\models \varphi$;
- $(\mathcal{K}, D) \models \varphi \rightarrow \psi$ *iff* $(\mathcal{K}, D) \not\models \varphi$ *or* $(\mathcal{K}, D) \models \psi$;
- $(\mathcal{K}, D) \models AX\varphi$ *iff for all $\mathcal{K}$-runs r s.t. $r(0) = D$,* $(\mathcal{K}, r(1)) \models \varphi$;
- $(\mathcal{K}, D) \models A\varphi\mathcal{U}\psi$ *iff for all $\mathcal{K}$-runs r s.t. $r(0) = D$,* $\exists k \geq 0$ *s.t.* $(\mathcal{K}, r(k)) \models \psi$ *and* $\forall j$ *s.t.* $0 \leq j < k$, $(\mathcal{K}, r(j)) \models \varphi$;
- $(\mathcal{K}, D) \models E\varphi\mathcal{U}\psi$ *iff for some $\mathcal{K}$-run r, $r(0) = D$,* $\exists k \geq 0$ *s.t.* $(\mathcal{K}, r(k)) \models \psi$, *and* $\forall j$ *s.t.* $0 \leq j < k$, $(\mathcal{K}, r(j)) \models \varphi$.

A formula φ is true *in $\mathcal{K}$, written $\mathcal{K} \models \varphi$, if $(\mathcal{K}, D_0) \models \varphi$. We say that $\mathcal{S}$ satisfies φ, written $\mathcal{S} \models \varphi$, if $\mathcal{K} \models \varphi$.*

In the following we describe some properties of the artifact system introduced in Section 3 to model the order-to-cash business process. The first one requires that a product can be shipped to a customer only if all the required materials are already shipped to the manufacturer:

$$\varphi_{ship} = AG\,\forall c\,\big(shippedCPO(c) \rightarrow \forall m\,(related(c,m) \rightarrow shippedMPO(m))\big),$$

where: $shippedCPO(x) \equiv \exists c, p\ CPO(x, c, p, \mathsf{shipped})$ and $shippedMPO(x) \equiv \exists w, sp\ MPO(x, w, sp, \mathsf{shipped})$, respectively, capture the fact that the CPO and the MPO with $id = x$ are in status shipped; and $related(x, y) \equiv \exists c, p, s\, CPO(x, c, p, s) \wedge \exists w, s\, WO(w, x, s) \wedge \exists sp, st\, MPO(y, w, sp, st)$ holds iff the MPO with $id = y$ is related, via some WO, to the CPO with $id = x$.

The next property captures the existence of a run containing a state whose active domain exceeds a given size-threshold t:

$$\varphi_{t+} = EF\ \exists x_1, \ldots, x_{t+1} \bigwedge_{i \neq j} x_i \neq x_j.$$

We can also express the fact that from some state there exists a way to achieve some goal (although this may not necessarily happen). For instance, the next formula states that there exists always a way to empty all non-static relations:

$$\varphi_{empty} = AG\ EF\ (emptyCPO \wedge emptyWO \wedge emptyMPO),$$

where: $emptyCPO \equiv \neg\exists i, c, p, s\, CPO(i, c, p, s)$, $emptyWO \equiv \neg\exists i, c, s\, WO(i, c, s) \wedge \neg\exists w, m, q\, WO\_LI(w, m, q)$, and $emptyMPO$ is similar to $emptyWO$.

Specifications such as those above are useful to describe properties of ASs. Typically, we are interested in checking automatically whether they are satisfied on particular systems. If we consider the AS described previously in the order-to-cash scenario, it is not difficult to see that φ_{ship}, φ_{t+}, and φ_{empty} are satisfied.

Observe that while we may, and in fact can, ascertain the truth of those specifications on this specific example, we cannot be able to do so automatically on any possible system, as the general model checking problem is undecidable. It is therefore natural to investigate decidable subclasses of this problem.

4.1 The General Problem

We are interested in exploring the model checking problem for artifact systems. Formally, this amounts to checking whether an artifact system $\mathcal{S}$ satisfies a specification $\varphi \in \mathcal{L}_{\mathcal{S}}$, i.e., $\mathcal{S} \models \varphi$. It can be shown that the problem is decidable for U finite, and undecidable otherwise.

To see the former, observe that if U contains only finitely many distinct elements, its model contains a finite set of states, whose (relational) data content can be captured by a finite set of propositions (namely, one proposition per fact). By quantifier elimination φ can be transformed into an equivalent propositional (CTL) temporal formula, whose propositions are ground atoms from $\mathcal{L}_{\mathcal{S}}$. This corresponds to reducing the whole problem to standard model checking, which is known to be decidable [8].

For the latter, we have the following result.

Theorem 1. *The model checking problem for artifact systems is undecidable.*

Proof (Sketch). The theorem can be proven by showing that every Turing machine T whose tape contains an initial input I can be simulated by an artifact system $\mathcal{S}_{T,I}$, and that the problem of checking whether T terminates on that particular input can be reduced to checking whether $\mathcal{S}_{T,I} \models \varphi$, where φ encodes the termination condition. The detailed construction is similar to that of Th. 4.10 in [11].

The theorem essentially states the general impossibility of checking the correctness of an artifact system's design, when the interpretation domain is infinite. As this assumption is typically fulfilled in practice, this is a considerably negative result. In the following, rather than focusing on the design, we explore conditions on the concrete implementation of a system that yield decidability and enable the reduction of the verification task to standard model checking of finite-state systems.

5 Verification of Deployed Artifact Systems

Artifact systems serve as a theoretical model for systems to be implemented and deployed on actual machines [9,14]. It is therefore of interest, and of particular relevance in practice, to investigate the model checking problem for such concrete implementations. Observe that any running system can use only the finite, bounded memory provided by the machine it is deployed on (e.g., corresponding to all virtual and physical memory of the server). We show in the following that in this concrete setting the model checking problem against FO-CTL specifications is decidable. Precisely, we show that given a size-bound on the number of values a machine can store at each state, it is decidable whether the artifact system executed on a machine with that bound satisfies the specification. This result can be used to perform a particular form of data abstraction on artifact-systems that guarantees, in limited cases, the preservation of specifications between abstract and concrete systems.

We start this analysis by defining formally the model of an artifact system deployed on a concrete machine.

Definition 10 (b-bounded model of a system $\mathcal{S}$). *Consider a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, its model $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$, and a bound $b \in \mathbb{N}$ such that $b \geq |adom(D_0)|$. The* b-bounded model of $\mathcal{S}$ *is the Kripke structure $\mathcal{K}_b = \langle \Sigma_b, D_0, \tau_b \rangle$, such that:*

- $\Sigma_b \doteq \{D \in \Sigma$ *such that* $|adom(D)| \leq b\}$;
- $\tau_b \doteq \{\langle D, D' \rangle \in \tau$ *such that* $D, D' \in \Sigma_b\}$.

Roughly speaking, $\mathcal{K}_b$ is a sub-model of $\mathcal{S}$, obtained from $\mathcal{K}$ by considering only those $\mathcal{K}$-runs whose states do not exceed the size-bound b. Intuitively, bounded models capture the possible executions of $\mathcal{S}$ on a machine able to accommodate at most b elements. Notice that because the artifact system may still reach infinitely many states, verifying $\mathcal{K}_b$ against a specification φ via an exhaustive visit of its state space is not a viable approach. Nonetheless, we next show that a finite-state, *abstract* model $\hat{\mathcal{K}}_{b,\varphi}$ capturing all the features of $\mathcal{K}_b$ relevant to φ can be constructed. Specifically, we demonstrate that verifying $\hat{\mathcal{K}}_{b,\varphi}$ against φ is equivalent to verifying $\mathcal{K}_b$ against φ. This will show that the verification of *deployed* ASs is actually decidable. In practice, $\hat{\mathcal{K}}_{b,\varphi}$ is defined indirectly, as the b-bounded model of an *abstract* system $\hat{\mathcal{S}}_{b,\varphi}$ obtained from $\mathcal{S}$, b, and φ, as follows.

Definition 11 ((b, φ)-bounded abstract system). *Given a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, a sentence-atomic FO-CTL sentence $\varphi \in \mathcal{L}_{\mathcal{S}}$, and a bound $b \geq |adom(D_0)|$, the* (b, φ)-bounded abstract system of $\mathcal{S}$ *is the system $\hat{\mathcal{S}}_{b,\varphi} = \langle \mathcal{D}, \hat{U}, D_0, \Phi \rangle$, where $\hat{U} = C_{\mathcal{S},\varphi} \cup \hat{C}$, and:*

- $C_{\mathcal{S},\varphi} = const(\varphi) \cup \bigcup_{\phi \in \Phi} const(\phi) \cup adom(D_0)$;
- $\hat{C}$ *is any set of* symbols *s.t.:*
  - $\hat{C} \cap C_{\mathcal{S},\varphi} = \emptyset$,
  - $|\hat{C}| = b + v$, *with* $v = \max_{\phi \in \Phi}\{|vars(\phi)|\}$.

As it turns out, $\hat{\mathcal{S}}_{b,\varphi}$ is an artifact system analogous to $\mathcal{S}$, except for the interpretation domain. Specifically, $\hat{U}$ contains all the constants mentioned in $\mathcal{S}$ or in φ, plus $b + v$ additional symbols. Intuitively, these symbols are used to *simulate* the database content at each state, as well as the new values that actions may introduce upon execution. In particular, at least b distinct symbols are required for the former and v for the latter. Observe that by preserving (the identity of) all mentioned constants, the FO-formulas occurring in $\mathcal{S}$ and φ need no syntactic transformation to preserve their semantics.

As anticipated above, since $\hat{U}$ is finite, checking $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$ is decidable. Below, we show that $\hat{\mathcal{S}}_{b,\varphi}$ contains enough information to check the bounded model of the original artifact system $\mathcal{S}$ against the specification φ.

Theorem 2. *Consider a system $\mathcal{S}$ with U infinite, a bound $b \geq |adom(D_0)|$, and a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$. If $\hat{\mathcal{S}}_{b,\varphi}$ is the (b,φ)-bounded abstract system of $\mathcal{S}$ then $\mathcal{K}_b \models \varphi \Leftrightarrow \hat{\mathcal{K}}_{b,\varphi} \models \varphi$, where $\mathcal{K}_b$ is the b-bounded model of $\mathcal{S}$ and $\hat{\mathcal{K}}_{b,\varphi}$ is the b-bounded model of $\hat{\mathcal{S}}_{b,\varphi}$.*

The theorem shows that instead of checking $\mathcal{K}_b \models \varphi$ (where $\mathcal{K}_b$ is infinite), we can check $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$. Since $\hat{\mathcal{K}}_{b,\varphi}$ is finite, this essentially corresponds to a standard model checking problem, thus any technique for this is also effective for the verification of the b-bounded model of $\mathcal{S}$.

Next, we sketch the main steps of the proof of Theorem 2. We essentially show that $\hat{\mathcal{K}}_{b,\varphi}$ is a sound abstraction of the bounded model of $\mathcal{S}$, that is, it retains enough information to carry out the verification task. Our approach is inspired by the decidability proof of verification of *input-bounded* ASM$^+$s presented in [11]. Interestingly, differently from that, the assumption of size-boundedness allows us to conclude that $\mathcal{K}_b$ and $\hat{\mathcal{K}}_{b,\varphi}$ are *bi-similar*, thus enabling verification of branching-time properties. Firstly, we define when two $\mathcal{D}$-instances are *isomorphic*.

Definition 12 (C-isomorphic $\mathcal{D}$-instances). *Two $\mathcal{D}$-instances D and $\hat{D}$, respectively over U and $\hat{U}$, are said C-isomorphic, for $C \subseteq U, \hat{U}$, written $D \sim_C \hat{D}$, iff there exists a bijection $i : adom(D) \cup C \mapsto adom(\hat{D}) \cup C$ that is the identity on C, and such that for every $j = 1, \ldots, n$, and for every $\boldsymbol{u} \in adom(D)^{a_i}$, $D \models P_j(\boldsymbol{u}) \Leftrightarrow \hat{D} \models P_j(i(\boldsymbol{u}))$, where $i(\boldsymbol{u}) \doteq \langle i(u_1), \ldots, i(u_{a_j})\rangle$.*

The relation $\sim_C$ can be shown to be an equivalence relation.

The following rephrases a well-known result.

Proposition 1. *Given two $\mathcal{D}$-instances D and $\hat{D}$, respectively over U and $\hat{U}$, an FO-sentence φ from $\mathcal{L}_{\mathcal{D},U}$, and a set $C \subseteq U, \hat{U}$ such that $const(\varphi) \subseteq C$, if $D \sim_C \hat{D}$, then*

$$D \models \varphi \Leftrightarrow \hat{D} \models \varphi.$$

This is the main ingredient that allows us to prove Theorem 2. It states that if two $\mathcal{D}$-instances are C-isomorphic, they are indistinguishable by any sentence over $\mathcal{D}$ containing only constants from C. We can now define when two Kripke structures are *bi-similar*.

Definition 13 (C-bisimilar Kripke structures). *Given two Kripke structures $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$ and $\hat{\mathcal{K}} = \langle \hat{\Sigma}, \hat{D}_0, \hat{\tau} \rangle$, with $\Sigma \subseteq \mathcal{I}_{\mathcal{D}}(U)$ and $\hat{\Sigma} \subseteq \mathcal{I}_{\mathcal{D}}(\hat{U})$, and a finite set of constants $C \subseteq U, \hat{U}$, $\mathcal{K}$ and $\hat{\mathcal{K}}$ are said* C-bisimilar*, written $\mathcal{K} \approx_C \hat{\mathcal{K}}$, iff there exists a relation $R \subseteq \Sigma \times \hat{\Sigma}$, called* C(-preserving) bisimulation*, s.t. $\langle D_0, \hat{D}_0 \rangle \in R$, and if $\langle D, \hat{D} \rangle \in R$ then:*

- *$D \sim_C \hat{D}$;*
- *for all D' s.t. $\tau(D, D')$ there exists $\hat{D}'$ s.t. $\hat{\tau}(\hat{D}, \hat{D}')$ and $\langle D', \hat{D}' \rangle \in R$;*
- *for all $\hat{D}'$ s.t. $\hat{\tau}(\hat{D}, \hat{D}')$ there exists D' s.t. $\tau(D, D')$ and $\langle D', \hat{D}' \rangle \in R$.*

When $\langle D, \hat{D} \rangle \in R$, we say that D and $\hat{D}$ are C-bisimilar (with respect to $\mathcal{K}$ and $\hat{\mathcal{K}}$), written $D \approx_C \hat{D}$.

Since by definition $D \approx_C \hat{D}$ implies $D \sim_C \hat{D}$, by Proposition 1, for every FO-sentence φ over $\mathcal{D}$ and U such that $const(\varphi) \subseteq C$, $D \models \varphi \Leftrightarrow \hat{D} \models \varphi$. Observe that the atoms of a FO-CTL sentence are FO-sentences and can thus be evaluated at each state of a Kripke structure. Therefore, we have the following result.

Lemma 1. *Given $\mathcal{K}$, $\hat{\mathcal{K}}$, and C as above, for every sentence-atomic FO-CTL formula φ over $\mathcal{D}$ and U such that $const(\varphi) \subseteq C$, if $D \in \Sigma$ and $\hat{D} \in \hat{\Sigma}$ are such that $D \approx_C \hat{D}$, then*

$$(\mathcal{K}, D) \models \varphi \Leftrightarrow (\hat{\mathcal{K}}, \hat{D}) \models \varphi.$$

Proof (Sketch). By induction on the structure of φ.

In other words sentence-atomic FO-CTL formulas containing only constants from C do not distinguish among C-bisimilar Kripke structures. As a consequence, an infinite-state Kripke structure can be verified against a sentence-atomic FO-CTL formula by verifying any other C-bisimilar structure, including a finite one, against the same specification.

The final step of the proof consists in showing that the b-bounded model of $\mathcal{S}$, $\mathcal{K}_b$, which is infinite-state in general, is $C_{\mathcal{S},\varphi}$-bisimilar to $\hat{\mathcal{K}}_{b,\varphi}$, which is, instead, finite by construction (see Def. 11).

Lemma 2. *Consider a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$ and a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$. Fix a bound $b \geq |adom(D_0)|$, let $\mathcal{K}_b$ be the b-bounded model of $\mathcal{S}$, $\hat{\mathcal{S}}_{b,\varphi} = \langle \mathcal{D}, \hat{U}, D_0, \Phi \rangle$ the (b, φ)-bounded abstract system of $\mathcal{S}$, and $\hat{\mathcal{K}}_{b,\varphi}$ its b-bounded model. Then, for $C_{\mathcal{S},\varphi}$ as in Def. 11, $\mathcal{K}_b \approx_{C_{\mathcal{S},\varphi}} \hat{\mathcal{K}}_{b,\varphi}$.*

Proof (Sketch). The proof consists in constructing a particular $C_{\mathcal{S},\varphi}$-bisimulation between $\mathcal{K}_b$ and $\hat{\mathcal{K}}_{b,\varphi}$. The result then follows.

Lemmas 1 and 2 allow us to prove Theorem 2; so we achieve decidability of the problem in presence of a known bound.

Obviously, not all specifications satisfied by $\hat{\mathcal{K}}_{b,\varphi}$ are preserved in the original (unbounded execution of) $\mathcal{S}$. For instance, consider the specification $\varphi_{t-} \doteq \neg\varphi_{t+}$, with φ_{t+} as in Section 3, which expresses the fact that all states of every run contain at most t distinct elements. This is clearly satisfied by $\hat{\mathcal{K}}_{t,\varphi_{t-}}$, but not by $\mathcal{S}$. On the other hand, preservation is guaranteed for existential specifications. Precisely, let $\mathcal{L}^E_{\mathcal{S}} \subseteq \mathcal{L}_{\mathcal{S}}$ be the sublanguage of sentence-atomic FO-ECTL formulas φ, inductively defined as:

$$\varphi ::= \phi \mid (\varphi) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid EX\varphi \mid E\varphi\mathcal{U}\varphi,$$

where ϕ is a FO-sentence. Then, we have the following result:

Theorem 3. *Given a system $\mathcal{S}$, a bound $b \geq |adom(D_0)|$, and a sentence-atomic FO-ECTL formula $\varphi \in \mathcal{L}^E_{\mathcal{S}}$, if $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$ then $\mathcal{S} \models \varphi$.*

Thus, a sound (though incomplete) technique to check whether $\mathcal{S} \models \varphi$ for $\varphi \in \mathcal{L}^E_{\mathcal{S}}$ consists in iteratively increasing b and checking whether $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$. If at some point the check is successful, then $\mathcal{S} \models \varphi$. For instance, to check whether the AS $\mathcal{S}$ of Section 3 contains a run with some state exceeding a size-threshold T, one can iteratively check whether $\hat{\mathcal{K}}_{b,\varphi_{T+}} \models \varphi_{T+}$ by increasing b at each iteration, starting with $b = T + 1$. In this particular case, it is easy to see that for any T the check is successful for $b = T+1$, as the system is unbounded. A similar approach can be adopted to find counterexamples of *universal* specifications (i.e., negations of existential ones) such as φ_{ship}. There are obvious correspondences with the technique of Bounded Model Checking [5]. Here, however, the bound is on the size of the states, rather than the length of runs.

If $\mathcal{S}$ is size-bounded itself (and the bound is known), all specifications are preserved from the abstract to the concrete model and viceversa. If this is not the case, however, nothing can be said with respect to specifications that are neither universal nor existential. For instance, by checking that $\hat{\mathcal{K}}_{b,\varphi_{empty}} \models \varphi_{empty}$ we cannot conclude anything in general about $\mathcal{S} \models \varphi_{empty}$. That is, our technique fails in proving that $\mathcal{S} \models \varphi_{empty}$ (although we know that this holds). However, by changing the bound b, we can check whether the property holds on any deployed instance of $\mathcal{S}$.

We now analyse the time-complexity of our technique, by considering the cost of reducing the problem of checking whether $\mathcal{K}_b \models \varphi$ to standard CTL model checking. Given $\mathcal{S}$, b, and φ, this essentially requires: building $\hat{\mathcal{K}}_{b,\varphi}$ (from $\hat{\mathcal{S}}_{b,\varphi}$); transforming φ into a propositional CTL formula φ_p, by recursively replacing each formula of the form $\forall x\varphi(x)$ with $\bigwedge_{\hat{u}\in\hat{U}} \varphi(\hat{u})$; and then applying an algorithm for CTL model checking, to check whether $\hat{\mathcal{K}}_{b,\varphi} \models \varphi_p$. This gives the following result.

Theorem 4. *Given an artifact system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi\rangle$, a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$, and a bound $b \geq |adom(D_0)|$, checking whether $\mathcal{K}_b \models \varphi$ can be done in time $\mathcal{O}(2^{2|\hat{U}|^a}|\varphi||\hat{U}|^{|\varphi|})$, where $a = \sum_{i=1,\ldots,n} a_i$, with a_i the arity of $P_i \in \mathcal{D}$, and $\hat{U}$ defined as in Def. 11.*

Proof (sketch). $\hat{\mathcal{K}}_{b,\varphi} \models \varphi_p$ can be checked in time $\mathcal{O}((s+t)\cdot|\varphi_p|)$, where s and t are, respectively, the number of states and transitions of $\hat{\mathcal{K}}_{b,\varphi}$ [8]. We have $s \leq |\mathcal{I}_{\mathcal{D}}(\hat{U})| \leq 2^{|\hat{U}|^a}$, $t \leq s^2$, and $s + t \leq 2s^2 \leq 2^{2|\hat{U}|^a+1}$. For φ_p, each quantifier elimination makes the current expansion grow by a factor $|\hat{U}|$, thus $|\varphi_p| \leq |\varphi| \cdot |\hat{U}|^{|\varphi|}$.

Theorem 4 gives a doubly exponential bound, which comes from the arity of the relations in $\mathcal{D}$. Observe that the bound is singly exponential in $|\hat{U}|$, which is typically greater than a. Moreover, if one needs to check the correctness of $\mathcal{S}$ against φ for different bounds b, then a can be considered constant, and the cost of increasing b becomes only single exponential. While certainly a high complexity, we observe that comparable bounds are obtained in [1] and [11]. In particular, the latter led to the implementation of a system performing surprisingly well in cases of practical interest. This may suggest that worst-case instances are not frequent in practice, and that a similar behavior might be observed also in implementations of our technique.

6 Conclusions and Future Work

In this paper we have considered the problem of checking a deployed artifact system against a temporal specification expressed in a FO extension of CTL. A notable feature of deployed systems is the existence of an upper bound on the number of elements they can store at each state at execution time. This allows us to reduce the problem to standard model checking by executing the system using only a finite number of abstract symbols instead of an infinite number of concrete ones.

Roughly speaking, our technique can be seen as an inspection of a fragment, containing only bounded states, of the original, concrete system. While this does not allow us to draw conclusions in the general case, it may provide some answers in particular cases. In this respect, we have shown that FO-ECTL properties satisfied by the abstract system are also satisfied by the concrete system, and that if the concrete system is bounded itself, our technique is complete.

We are interested in pursuing this work further. Firstly, we have shown that the bounded model of the abstract system is bisimilar to the bounded model of the concrete system. This suggests that all the obtained results, here presented in the context of FO-CTL, also hold for a FO extension of the μ-calculus analogous to [1]. If confirmed, this would imply that our work can be generalised to this setting.

Secondly, an interesting extension concerns the possibility of quantifying over variables across the scopes of modal operators, thus enabling us to capture temporal relationships among elements at different states. This introduces a major difficulty as it apparently requires to record elements from potentially infinitely many states. We are interested in pursuing abstraction techniques to avoid this problem.

Finally, the framework considered here may be extended to a Multi-Agent framework similarly to [2], thus accounting for the agents that execute the actions and their knowledge about the system.

Acknowledgements. The research leading to these results has received funding from the EC FP7 under grant agreements n. 257593, and from the EPRSC grant EP/I00520X.

References

1. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P.: Foundations of Relational Artifacts Verification. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 379–395. Springer, Heidelberg (2011)

2. Belardinelli, F., Lomuscio, A., Patrizi, F.: A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In: Proc. of IJCAI (to appear, 2011)
3. Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., Mecella, M.: Automatic Composition of Transition-based Semantic Web Services with Messaging. In: Proc. of VLDB (2005)
4. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
5. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded Model Checking. Advances in Computers 58, 118–149 (2003)
6. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract Regular Model Checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 372–386. Springer, Heidelberg (2004)
7. Caucal, D.: On Infinite Transition Graphs having a Decidable Monadic Theory. Theoretical Computer Science 290(1), 79–115 (2003)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (2000)
9. Cohn, D., Hull, R.: Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. IEEE Data Eng. Bull. 32(3), 3–9 (2009)
10. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic Verification of Data-centric Business Processes. In: Proc. of ICDT (2009)
11. Deutsch, A., Sui, L., Vianu, V.: Specification and Verification of Data-Driven Web Applications. J. Comput. Syst. Sci. 73(3), 442–474 (2007)
12. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic, vol. 148. Elsevier (2003)
13. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In: Proc. of DEBS (to appear, 2011)
14. Hull, R., Narendra, N.C., Nigam, A.: Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 1–18. Springer, Heidelberg (2009)
15. Walukiewicz, I.: Model Checking CTL Properties of Pushdown Systems. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000. LNCS, vol. 1974, pp. 127–138. Springer, Heidelberg (2000)

Profiling-as-a-Service: Adaptive Scalable Resource Profiling for the Cloud in the Cloud

Nima Kaviani[1], Eric Wohlstadter[1], and Rodger Lea[2]

[1] Department of Computer Science
[2] Department of Electrical and Computer Engineering,
University of British Columbia
201-2366 Main Mall, Vancouver, B.C. V6T 1Z4 Canada
{nkaviani,wohlstad}@cs.ubc.ca, rodgerl@ece.ubc.ca

Abstract. Runtime profiling of Web-based applications and services is an effective method to aid in the provisioning of required resources, for monitoring service-level objectives, and for detecting implementation defects. Unfortunately, it is difficult to obtain accurate profile data on live client workloads due to the high overhead of instrumentation. This paper describes a cloud-based profiling service for managing the tradeoffs between: (i) profiling accuracy, (ii) performance overhead, and (iii) costs incurred for cloud computing platform usage. We validate our cloud-based profiling service by applying it to an open-source e-commerce Web application.

Keywords: Cloud Computing, Resource Monitoring, Application Profiling.

1 Introduction

Dynamic runtime instrumentation of applications is an effective method for understanding application behavior, but imposes significant overhead to the overall execution of the application [2,7,8,10]. One approach to mitigating this overhead is offline profiling which allows the profiling process to be executed in a controlled environment, using collected traces from a previously running application. However, relying on offline collected traces often leads to inaccurate or incomplete datasets which may not represent the full spectrum of application execution states [9].

With the emergence of cloud computing and its direct mapping of resource usage to financial costs, the need to understand the low-level behavior of services and applications has become more critical, yet the challenges in profiling stay the same. However, cloud computing offers unique features which we believe can mitigate some of the above concerns. In particular, elastic and adaptive resource usage can be utilized to provide realtime analysis of system behavior with minimal performance degradation. This is achieved essentially by selectively and adaptively instrumenting only a specific subset of virtual machine (VM) instances of a deployed application.

This approach, which we refer to as *Profiling-as-a-Service* (PraaS), offers adaptive instrumentation strategies that can collect realistic profiling information about running applications in the cloud while respecting desired quality of service requirements (QoS) (e.g., response time, throughput, and cost of deployment). Such QoS strategies need

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 157–171, 2011.

to adhere to both *business* and *performance* requirements specified for an application undergoing instrumentation and monitoring. Essentially, a profiling service should help to manage tradeoffs between three factors:

- *Accuracy:* Accurate profiling information is important for software developers who need to make decisions using this data, under tight business schedules. Unfortunately, accuracy could come at the expense of performance or financial costs. Collecting detailed profiling information results in application performance degradation. Performance degradations often are tried to be overcome by supplying more resources (CPU, memory, etc.) which in turn imply higher execution costs for the application under instrumentation.
- *Performance:* Cloud-based services and applications must ensure high performance to meet expected service-level agreements and good user experience. In the cloud, performance can be obtained through elastic scaling of virtual machine (VM) instances. Unfortunately, a naive approach to scaling could be wasteful and require unnecessary financial cost.
- *Cost:* Public cloud providers offer flexible infrastructure for system scaling and reconfiguration but obviously "there is no free lunch". A profiling service will need to consider the financial costs of ensuring good accuracy and performance.

PraaS allows system architects to define policies describing their desired level of accuracy for collecting profiling information, expected QoS, and cost constraints. These policies are then uploaded to a PraaS cloud service, together with the original code for the target application, where the application is instrumented and deployed for resource usage monitoring. During application execution, PraaS will accommodate the application with just enough resources from the cloud to satisfy the specified constraints. We present our implementation of PraaS and evaluate it for a stateless, horizontally scalable, open-source Web application called RUBiS. However, we believe our approach is generalizable to other types of applications deployable to the cloud. The paper is organized as follows: in Section 2 we define the concept of Profiling-as-a-Service. In Section 3 the technical details of our framework are described. Section 4 shows some evaluation of our implementation of the service, Section 5 goes over the related work, and finally we conclude in Section 6.

2 Profiling as a Service (PraaS)

Profiling in the cloud is important for closely metering resource usage of software and inferring the corresponding financial implications. Applying traditional models of offline profiling for monitoring and provisioning of resources is not effective for applications migrated to the cloud. This is mainly due to architectural differences before and after deployment to the cloud and the heterogeneity of various cloud infrastructures.

To illustrate the benefits of integrating the profiling process with the cloud, consider a typical 3-tier throughput-intensive auctioning Web application. We use the

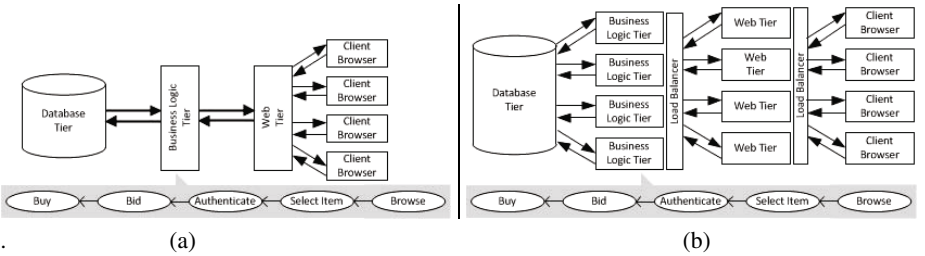
example of RUBiS [1,6], an open-source benchmark which simulates the activities of an e-commerce auction site. The original architecture of the system consists of a Web tier serving as the entry point for the application, a business logic tier containing the business logic of the application (e.g., searching, commenting, bidding, buying, authentication, and browsing of items as shown in Figure 1a-bottom), and finally a database tier to persist the transactions.

Figure 1b shows a potential architecture of the application after deployment to the cloud. As can be seen in the picture, several VM instances of the business logic tier and the Web server tier are instantiated and are placed behind load balancers. Clearly, profiling and monitoring of resources for the original application (Figure 1a) would not be helpful in understanding the behavior of the migrated application (Figure 1b). Consequently, resource usage footprints of the application in the cloud can be more effectively analyzed if profiling happens in the cloud.

When re-architectured for the cloud, all the instances in the Web server tier and the business logic tier are placed behind load balancers and hence their addition, removal, or modification stays transparent to the end-user clients of the application. These changes may only come to the attention of the end-user clients as response time delays or throughput alterations. Subsequently, as long as throughput shortfalls or response time delays are not significantly noticeable to the end-user clients, adaptive profiling strategies can be effectively blended into the overall behavior of the application.

2.1 Adaptive Resource Profiling in the Cloud

Adaptive profiling has been utilized in the past by many researchers [2,7,8,9]. Those efforts usually rely on duplicating the code blocks in an application, keeping an original version of the code along with an instrumented version. Upon occurrence of some triggering event, the instrumented and non-instrumented code are swapped [7], taking advantage of certain low-level code hot-swapping features available for some programming languages. In other efforts, instrumented and non-instrumented code are executed on different processors on one single machine [15]. In contrast, our high-level service performs adaptation at the granularity of operating system VM instances in the cloud. As such, our approach is compatible with a wider range of heterogeneous instrumentation strategies and programming platforms.



. (a) (b)

Fig. 1. A 3-tier Web application (a) before deployment to the cloud; and (b) after deployment to the cloud with potential architectural changes after cloud deployment

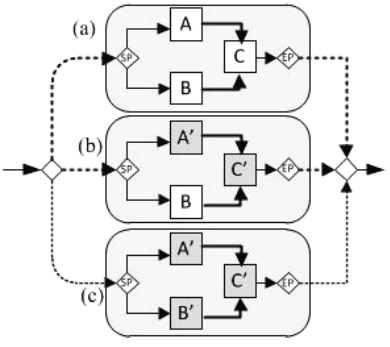


Fig. 2. Request flow through three operating system (OS) VM instances deployed from different VM images by the profiling service. The figure shows an illustrative example (i.e. the number of VM instances and application modules varies by application). This figure shows VM images with: (a) no instrumentation; (b) partial instrumentation; (c) full instrumentation. SP and EP define the start point and the end point for request flow in an application instance. A, B, and C represent different modules (e.g. classes) of the application. The shaded modules are the instrumented duplicates of the original ones for the application.

Figure 2 demonstrates the flow of a client request through an application using our profiling service. In the figure, several cloud-based VM instances of the example application are shown, processing client requests behind a load balancer. Each VM instance (a, b, or c) represents a previously imaged operating system with our custom-instrumented version of the application, deployed on a hypervisor in the cloud. For illustration we show a scenario with three different versions: (a) an instance with no instrumentation, (b) an instance with some instrumentation, and (c) an instance with full instrumentation. Our profiling service manages a repository of such VM image versions with their differing levels of profiling instrumentation. Each VM in the repository hosts a copy of the application instrumented statically right after the upload time of the original application. As described next, by utilizing declarative policies specified by developers and by monitoring certain QoS parameters, the service makes adjustments to the number of these different VM image types deployed for the application. This allows to tradeoff between performance and business requirements by adjusting the ratio of instrumented VM images to the non-instrumented ones.

2.2 Constraint-Guided Profiling Adaptation

Two major QoS requirements for Web applications deployed in the cloud are *performance*, particularly how it is perceived by the end-user clients (i.e., throughput and response time), and *cost of deployment*. Any effort to integrate profiling into the lifecycle of a deployed application to the cloud should actively respect these QoS requirements. However, supplying resources to boost the performance of an application deployed to the cloud will result in extra charges billed in monthly cycles to the clients of the cloud.

Given an upper limit for the target cost of deployment in one billing cycle in the cloud, C_t, a target performance requirement (e.g., specific throughput or response time)

for the application, P_t, and the expected performance P_{inst} after deploying a *fully-instrumented* application on a VM instance in the cloud, our adaptive swapping strategy will eventually ensure that the following inequalities hold:

$$m \times (P - P_{inst}) \geq P_t \quad (1)$$

$$C_{inst} < C_t \quad (2)$$

where m is the total number of VM instances leased from the cloud; P is the performance measure for the target application on a single VM in the absence of the instrumented code; and C_{inst} is the overall cost of running the application (in any of the non-instrumented, partially-instrumented, and fully-instrumented modes) in the cloud during a single billing cycle.

Our current strategy for implementing the above measures is based on a simple heuristic which increases the number of VM instances by a rate of $\lceil \alpha \times VM_{inst} \times (P_{inst}/P) \rceil$ where VM_{inst} is the number of instrumented virtual machines ($VM_{inst} + VM_{noinst} = m$), and α is a constant. In case the cost of instrumented deployment with increased number of VMs (C_{inst}) exceeds the previously set threshold C_t, we revert the instrumented instances back and incrementally replace the VMs running the instrumented code (VM_{inst}) with VMs running non-instrumented code (VM_{noinst}) until Inequality (2) holds again. Consequently, the current algorithm always prioritizes cost constraints to the expected performance measures. In other words, the algorithm can be thought of as a simple state machine. As long as the overall performance does not violate Inequality (1), the machine stays in an acceptable state. Once Inequality (1) is violated, the algorithm tries to bring the machine back to an acceptable state by adding more VMs or replacing instrumented VMs (VM_{inst}) with non-instrumented ones (VM_{noinst}). The state machine stabilizes under one of the following conditions: *i)* adding extra VMs brings the performance requirements back to normal without exceeding cost constraints of Inequality (2); *ii)* reverting some of the VM_{inst} machines to VM_{noinst} machines brings the performance requirements back to normal while Inequality (2) holds; or *iii)* All running machines are VM_{noinst} machines and while Inequality (1) is not satisfied, addition of another VM_{noinst} will violate Inequality (2).

In our current implementation, we translate application performance to the average application response time for requests. Hence, in Inequality (1), $P = RT$ where RT indicates the average response time when the application is in *no-instrumentation* mode; and $P_{inst} = RT_{inst}$, where RT_{inst} indicates the average response time degradation when the application is under *full instrumentation*.

For the cost of deployment, currently we consider C_{inst} equal to the total cost of application deployment (i.e., $\sum^{m} Cost(VM)$) during one billing cycle as defined by each public cloud provider (we provide details for Microsoft's Azure cloud in our evaluation). At this stage, we ignore other costs, e.g. the inbound and outbound communication costs and the costs of storing data in the cloud.

3 Technical Details

Now we turn to the specific details of our PraaS system starting with our policy specification support (Section 3.1), system architecture (3.2) and some implementation details for our specific prototype (Section 3.3).

3.1 Profiling Service Policy Specifications

To effectively expose profiling as a service to system architects, we wanted to provide a declarative policy model for controlling service parameters. Our current implementation allows for two sets of policy requirements to be specified: *Profiling Requirements* & *QoS Requirements*.

Profiling Requirements. System architects can define the level of granularity and the type of profiling that they want to be applied to the application during the execution of the application. The profiling requirements and specifications can be modified arbitrarily and even during the execution of the applications. For the level of granularity, they can choose instrumentation strategies to apply to the full application or a specific set of modules, classes, and methods in an application. They can also decide on the type of instrumentation, e.g., *CPU usage* or *Data Exchange* (described further in Section 3.3) and the scope of profiling. The scope of profiling can be defined as either *internal* or *external*.

Internal profiling only measures information internal to the elements of a module (e.g., its components, classes, and methods) while external profiling collects information from inter-module interactions in the application. Figure 3a shows a sample policy for RUBiS.

```
<profiling-spec>
  <!-- instrumentation constraints -->
  <instrumentation-map>
    <unit name="rubis.auth">
      <type>module</type>
      <profile>
        <mode>CPU</mode>
        <mode>Data</mode>
      </profile>
      <scope>internal</scope>
    </unit>

    <unit name="rubis.buy.BuyItem">
      <type>class</type>
      <profile>
        <mode>CPU</mode>
      </profile>
      <scope>internal</scope>
    </unit>

    <unit name="rubis.bid">
      <type>module</type>
      <profile>
        <mode>Data</mode>
      </profile>
      <scope>external</scope>
    </unit>
  </instrumentation-map>
```

(a)

```
  <!-- quality of service requirements
-->
  <qos-requirements>
    <cost>
      <vm-cost>2000</vm-cost>
    </cost>
    <performance>
      <resp-time>500ms</resp-time>
    </performance>
  </qos-requirements>
</profiling-spec>
```

(b)

Fig. 3. A sample instrumentation map defining (a) the modes of profiling for different modules in RUBiS and (b) QoS constraints

QoS Requirements. We also enable system architects to define their QoS constraints for instrumentation and profiling. QoS requirements are taken into consideration when ensuring Inequalities (1) and (2). As mentioned earlier, we consider a defined response time (RT_t) in milliseconds for the performance constraint of deployed Web applications and the upper limit dollar amount for leasing VMs from the cloud as the cost of deployment (C_t). Our framework supports extending these constraints with performance measures such as *throughput* or *database transactions*, and cost measures including *inbound/outbound communication costs*, and *data storage costs*. Figure 3b shows a sample QoS specification used in our RUBiS case-study.

The policy requirements of the developer are formulated into an *Instrumentation Map* document stored and loaded to a service *master node* as we describe next.

3.2 System Architecture

The architecture for our PraaS concept extends the architecture of a Web application deployed to the cloud, similar to the one in Figure 1b by adding a *master node* to each tier in the application that sits behind a load balancer. The master node encapsulates the core of the service and is loosely coupled to individual applications, communicating through an interface that specifies the exchange of profiling data and control messages. As the low-level instrumentation of code must be platform specific, this part of the service is isolated into a customized *profiler agent* colocated with each VM instance.

The profiler agent is in charge of collecting information about an application instance and reporting the collected results back to the service master node. The master node aggregates the results from all the agents and checks the validity of Inequalities (1) and (2) during the execution of the application.

As shown in Figure 4, the master node consists of the following five components: a *Profiler Specification Engine*, a *Policy Controller*, an *Instrumentation Map*, a *Reconfiguration Engine*, and a *Result Aggregator*. The profiler specification module allows the system architect to define the required profiling specification (as in Figure 3). Once the specification is loaded to the master node it is used by the master node to initiate the policy controller engine based on the `qos-requirements` part of the profiling specification, and to provide an instrumentation map. The instrumentation map is then communicated to each *Profiler Agent* to orchestrate the profiling behavior among all instances of the application.

The *Profiler Agent*, deployed together on each application node, has two components: *i)* a platform (e.g. Java, C#, etc..) specific component which takes care of instrumentation of application code, and *ii)* a *service integration module*. The SIM mediates communication of profiling data to the service master node. Through communication with the master node, SIM receives the instrumentation map specified by the system architect from the master node. The SIM then coordinates the loading of a VM image with the appropriate instrumentation.

During the profiling process, the performance on each application node gets reported to the SIM and the SIM periodically updates the master node about the status of the running application on its VM_{inst}. The master node aggregates the results from all application nodes and decides about potential reconfigurations for each node in the deployment. Upon a need for change in profiling, the SIM manages stopping and starting

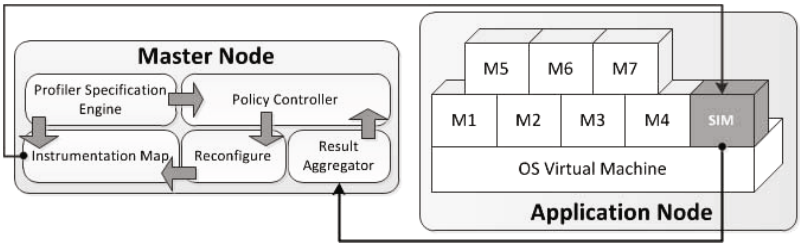


Fig. 4. Integration of Master Node and Application Node into the framework. M1 to M7 represent the classes/modules for the application and SIM is the service integration module.

new VM instances with the required instrumentation. Once the application is back to meeting all constraints, the SIM informs the master node and the master node coordinates the profiling process among all the running instances of the application again.

The master node can change the mode of profiling to one of the three already explained modes of *no instrumentation*, *partial instrumentation*, and *full instrumentation*. The adaptive switching of the profiling mode at this point is done by stopping the application on a target VM instance, replacing it with an application image in a different profiling mode, and starting the new instance. The process of mode switching is done for one instance at a time in order to minimize performance degradation caused by removing one instance of the running application. In addition to mode switching during profiling, the master node regulates the type of profiling to be performed for different instances of the application and also enables updates to the profiling process to be performed by system architects while the application is deployed in a production environment.

Our implementation for PraaS works independently of the type of instrumentation mechanism used by the profiler agents for deployed applications. Instrumentation strategies ranging from memory-leak [8] and performance bottleneck [4] detection to security related taint tracing [13] and resource usage monitoring [11,12] could be integrated into our framework. However, for our current implementation, we have particularly focused on monitoring resource utilization by different components of the application.

3.3 Prototype Profiling Support

While the service exposed by the master node is agnostic to specific platforms being profiled, a customized profiling agent is required for each programming platform (e.g. Java, C#, etc..) to be supported. Our current implementation supports Java profiling and we provide a profiler agent on top of The Java Interactive Profiler (JiP). JiP is a code profiling tool that supports performance timing, fine-grained CPU usage profiling to the level of classes and packages and requires no native code to enable profiling. JiP uses the ASM [5] library to provide manipulation, transformation, and analysis of Java classes at the level of byte code. We used the combination of JiP and ASM to collect information on CPU usage and data exchange between code blocks.

CPU Usage Profiling. CPU usage profiling is achieved simply by adding performance timers to the beginning and end of each function in the application. This is done by rewriting the Java bytecode for the function to include a `System.nanoTime()` timer.

Data Exchange Profiling. In order to make decisions on how to optimally partition software components across VM hosts in a cloud infrastructure, software developers can use profiling to determine the costs of information exchange between distributed components. Our instrumentation measures data exchange between software components by monitoring the size of remote function call arguments and return values. For local intra-VM method calls, such arguments and return values are typically passed by reference. So, in the case where developers are considering partitioning a local function into a remote function call, our framework will provide details on the size of the equivalent serialized data for each referenced argument or return value. This instrumentation strategy gives application developers a chance to measure data exchange in a monolithic application before deciding on the actual distribution.

4 Evaluation

We evaluated our current implementation against a case-study of the RUBiS benchmark.

As discussed earlier, RUBiS implements the basic functionality of an auctioning Web site following a 3-tier Web architecture with eleven components: a front-end Web server tier, nine business logic components (`User`, `UserTransactions`, `Region`, `Item`, `Category`, `Comment`, `Region`, `Bid`, and `Buy`), and a back-end database tier. Several implementations of RUBiS exist, but for our evaluation we used its Java Servlet implementation that makes use of the Hibernate middleware to provide data persistence. We deployed RUBiS together with our profiler agent modules on small instances of Microsoft's Windows Azure cloud platform. Each small Azure instance is equipped with a 1.6GHz CPU and 1.75GB of memory.

On each small Azure instance, we deployed the Web server along with all business logic components of RUBiS. For the database server, we used a 5GB SQLAzure database instance running SQLServer 2008 R2. When deploying the application on more than one instance, the Azure platform automatically places the instances behind a load balancer and distributes the load across all existing instances.

To provide a realistic client workload we used the RUBiS client simulator that comes bundled with the RUBiS benchmark [1]. The simulator was designed to operate in either a browsing mode or in a buy mode. In the browsing mode only browsing requests for items, users, comments, bids, etc. are launched. In the buy mode in addition to the browsing requests, requests to authenticate, bid on an item, or purchase of an item are also made. In our experiments, we used the clients in the browsing mode unless otherwise mentioned. Clients were launched from two machines, each equipped with a dual core 2.1GHz CPU and 4GB of memory.

4.1 Measuring Profiling Overhead

We modified the RUBiS client so that each client would generate requests at a pace of one every 125 milliseconds. To set a base line, in Table 1, we show the throughput and response time when only one single client launches requests to a single instance of RUBiS deployed on Windows Azure. We collected the data for both the profiling and the non-profiling modes. For the profiling mode, the entire set of components on the Azure instance were profiled to collect CPU usage information (cf. Section 3.3) and in the non-profiling mode, no profiling data was collected.

Table 1. Baseline throughput and response time when one client launches requests to a single instance of RUBiS deployed on one small Azure instance

	Throughput (req/sec)	Response Time (millisec)
Profiling Mode	2	432
Non-profiling Mode	6	74

From Table 1, we see the overhead of instrumentation in an isolated case. To mitigate this overhead we need to amortize it over our system. So next, in order to measure the effects of delegating profiling to a subset of all instances running an application, we made two deployments of RUBiS, one on 4 and another on 6 small Azure instances. We measured the change in response time and throughput when the number of instances running the profiling process goes from no instance (i.e., a profiling ratio of 0) to all instances (i.e., a profiling ratio of 1). A moderate load was generated on all instances in both deployments by launching 100 clients to perform 1000 transactions during a period of 5 minutes.

As Figure 5 shows, although throughput does decrease and response-time does increase as we increase the VM profiling ratio, our approach does mitigate the performance overhead of profiling. In particular, by keeping the ratio of instances that are profiled to less than 0.5 we are able to maintain throughput and response-time close to the non profile case, i.e. 200 requests per second. Only after we increase the ratio of profiled to non-profiled past 0.5 does performance significantly decline. In essence, this indicates that the Azure load balancer does a good job of intelligently redirecting requests to less busy nodes of the deployment.

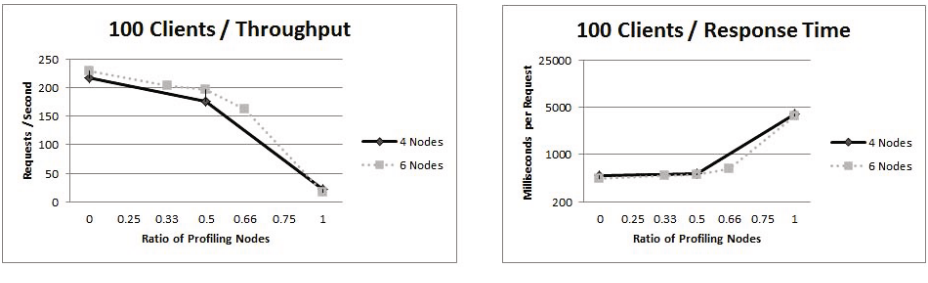


Fig. 5. Throughput and Response Time when 4 and 6 clients with different modes of profiling are placed under moderate load of client requests

4.2 Measuring Profiling Accuracy

To measure the accuracy of the PraaS system, profiling data collected at different profiling ratios were compared against the perfect profiling data (i.e., where all nodes were doing profiling), and we computed the corresponding accuracy metrics. We used the *Overlap Percentage* metric suggested by Arnold and Ryder [2] to measure the accuracy of collected profiling information. As described in [2], the overlap of two profiles represents the percentage of profiled information weighted by execution frequency that exists in both profiles. Obviously, the overlap percentage metric is a function of the diversity of requests for which profiling data is collected. In order to assess how the diversity of requests affects the overlap percentage, we collected the overlap percentage data in two modes: *i) Single Request Mode*, where RUBiS clients only launched *BrowseCategories* requests, and *ii) Multi-Request Mode*, where RUBiS clients launched all sorts of browsing requests, from browsing item categories, to browsing and searching items, browsing user information and their bid and buy histories. Table 2 shows the result of measuring overlap percentage for each of these deployments.

Table 2. The Overlap Percentage measure for accuracy of profiling information subject to the diversity of requests. Table 2 provides the overlap percentage measures for all profiling ratios of Section 4.2 for which we have throughput and response time collected.

	Single Request			Multi-Request		
Profiling Ratio	Num Samples	Num Unique Methods	Overlap %	Num Samples	Num Unique Methods	Overlap %
2 of 4 (0.5)	6.07×10^7	6338	99.65	2.74×10^7	7094	97.62
2 of 6 (0.33)	3.00×10^7	6325	99.03	3.07×10^7	6992	91.53
3 of 6 (0.5)	5.78×10^7	6329	99.10	4.35×10^7	7014	92.21
4 of 6 (0.66)	9.19×10^7	6339	99.34	5.32×10^7	7092	93.13

As expected, increasing the number of instances increases the number of samples taken during profiling. However, as we discussed earlier, increased diversity in types of requests results in a lower overlap percentage between partial profiling and full profiling. Since RUBiS is only a small representative of potential enterprise Web deployments, we expect deployments of larger applications to result in lower overlap percentages when doing partial profiling. Nonetheless, an increase in the number of nodes clearly brings the collected profiling results closer to a full profiling deployment.

4.3 Stress Testing of the Deployment and Financial Implications

In order to stress test the application, we ran three deployments of RUBiS using 4, 6, and 8 small instances on Windows Azure. Each deployment was tested with batches of 1600 and 3200 clients launching 1000 requests to it during a period of 5 minutes. Figure 6 shows the throughput and response time when 1600 and 3200 clients send requests to the deployed RUBiS application. We summarize the implications of these results next.

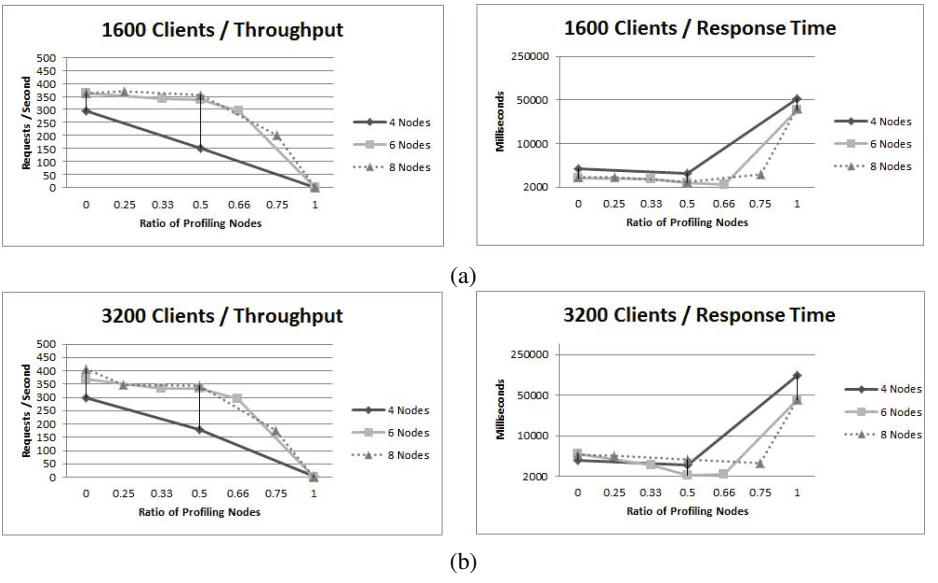


Fig. 6. Response time and throughput for (a) 1600 clients, and (b) 3200 clients; sending requests to RUBiS deployed on 4, 6, and 8 small Azure instances

In order to assess financial implications, in Table 3 we calculate the hourly and monthly costs of deployment for each of the three deployments above.

Table 3. Deployment costs for RUBiS on various number of small Azure instances

	Deployment Costs (USD)		
Num Instances	Hourly	Monthly	Yearly
4	$0.48	$345.6	$4147.2
6	$0.72	$518.4	$6220.8
8	$0.96	$691.2	$8294.4

4.4 Evaluation Summary

To summarize these findings we see that our approach of "Profiling-as-a-Service" does indeed provide a way to more accurately profile an application while respecting performance and cost constraints. In particular, it should be noted that the throughput could be kept above *300 req/sec* with 8 nodes of 0.5-profiling-ratio and marginal throughput loss (i.e. within 1% of a baseline with no profiled instances), or 6 nodes of at most 0.66-profiling-ratio (i.e. within 20% of a baseline). With a 0.75-profiling-ratio for 8 nodes, we achieved a lower throughput compared to the 0.66-profiling-ratio with 6 nodes, however the response time measured with 8 nodes was significantly smaller than the response time measured with 6 nodes. Conversely, from a financial perspective running the deployment on 6 nodes costs almost $173/month less compared to running 8 nodes. Again this demonstrates the flexibility our approach offers, allowing developers

to trade-off throughput against response time, and then factor in cost. During our experiments with 3200 clients sending requests to our RUBiS deployment, our CPU resources reached their limits and hence, even though we expected to see a higher throughput, the throughput stayed the same as for our experiment with 1600 RUBiS clients. It is worth mentioning that in our current implementation, transitions from 4 nodes to 6 nodes to 8 nodes were done manually. It is part of our future work to make this transition dynamic and based on the requirements of the target application.

5 Related Work

Profiling of service-oriented applications forms the basis of much existing work in both on-line monitoring of SLAs [3] and also the autonomic management of services. While the scope of previous work is too large to cover here in depth, we can say that compared to previous work, this paper focuses on the low-level support of profiling in the cloud environment. Previous work, on the other hand, has focused on more specific strategies for utilizing runtime profiles i.e. how to analyze and react to such profiles. Thus previous work did not cover the cloud-based adaptive instrumentation provided by our profiling service. This research simply supports an efficient profiling mechanism at the systems level. We did not address any specific policies for utilizing profile data, as we sought to provide profiling as a generic reusable service.

One of the first approaches to directly address the performance problem of on-line profiling was presented by Arnold and Ryder [2]. They use a compiler-based approach which duplicates the code for each method into instrumented and non-instrumented versions. Additionally, the compiler inserts certain "switches" in the code to allow execution to be dynamically re-directed along either instrumented or non-instrumented paths. In similar efforts Dmitriev [7] and BEA Systems' JRockit [14] use modified compilers which enable dynamic code hot-swapping to reduce profiling overhead. The approach in this paper also uses code duplication to manage profile overhead. However, our research work is different in that duplication occurs at the level of entire VM-instances. This allows a more general technique, independent of the details for specific compilers. We take advantage of the transparency afforded by cloud platforms to shield end-users from the details of swapping VM-instances dynamically. In this sense, our work is similar to the work done by Wallace and Hazlewood for SuperPin [15]. In SuperPin the authors slice the non-overlapping pieces of code into separate execution threads and run them in parallel and on multiple processor cores, gaining significant performance improvements through the added parallelism. The main difference as mentioned earlier is that we benefit from parallelism at the level of Operating System VM instances by spreading the instrumented code for the application across multiple machines in the cloud rather than using different cores on a single machine. Benefiting from the elasticity of cloud and by increasing the number of machines used for profiling, we can overcome the limitations to the degree of parallelism caused by the limited number of cores when running the instrumented software on one single machine.

Adaptive bursty tracing (ABT) [8,9] is a particular technique built for the collection of traces in profiled applications. Since trace logs can grow to enormous sizes, most profiling approaches use sampling to limit log sizes. The problem with sampling is that

it may capture very limited information about infrequently executed code. However, as authors claim, often the worst bugs and performance bottlenecks hide themselves in such code. ABT ensures that detailed traces are generated for infrequently running code, by providing a sampling rate inversely proportional to code execution frequency. In the future we will explore applying ABT to our the context of our distributed service.

AjaxScope [10] implements a JavaScript instrumentation proxy to provide monitoring and profiling of code that executes in an end-user's Web browser. This allows on-line profiling in a distributed context, where code is deployed on a server, yet later executed on the client. Traces of client behavior are periodically uploaded to the server infrastructure for analysis. Similar to our research, AjaxScope targets a distributed computing context. However, where AjaxScope focuses on client behavior, this research is focused on the server-side. This distinction changes the kind of techniques which are applicable for providing transparency. In AjaxScope, transparency is provided to clients through an instrumentation proxy whereas our research leverages the flexibility of OS VMs used in a cloud computing context.

6 Conclusion

This paper described the design and implementation for a cloud-based profiling service. This service was motivated by the need to manage tradeoffs between three important factors in the deployment of cloud services and applications: performance, cost, and accuracy of monitored profile data. We showed the validity of the approach in the context of an existing Web application deployed to the cloud. The results showed that while the reduction of profiled instances through adaptation did reduce the accuracy of profiling, it also improved performance and reduced cost. More importantly, accuracy degraded at a much slower rate than performance and cost improved.

Acknowledgements. We would like to thank Microsoft Windows Azure team, and particularly Ori Amiga, for providing us with access to resources on Windows Azure.

References

1. Amza, C., Chanda, A., Cox, A., Elnikety, S., Gil, R., Rajamani, K., Zwaenepoel, W., Cecchet, E., Marguerite, J.: Specification and implementation of dynamic Web site benchmarks. In: IEEE International Workshop on Workload Characterization, pp. 3–13 (November 2002)
2. Arnold, M., Ryder, B.G.: A framework for reducing the cost of instrumented code. In: Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation, PLDI 2001, pp. 168–179. ACM, New York (2001)
3. Baresi, L., Guinea, S., Pasquale, L.: Integrated and Composable Supervision of BPEL Processes. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 614–619. Springer, Heidelberg (2008)
4. Brear, D., Weise, T., Wiffen, T., Yeung, K., Bennett, S., Kelly, P.: Search strategies for java bottleneck location by dynamic instrumentation. IEE Proceedings - Software 150(4), 235–241 (2003)
5. Bruneton, E., Lenglet, R., Coupaye, T.: ASM: A code manipulation tool to implement adaptable systems. In: Adaptable and Extensible Component Systems, Grenoble, France (November 2002)

6. Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., Zwaenepoel, W.: Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 242–261. Springer, Heidelberg (2003)
7. Dmitriev, M.: Profiling Java applications using code hotswapping and dynamic call graph revelation. ACM Sigsoft Softwre Engineering Notes 29(1), 139–150 (2004)
8. Hauswirth, M., Chilimbi, T.M.: Low-overhead memory leak detection using adaptive statistical profiling. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-XI, pp. 156–164. ACM, New York (2004)
9. Hirzel, M., Chilimbi, T.: Bursty tracing: A framework for low-overhead temporal profiling. In: The 4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO4), pp. 117–126 (2001)
10. Kiciman, E., Livshits, B.: AjaxScope: a platform for remotely monitoring the client-side behavior of web 2.0 applications. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP 2007, pp. 17–30. ACM, New York (2007)
11. Luk, C.K., Cohn, R.S., Muth, R., Patil, H., Klauser, A., Lowney, P.G., Wallace, S., Reddi, V.J., Hazelwood, K.M.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of Programming Language Design and Implementation Conference, pp. 190–200. ACM (2005)
12. Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K., Newhall, T.: The Paradyn Parallel Performance Measurement Tool. IEEE Computer 28(11), 37–46 (1995)
13. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Network and Distributed System Security Symposium, NDSS (2005)
14. Systems, I.B.: Jrockit (August 2008), `http://www.bea.com/jrockit/`
15. Wallace, S., Hazelwood, K.: Superpin: Parallelizing dynamic instrumentation for real-time performance. In: Proceedings of the International Symposium on Code Generation and Optimization, CGO 2007, pp. 209–220. IEEE Computer Society Press, Washington, DC, USA (2007)

VM Placement in non-Homogeneous IaaS-Clouds

Konstantinos Tsakalozos, Mema Roussopoulos, and Alex Delis

University of Athens, Athens, 15748, Greece,
{k.tsakalozos,mema,ad}@di.uoa.gr

Abstract. Infrastructure-as-a-Service (IaaS) cloud providers often combine different hardware components in an attempt to form a single infrastructure. This single infrastructure hides any underlying heterogeneity and complexity of the physical layer. Given a non-homogeneous hardware infrastructure, assigning *VM*s to physical machines (*PM*s) becomes a particularly challenging task. *VM* placement decisions have to take into account the operational conditions of the cloud (e.g., current *PM* load) and load balancing prospects through *VM* migrations. In this work, we propose a service realizing a two-phase *VM*-to-*PM* placement scheme. In the first phase, we identify a promising group of *PM*s, termed *cohort*, among the many choices that might be available; such a cohort hosts the virtual infrastructure of the user request. In the second phase, we determine the final *VM*-to-*PM* mapping considering all low-level constraints arising from the particular user requests and special characteristics of the selected cohort. Our evaluation shows that in large non-homogeneous physical infrastructures, we significantly reduce the *VM* placement plan production time and improve plan quality.

1 Introduction

Infrastructure-as-a-Service (IaaS) cloud providers often face the following challenge: they must offer uniform access (resource provision) over a non-uniform hardware infrastructure. Non-homogeneous infrastructures may be the product of hardware upgrades, where old resources are left operational alongside new ones, or federated environments, where several parties are willing to share hardware resources with diverse characteristics.

Resource management of non-homogeneous hardware resources has been extensively studied [7]. Typically, a resource management system receives, queues, and finally matches user job requirements with the characteristics of the offered hardware. For instance, scheduling jobs in the Grid requires choosing an appropriate Grid site that complies with the user requirements. The advent of the clouds has introduced very strict abstractions over the physical resources. IaaS clouds restrict users from specifying the exact physical resources to be consumed when instantiating virtual machines (*VM*s). Cloud consumers remain agnostic of the underlying physical infrastructure. Only high-level resource requirements such as CPU and RAM are stated in user requests. In return, the cloud offers new options for load balancing. Live *VM* migration allows for relocation of jobs to offloaded hardware inside the cloud in a manner transparent to the user. Thus,

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 172–187, 2011.

the *VM*-to-physical machines placement policies must be revisited in the context of the cloud to take into account both the new enhancements and the additional constraints that cloud abstractions offer.

In this paper, we focus on the problem of instantiating entire virtual infrastructures in large non-homogeneous IaaS clouds. We introduce a service implementing a two-phase mechanism. In the first phase, we synthesize infrastructures out of existing promising physical machines (*PM*s). These dynamically-formed physical infrastructures, termed *cohorts*, host the user-requested *VM*s. In the second phase, we determine the final *VM*-to-*PM* mapping considering all low-level constraints arising from the particular user requests and special characteristics of the most promising selected *cohorts*. Compared to other constraint-based *VM* scheduling systems [8,9,20], the novelty of our approach mainly lies in the first phase. During this phase, besides resource availability, we also take into account properties such as migration capabilities, network bandwidth connectivity, and user-provided deployment hints. This helps prune out many possible *cohorts* within the cloud, and thus reduces the time required to produce a deployment plan in the second phase. We express both the selection of hosting nodes and the production of *VM*-to-*PM* mappings as constraint satisfaction problems and we use cloud-resources to solve these problems. Our evaluation shows that this approach *1)* scales effectively for hundreds of *PM*s, *2)* reduces plan production time by up to a factor of 9, and *3)* improves plan quality by up to a factor of 4, when compared to a single-phase *VM* placement approach.

2 Overview of Our Approach

We assign user-requested *VM*s to cloud-provided *PM*s through a service implementing a two-phase optimization process. During the first phase we select a subset of *PM*s with properties that best serve the *VM* placement. We term these *dynamically formed subsets* of *PM*s *cohorts*. Cohorts may entail *PM*s from a single rack and/or machines spread across the network. In the second phase, we solve a constraint satisfaction problem that yields a near optimal *VM*-to-*PM* mapping. Constraints emanate from user-provided deployment hints and internal cloud specifications such as hardware resource characteristics and administration preferences. The goal in selecting a subset of all available *PM*s, during the first optimization phase, is to reduce the number of constraints and the search space during the second phase.

To serve a user request for a virtual infrastructure, a single cohort has to be selected to host all *VM*s involved. The main idea in cohort selection is that we need to assist future load balancing requests in the context of an IaaS cloud. Since load balancing is better performed among *PM*s supporting live migration, we favor cohort formation among such nodes. In case, the user-requested resource requirements exceed the *VM* hosting capacity of all *PM* pools supporting live migration, we must merge neighboring pools to form larger ones. We synthesize cohorts based on a *4-level hierarchy*, depicted as a triangle in Figure 1. These four levels are defined as follows:

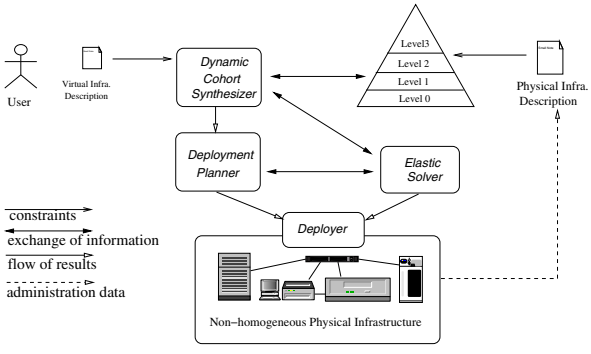


Fig. 1. High level view of our approach

- At **level 0**, all groups of *PM*s that support live migration make up corresponding cohorts. Load balancing through *VM* migration for these cohorts is transparent. In addition, each *PM* that does not feature live migration makes up a cohort on its own.
- At **level 1**, we may form cohorts from diverse groups of level 0. Cohorts of level 1 may involve *PM*s supporting both live migration as well as migration through suspend/resume.
- **Level 2** features cohorts among which migration is infeasible due to hardware incompatibilities, insufficient network bandwidth, etc.
- **Level 3** is a single cohort consisting of the entire non-homogeneous physical infrastructure and so it signifies the maximum amount of resources available.

When serving a user request, we try to satisfy all virtual infrastructure requirements through a single *cohort*. The search for such a cohort starts from level 0 and may reach up to level 3. The higher the level of the selected cohort is, the more computational resources are needed to solve the *VM* assignment problem in the second phase. As more *PM*s are available in cohorts generated at higher levels, the search space of the *VM*-to-*PM* mapping increases. Cohort selection is formulated as a constraint satisfaction problem discussed in detail in Section 4.

In the second phase, where the final *VM*-to-*PM* assignment is produced, all fine-grained constraints of the selected cohort are taken into account in a constraint satisfaction problem. These fine-grained constraints refer to *a)* all specifics regarding the hosting capacity including features and resource availability in *PM*s, *b)* user-provided deployment hints and *c)* cloud administrative goals.

Figure 1 presents a high-level view of our approach. The user submits a request for a virtual infrastructure to the *Dynamic Cohort Synthesizer*. The user request includes both *VM* specifications and deployment hints. The *PM*s of the selected cohort along with the user request are forwarded to the *Deployment Planner* that produces the final *VM*-to-*PM* mapping. We term this mapping *deployment plan*. For all user requests encountered all respective deployment plans are delivered to the *Deployer*, that in turn interacts with the cloud's facilities and coordinates each *VM* instantiation.

Both the *Dynamic Cohort Synthesizer* and the *Deployment Planner* have to solve constraint satisfaction problems. They do so using resources of the cloud itself. The *Elastic Solver*, of Figure 1, is a service providing access to a set of *VM*s in the cloud that form a distributed constraint satisfaction solver.

3 User Provided Hints and Constraints

Shown in Figure 1, the deployment of a virtual infrastructure starts with the user submitting a request for *VM*s to our service. The user request is an infrastructure description (XML document [19]) with the following sections:

1. Description of all *VM*s and resource requirements.
2. One or more possible infrastructure deployments, each one accompanied with its own set of deployment hints. The deployment hints are translated into user-provided constraints that drive the cloud's *VM*-to-*PM* assignment. Table 1 shows some commonly used hints.
3. The conditions under which a transition from one infrastructure deployment to another is required. A transition may enable and/or disable deployment hints associated with the respective infrastructure deployments. In turn, this may call for *VM* migrations.

Apart from the constraints derived from hints in the infrastructure description, our approach also leverages constraints describing the internal physical cloud infrastructure. Such constraints refer to both the availability of resources and high-level resource management goals that the cloud administration may require (e.g., *PowerSave* hint of Table 1).

Within the *Dynamic Cohort Synthesizer*, hints are realized as cohort evaluation cost functions. The same hints are realized by the *Deployment Planner* as specific deployment plan evaluation functions driving the *VM*-to-*PM* assignment. In other words, deployment hints are interpreted in different ways depending on the phase of our approach. Some hints are even ignored during the cohort selection phase. *ParVMs* is a typical example of how the same hint is treated in different ways. During cohort selection, we penalize cohorts that provide fewer *PM*s than the number of *VM*s referenced in the *ParVMs* deployment hint, whereas during the final *VM*-to-*PM* assignment we penalize plans that place the *VM*s referenced in the same *ParVMs* hint on the same *PM*. To reduce the number of constraints considered, the *Dynamic Cohort Synthesizer* ignores certain hints such as the *MinTraf*.

Table 1. Commonly used deployment hints

FavorVM	Try to reserve a single *PM* for a specific *VM*.
MinTraf	Minimize traffic by co-deploying a set of *VM*s on the same *PM*.
ParVMs	Spread *VM*s across separate *PM*s.
PowerSave	Reduce the number of *PM*s used for *VM* deployment.
EmptyNode	Offload a specific *PM*.

After a deployment plan is applied –through respective *VM* placement operations – the deployment hints used in the production of the *VM*-to-*PM* mapping are not discarded. Some deployment hints have "side-effects" on future deployment plans. A deployment hint is marked to have "side-effects" if it has to be considered during the deployment of virtual infrastructures of future user requests. Deployment hints with "side effects" are known to the cloud administration and are marked as such upon their implementation. A typical example of such a hint is the *FavorVM*, which calls for a *VM* to be placed on an offloaded *PM*. This *PM* should be kept offloaded as long as the *VM* referenced in the deployment hint is online. Consequently, the deployment of future virtual infrastructures should also respect any *FavorVM* hints already in place.

4 Synthesis of Dynamic Infrastructures

Selecting a single subset of all *PM*s (cohort) requires iterating over the levels discussed in Section 2. The *Dynamic Cohort Synthesizer* ranks the cohorts of the same level based on metrics provided by the cloud administration. These metrics may reflect the cohort's load, its reliability (e.g. redundant hardware), or even higher level properties such as its prospect of load exchange with other cohorts.

Algorithm 1 gradually explores all cohort levels in search of a promising "neighborhood". The input of the algorithm consists of a) the user *request*, b) the number of candidate cohorts (k) which should be used as the starting seed for the dynamic formation of the next level cohorts, c) the *load_threshold* (i.e., average CPU utilization) over which a cohort is considered to be overloaded and d) a resource availability factor *(overcommit)* indicating how many times the resources of a cohort should surpass the resources requested. Both input parameters *overcommit* and k allow cloud administrators to tune the quality of the produced deployment plans. The *overcommit* parameter ensures that the *Deployment Planner* will have enough space to search for a *VM*-to-*PM* mapping during the second optimization phase of our approach. The k parameter allows cloud administrators to reduce the amount of lower level cohorts used as a starting point in cohort synthesis. Since each cohort synthesis attempt results in a simulated annealing execution, high k values reduce the danger of getting trapped into a local optimum. This is because each execution of the simulated annealing starts with a different cohort as seed.

Starting from *level 0*, we first rank cohorts given that we need to satisfy the provided user request (*CohortRanking* function call of line 2). We also filter out cohorts that do not satisfy the overload threshold and the resource availability factor (*CohortFiltering* call of line 3). If all cohorts are filtered out, then we must search higher level cohorts using the while loop of lines 4 to 12. In line 6, we use the *CohortRanking* function to grade all cohorts of the level indicated by variable *level*. The top-k highest scoring cohorts of the current level are used as a starting point in exploring the next level up. Merging lower level cohorts is performed in the *MergeCohorts* call of line 8. Below, we outline the functionality of the following routines: *CohortRanking*, *CohortFiltering* and *MergeCohorts*.

Algorithm 1. *Dynamic Cohort Synthesizer*

Input: *request*: user request for a virtual infrastructure
k: The top-k cohorts will be returned
load_threshold: Threshold over which the cohort is considered overloaded
overcommit: How many times the cohort's resources must surpass the requested resources
Output: Set of cohorts we will consider for deployment

```
1: level := 0 ; ranked_cohorts := ∅
2: ranked_cohorts := CohortRanking(level,request);
3: ranked_cohorts := CohortFiltering(ranked_cohorts, load_threshold, overcommit,
       request);
4: while ranked_cohorts = ∅ and level < 4 do
5:    ranked_cohorts := ∅
6:    graded_cohorts := CohortRanking(level,request);
7:    for i := 0 ; i < k; i++ do
8:       good_cohort := MergeCohorts(level, graded_cohorts[i], request,
             load_threshold, overcommit)
9:       ranked_cohorts := ranked_cohorts ∪ {good_cohort}
10:   end for
11:   level := level + 1
12: end while
13: return ranked_cohorts
```

Cohort Ranking: Each cohort maintains the following key properties: *a)* the number of *PM*s it contains, *b)* resource availability indicators including CPU average load, total/unused RAM, hard disk capacity, redundancy and high availability features. For simplicity, we elaborate the first two indicators in the discussion that follows, *c)* average bandwidth of network connections among *PM*s within the cohort and *d)* a set of cohort-specific characteristics (e.g., CPU architecture).

Static characteristics of cohorts at level 0 are provided by the cloud administrator (with the *"Physical Infrastructure Description"* of Figure 1). In this regard, we expect the administrator to specify the properties of all *PM*s as well as the cohorts supporting live migration. Recall that each live migration group of *PM*s is a level 0 cohort and each *PM* that does not support live migration forms a cohort by itself. Using the *PM* properties we compute a score for every available cohort at level 0 as follows: initially we evaluate the resource availability within the cohort:

$$ResEval_0(Cohort) = \sum_{i \in R} w_i * (Provided_i(Cohort) - Required_i) \quad (1)$$

where R is the set of resources including average CPU, RAM, and network bandwidth utilization rates. $Provided_i$ and $Required_i$ represent the provision and requirement of resource i respectively. The weights w_i are set by the administrator to reflect the importance of each resource and to normalize the intermediate results. These administrator-defined weights allow our approach to be tuned to match specific requirements and administrative preferences. Next, we evaluate requirements resulting from both user provided hints and administrator's imperatives:

$$ConstrEval(Cohort) = \sum_{j \in Constr.} w_j * Constraint_j(Cohort) \quad (2)$$

where $Constraint_j$ are cost functions expressing user hints and administration preferences as we describe in Section 3. Again, w_j indicates constraint importance. Each constraint function takes as input a cohort and returns the degree of the human-provided preference/hint satisfaction ($Constraint : Cohort \mapsto [0, 1]$). We designate the sum of $ResEval$ and $ConstrEval$ to be the cohort's overall score:

$$Score(Cohort) = ResEval(Cohort) + ConstrEval(Cohort) \tag{3}$$

The $ResEval(Cohort)$ at levels above 0 are computed recursively as follows:

$$ResEval_n(Cohort) = \frac{\sum_{s \in S} ResEval_{n-1}(s)}{|S|} \tag{4}$$

where n represents the level and S is the set of all lower level cohorts within the $Cohort$.

Cohort Filtering: Algorithm 1 filters out cohorts that are not worthy to be considered in the final *VM* placement. This action takes place in line 3 and within the function *MergeCohorts* of line 8. Filtering uses the *overcommit*, *load_threshold* limits as well as information regarding the specific resources requested by the user. We exclude cohorts that do not comply with the rules of Table 2. *Rule 1*, avoids stressing overloaded cohorts. *Rule 2* ensures that the candidate cohorts have sufficient resources so that in the second phase (final *VM*-to-*PM* mapping) there will be enough options to choose from and produce a high-scoring plan. Finally, *Rule 3* functions in levels 0 and 1 and filters out migration-incompatible combinations of cohorts.

Cohort Merging: Through the merging of cohorts of a certain level we synthesize more comprehensive cohorts at the next level up. This operation is required when the hosting capacity of each and every cohort at the current level does not suffice to address all user resource demands. The cohort to be formed must have both the *VM* hosting capacity and the characteristics to match the user request. Thus, the goal of the merging process is to produce a number of high-scoring cohorts and then choose the "best". The formula used to compute the cohort ranking is also used here to designate this best selection. We have formulated the above selection as an optimization problem whose constraints are the user needs and administration preferences. As the number of these constraints and their combinations while merging cohorts may increase exponentially to the size of the physical infrastructure, we resort to finding a near-optimal solution.

Table 2. Cohort Filtering Rules

Rule 1:	A cohort's resource availability is below a certain "threshold" (set as input in Alg. 1)
Rule 2:	A cohort's resource availability is less than "overcommit" times the requested quantity
Rule 3: levels 0 & 1	Mismatch with user-provided specifications, such as differences in CPU architecture

Algorithm 2. Simulated-Annealing

Input: *same_iterations*: Maximum number of iterations yielding no improvement
T: Temperature
GetNeighborOf(): Space exploration function
Score(): Score function
seed: Starting seed of space exploration
Output: A near-optimal solution

```
1: same := 0
2: best_solution := current_solution := GetInitialSolution(seed)
3: while same <same_iterations do
4:    new_solution := GetNeighborOf(current_solution)
5:    D = Score(new_solution) - Score(current_solution)
6:    if ( T > 10^-5 AND e^(D/T) > Random()) OR (T < 10^-5 AND D > 0) then
7:       current_solution := new_solution
8:    end if
9:    if Score(new_solution) > Score(best_solution) then
10:      best_solution := new_solution ; same := 0
11:   end if
12:   same++ ; T := 0.99 * T
13: end while
14: return best_solution
```

To this end, we employ a stochastic and easily parallelizable approach –depicted in Algorithm 2– that is based on simulated annealing [11].

The primary objective of Algorithm 2 is to create from a lower-level *seed* cohort, a new formation at the current level. The solution has to be a single cohort that complies with the rules of Table 2. Towards achieving this objective, *GetNeighborOf* forms new potential "coalitions of resources" through navigation among cohorts of the current level. The function generates new cohorts by merging neighboring lower-level cohorts connected through at least one network route. More specifically, the lower level cohorts are randomly selected with only one requirement: each potential merging operation will result in a single cohort with its nodes adequately networked (i.e., preferably nodes that have direct physical links). Every cohort produced by *GetNeighborOf* is assigned a score within Algorithm 2 with the help of Eqn. 3. This computation is carried out efficiently as the resource evaluation is a single average sum and the constraint evaluation uses a reduced, in terms of size, set of constraints.

Algorithm 2 input parameters, T and *same_iterations* are used to designate the termination condition of the simulated-annealing procedure. As the algorithm visits more solutions, its temperature (T) drops. In high temperature states, we are allowed to choose a new solution even if it is worse than the one we currently have at hand. In this respect, Algorithm 2 avoids getting trapped in local optima. The maximum number of allowed consecutive iterations that yield no improvement is defined by the *same_iterations* parameter. As soon as the value of *same_iterations* is reached, we assume that a local near-optimum solution is found. As we show in Algorithm 1, *MergeCohorts* is called k times. Every time, we use as input a different cohort from the top-k cohorts of the previous level to serve as *seed* in Algorithm 2.

5 Deployment Plan Production

All properties of the selected cohort (i.e., *PM*s, resource availability, network connections and cohort-specific characteristics) along with the user-provided constraints (deployment hints) are used as input to the *Deployment Planner* that ultimately produces the actual mapping of *VM*s-to-*PM*s in a way similar to the one discussed in [19]. The *Deployment Planner* also takes into account previously encountered deployment hints referring to virtual infrastructures already deployed and operational on (some) nodes of the selected cohort. However, since these infrastructures are already deployed, the set of deployment hints can be trimmed down only to those hints marked to have "side effects" as we discuss in Section 3. We employ Algorithm 2 to produce the mapping of *VM*s to *PM*s. Again, we need to designate two aspects: a) how to select a neighbor solution commencing with a *seed*-mapping and b) how to ascertain the value of the produced candidate solution.

In general, the plans neighboring a specific deployment plan p are designated by the following formula:

$$N_p = \{P \in AllPlans \mid Prob(P(v) \neq p(v)) = d, \forall v \in V\}, \tag{5}$$

where V is the set of *VM*s under deployment and d is the value of the probability that a *VM* v is to be deployed on a *PM* other than the one currently set in plan p. High values of d result in producing almost random deployment plans that render Algorithm 2 ineffective. On the other hand, significantly reducing d may trap the search process for a neighbor(s) into local optima. A detailed discussion on this plan generation procedure can be found in [19].

The input scoring function is implemented as the weighted sum of the cost evaluation functions corresponding to the constraints relevant to the user request at hand. For a given deployment plan m the *Score* is:

$$Score(m) = \sum_{Const_i \in C_s} w_i Const_i(m) \tag{6}$$

where C_s is the set of constraints and w_i the respective user/administrator imposed weights indicating the importance of each constraint.

6 Elastic Solver Service

Simulated annealing is inherently parallelizable and can effectively harvest the distributed nature of the same cloud infrastructure we administer. Every time the simulated annealing is invoked, it can use different seeds (e.g. Algorithm 1 line 8). In this manner even if one execution gets trapped into a local optimum, a plausible solution can be ultimately found by another execution.

The *Elastic Solver* is a service providing virtual infrastructures used to solve the constraint satisfaction problems described in the previous sections.

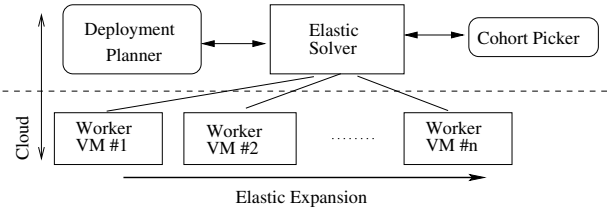


Fig. 2. High level view of the *Elastic Solver*

As Figure 2 depicts, the solver follows a master-workers architecture and interacts with both the *Dynamic Cohort Synthesizer* and the *Deployment Planner*. As the virtual infrastructures employed by the *Elastic Solver* are hosted in the cloud, this service also acts as a special cloud client. Worker *VM*s requested by the solver are configured so that as soon as they come online they register with the master component of the *Elastic Solver* and declare their availability. These workers remain idle until a request for solving a cohort-merging or a *VM* placement problem arrives. In [18], we show how we are able to dynamically adjust the exact number of worker nodes so that our "profit" in using this service is maximized. We employ an iterative process that periodically adds or removes worker nodes in an attempt to assess the *VM*'s performance and to properly adjust the number of worker nodes. Larger elastic solver infrastructures result in better deployment solutions and reduced deployment time but at the cost of higher maintenance overhead as they reserve more resources.

7 Evaluation

Our evaluation examines the performance of deployment plan production for a wide range of different infrastructures. A comparison of our constraint-based approach against other *VM* placement algorithms can be found in [19]. We have simulated two network topologies, shown in Figure 3, denoted as LAN and star. In both topologies, there are groups of *PM*s supporting live migration. The two topologies differ in the way these groups are connected amongst themselves. A LAN can be created by lining up switches, each one leading to a single live migration group (bottom left of Figure 3). In the star topology (right half of Figure 3), a central switch connects N other switches and each switch leads to N live migration groups. N is the fan-out of the star topology. In our evaluation, the network's fan-out also indicates the number of *PM*s inside any live migration group in both the star and LAN topologies.

With the exception of the network connections inside a live migration group, all other network links are assigned their bandwidth randomly out of three distinct values: 100, 1,000 and 10,000 *Kbps*. We expect connection within a live migration group to be dedicated and of high bandwidth, thus, we assign them the maximum bandwidth of 10,000 *Kbps*. Apart from the connectivity bandwidth,

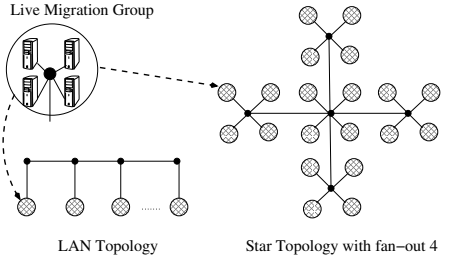


Fig. 3. Simulated network topologies

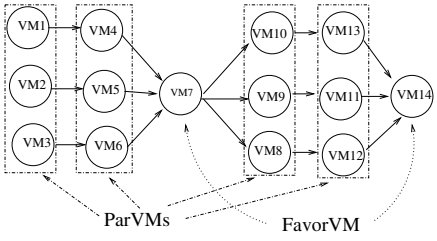


Fig. 4. LIGO inspired virtual infrastr

infrastructure non-homogeneity is also introduced through the characteristics of the *PM*s. Each live migration group is made of identical *PM*s belonging to one of three classes. Each of the three equally-sized classes has *PM*s of specific capacity in hosting *VM*s. Here, this capacity depends on the amount of available RAM. Therefore, classes feature *PM*s of 16, 8 and 4 *GB* of RAM respectively. In addition, we assume 20% of all live migration groups to be incompatible with the user request (e.g. due to different CPU architecture). In our simulation, whenever we set an average load percentage, we randomly deviate from it up to 10%. The load is realized as a reduction of the available resources of the infrastructure. Each *PM* features a single CPU from which *VM*s reserve a fraction. The CPU reserve fraction is explicitly stated in the user's virtual infrastructure request.

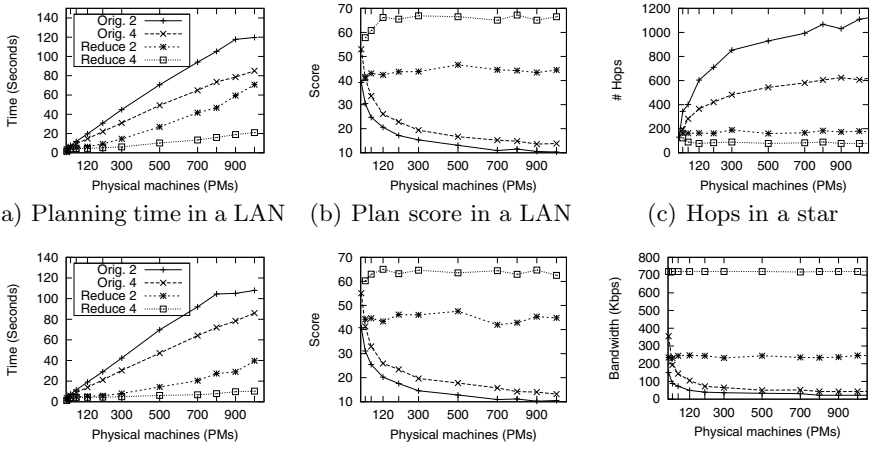
Workload Description: We have run experiments with all the workflows described in [3]. Here, we present the results of a virtual infrastructure request resembling the Laser Interferometer Gravitational Wave Observatory (LIGO) Inspiral Analysis Workflow. Figure 4 presents the *VM*s involved along with the deployment hints provided by the user. The requested virtual infrastructure is made of 14 *VM*s. Each *VM* reserves 1 *GB* of RAM and 10% of the CPU available on the hosting node. Table 3 summarizes all user hints and their weights used in this experiment. Apart from the user hints, we also employ administrative deployment hints (Table 4) so as to promote the deployment of *VM*s in neighboring *PM*s. The input parameters of Algorithm 1 are as follows: *load_threshold* is 10%, *overcommit* is set to 6, we perform 200 *same_iterations* and we set k to 100.

Table 3. User deployment hints

Hint	Involved VMs	Weight
`ParVMs` (x4)	VMs {1,2,3},{4,5,6}, {8,9,10},{11,12,13}	40.0
`FavorVM` (x2)	VM 7 & VM 14	40.0

Table 4. Administrative hints

Hint	Description	Weight
`Reduce Groups`	Reduce the number of live migration groups	40.0
`Reduce Dist`	Reduce the network hops between migration groups	40.0



(a) Planning time in a LAN (b) Plan score in a LAN (c) Hops in a star

(d) Planning time in a star (e) Plan score in a star (f) Bandwidth in a star

Fig. 5. Evaluating our approach when increasing the infrastructure's size

Scaling the Infrastructure: Figures 5(a) and 5(d) show the time required for producing a deployment plan as we increase the number of *PM*s from 30 to 1000. In configurations denoted as "Orig." we use only the *Deployment Planner* for plan production whereas in configurations denoted as "Reduce" we also employ the *Dynamic Cohort Synthesizer* to reduce the search space. For each of the two topologies, we offer two variations corresponding to different fan-out values. "Orig. 2" and "Reduce 2" correspond to configurations where the fan-out is 2 whereas for "Orig. 4" and "Reduce 4" the fan-out is 4.

To have a full understanding of the improvement, we must also examine the score of the produced plans, as computed through Equation 1. Figures 5(b) and 5(e) show the plan score for the LAN and star topology respectively. The *Deployment Planner* alone fails to produce high-scoring deployment plans as the size of the infrastructure increases. The reason for this is that the selection of *PM*s tends to disperse the virtual infrastructure across the physical infrastructure. The extra proximity cost functions that promote plans utilizing neighboring *PM*s are not enough to concentrate the user's *VM*s. The option of increasing the weight of the proximity functions proves to be ineffective in large infrastructures. High weight values for proximity administrative hints render the user provided hints insignificant playing a minor role in plan production. Instead, when we reduce the hosting infrastructure, *Deployment Planner* manages to produce high-scoring plans. The cohort selection renders the proximity cost most effective.

Network Efficiency When Scaling the Infrastructure: In non-homogeneous infrastructures, low-bandwidth network connections used by several *VM*s may quickly become a performance bottleneck. To this end, we try to concentrate *VM*s of the same virtual infrastructure in the same neighborhood of *PM*s. We use two metrics to measure network efficiency a) the "packet hop count" and

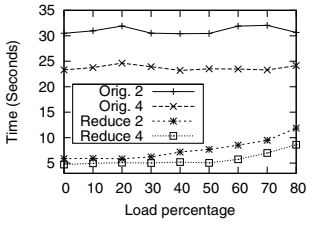


Fig. 6. Plan time under increasing load

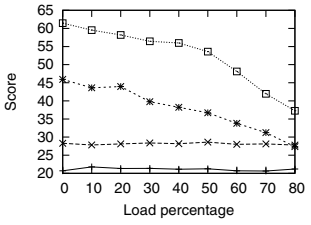


Fig. 7. Plan score under increasing load

b) the "average minimum path bandwidth". For each pair of *VM*s there is a path of minimum length over the physical network that connects the two *PM*s where *VM*s are deployed. We are interested in two properties of this min-path, a) its length and b) the connection with the minimum bandwidth along this min-path. The length of the min-path is the number of network switches (hops) a packet must pass starting from the source *VM* until it reaches the target *VM*. The "packet hop count" metric is the summation of all min-path lengths of all possible *VM* pairs. We use "packet hop count" to estimate the network latency within the virtual infrastructure. The connection with the minimum bandwidth in a min-path also determines the maximum bandwidth of the communication among the source and target *VM*s -as the min-path is often the only path. The "average minimum path bandwidth" also takes into account all min-paths of all possible *VM* pairs and produces an average of the minimum bandwidth. We use the "average minimum path bandwidth" as an indicator of the virtual network bandwidth within the virtual infrastructure.

Figures 5(c) and 5(f) show the average number of hops and the bandwidth in plans produced in a star topology when we use the *Dynamic Cohort Synthesizer* (Reduce 2 & 4) and when we do not (Orig. 2 & 4). The respective LAN results are similar. When selecting a cohort in our approach, the search space shrinks radically and remains unchanged regardless of the set of *PM*s available in the infrastructure. The confinement of *VM*s to a small sub-infrastructure with dedicated network connections such as those within live migration groups increases the average minimum bandwidth.

Infrastructure Load: Increasing the load of the cloud reduces the capacity of the physical infrastructure to host *VM*s. We fix the number of *PM*s to 100 and gradually increase their load from 0% to 80%. Figures 6 and 7 show that high load has little impact on the performance of the *Deployment Planner* operating alone, without the help of *Dynamic Cohort Synthesizer*. This is because the number of constraints in the respective constraint satisfaction problem remain the same regardless of load. Any constraints depicting the load on a particular *PM* must be taken into consideration by the *Deployment Planner* regardless of whether the *PM*'s load is low or high. In contrast, in our two-phase approach there is a performance degradation as the load increases. Yet, the worst performance we get in an unrealistic scenario, with 80% load, is still higher than the respective "Orig." configuration.

8 Related Work

The assignment of *VM*s-to-*PM*s can be reduced to the job assignment problem should *VM*s correspond to jobs and *PM*s to processing elements. The job assignment problem has been extensively studied [15,24], yet it is regularly revisited as application areas emerge. The placement policy in [16] exploits the tendency of *VM*s to have certain properties in common. In [23] a two level control management system is used for the placement of *VM*s to *PM*s using combinatorial and and multi-objective optimization to address potentially conflicting placement constraints. [5] reformulates the problem as a multi-unit combinatorial auction. In [17], placement constraints are treated as separate dimensions in a multi-dimensional *Knapsack* problem. User and administrative preferences expressed through constraints are also employed in [8,9,20,19]. Often, as the number of constraints increases more resources are needed to solve the constraint satisfaction problem. Here, we address the issue of *VM* placement scalability through the introduction of dynamically-formed cohorts.

Network connectivity is of critical importance for data centers [2] and has influenced our experimental group formation. [14] outlines an approach that builds networks of *VM* test-beds over physical infrastructures via simulated annealing. Algorithms that improve the embedding of virtual networks to physical layouts are considered in [6]. In [13], the *VM* placement is mainly addressed from the network traffic perspective.

The use of heuristics is a common approach among systems performing load balancing in data centers. *vManage* [12] describes a low overhead solution for managing load in an infrastructure hosting *VM*s. The *VM* placement policies primarily consider properties of both platform (e.g., power management) and the virtualization layer (e.g., SLA violations). Likely instability issues are also addressed. *Sandpiper* [22] detects and monitors performance bottlenecks in a cluster hosting *VM*s. Two approaches are evaluated in the decision making mechanism that produces the *VM* migration actions: the first, termed *black-box*, remains fully OS-and-applications agnostic while the second, termed *gray-box*, exploits statistics originating from both the OS and the application-layer. Compared to both *Sandpiper* and *vManage*, our approach addresses performance bottlenecks by exploiting provided preferences and not by monitoring the operation of *VM*s. Modeling the *VM* load is deemed important in the *black-box* approach used in IaaS clouds. [10,4,21] classify *VM* workloads and develop metrics to model the encountered workloads in an effort to reduce *VM* migration costs.

In our approach, we do not try to predict the performance requirements of *VM*s but we trust user-provided hints to avoid performance bottlenecks. Through *PM* grouping, we reduce the search space of the constraint satisfaction problem in the second phase (*VM* to *PM* mapping). In this way, we address both potential scalability issues [8,9,20,19], and balance load. Compared to [14,6,13], our approach is more general as it makes use of constraints expressing high-level properties. Finally, compared with other site-based approaches [1], our approach synthesizes physical infrastructures (cohorts) on-the-fly.

9 Conclusions

In this paper, we examine the problem of *VM* placement in non-homogeneous IaaS cloud environments. We propose a service realizing a two-phase approach that manages diversified resources. Compared to a heavyweight monolithic approach, our scheme can scale to several hundreds of physical nodes. In fact, the quality of the deployment plans we produce remains largely unaffected by the size of the physical infrastructure. A key concern has been the confinement of the solicited virtual infrastructure into a dynamically-adjusted set of physical nodes whose size and properties match the user requests. As a result, overall plan quality is improved since latency amongst deployed *VM*s is reduced and the average bandwidth increases dramatically. Our future work includes exploring the use of other constraint satisfaction algorithms and refining the cost and revenue functions of the *Elastic Solver* service.

Acknowledgments. We thank the reviewers for their comments. This work has been partially supported by the *EU D4Science I&II* FP7-projects.

References

1. Opennebula. (November 2010), `http://www.opennebula.org`
2. Al-Fares, M., Loukissas, A., Vahdat, A.: A Scalable, Commodity Data Center Network Architecture. In: Proc. of the ACM SIGCOMM Conference, pp. 63–74. ACM, Seattle (2008)
3. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of Scientific Workflows. In: 3rd Workshop on Workflows in Support of Large-Scale Science, Austin, TX, November 2008, pp. 1–10 (2008)
4. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In: Proc of the 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany (May 2007)
5. Breitgand, D., Epstein, A.: SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds. In: IFIP/IEEE International Symposium on Integrated Network Management, Dublin, Ireland (May 2011)
6. Chowdhury, N., Rahman, M., Boutaba, R.: Virtual Network Embedding with Coordinated Node and Link Mapping. In: Proc. of IEEE INFOCOM, Rio de Janeiro, Brazil (April 2009)
7. Cierniak, M., Zaki, M.J., Li, W.: Compile-Time Scheduling Algorithms for a Heterogeneous Network of Workstations. The Computer Journal 40(6), 356–372 (1997)
8. Hermenier, F., Lorca, X., Menaud, J., Muller, G., Lawall, J.: Entropy: a Consolidation Manager for Clusters. In: Proc. of the 2009 ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments, Washington, DC (March 2009)
9. Hyser, C., McKee, B., Gardner, R., Watson, B.J.: Autonomic Virtual Machine Placement in the Data Center. HP Laboratories HPL-2007 189 (2008)
10. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application Performance Management in Virtualized Server Environments. In: Proc of the 10th IEEE/IFIP Network Operations and Management Symposium, Vancouver, Canada (April 2006)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220, 671–680 (1983)

12. Kumar, S., Talwar, V., Kumar, V., Ranganathan, P., Schwan, K.: vManage: Loosely Coupled Platform and Virtualization Management in Data Centers. In: Proceedings of the 6th International Conference on Autonomic Computing, June 2009, pp. 127–136. ACM, Barcelona (2009)
13. Meng, X., Pappas, V., Zhang, L.: Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: Proceedings of IEEE INFOCOM, San Diego, CA, USA (March 2010)
14. Ricci, R., Alfeld, C., Lepreau, J.: A Solver for the Network Testbed Mapping Problem. SIGCOMM Computer Communications Review 33(2), 65–81 (2003)
15. Rotithor, H.: Taxonomy of dynamic task scheduling schemes in distributed computing systems. In: IEEE Proceedings Computers and Digital Techniques (January 1994)
16. Sindelar, M., Sitaraman, R.K., Shenoy, P.: Sharing-Aware Algorithms for Virtual Machine Colocation. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, California, USA (June 2011)
17. Singh, A., Korupolu, M., Mohapatra, D.: Server-Storage Virtualization: Integration and Load Balancing in Data Centers. In: Proc. of the 2008 ACM/IEEE Conference on Supercomputing SC 2008, pp. 53:1–53:12 (2008)
18. Tsakalozos, K., Kllapi, H., Sitaridi, E., Roussopoulos, M., Paparas, D., Delis, A.: Flexible Use of Cloud Resources through Profit Maximization and Price Discrimination. In: Proc. of the 27th IEEE Int. Conf. on Data Engineering (ICDE 2011), Hannover, Germany (April 2011)
19. Tsakalozos, K., Roussopoulos, M., Floros, V., Delis, A.: Nefeli: Hint-based Execution of Workloads in Clouds. In: Proc. of the 30th IEEE Int. Conf. on Distributed Computing Sytems (ICDCS 2010), Genoa, Italy (June 2010)
20. Wang, X., Lan, D., Wang, G., Fang, X., Ye, M., Chen, Y., Wang, Q.: Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center. In: Proc. of the 4th Int. Conf. on Autonomic Computing, Washington, DC, p. 29 (2007)
21. Weng, C., Li, M., Wang, Z., Lu, X.: Automatic Performance Tuning for the Virtualized Cluster System. In: Proc. of the 29th IEEE International Conference on Distributed Computing Systems, Montreal, Canada (June 2009)
22. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In: Proc of the 4th USENIX Symposium on Networked Systems Design and Implementation, Cambridge, MA (2007)
23. Xu, J., Fortes, J.A.B.: Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. In: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, Hangshou, PR of China (December 2010)
24. Yeo, C.S., Buyya, R.: A taxonomy of market-based resource management systems for utility-driven cluster computing. Softw. Pract. Exper. 36(13) (November 2006)

Place Semantics into Context: Service Community Discovery from the WSDL Corpus

Qi Yu

College of Computing and Information Science
Rochester Institute of Technology
qi.yu@rit.edu

Abstract. We propose a novel framework to automatically discover service communities that group together related services in a diverse and large scale service space. Community discovery is a key enabler to address a set of fundamental issues in service computing, which include service discovery, service composition, and quality-based service selection. The standard Web service description language, WSDL, primarily describes a service from the syntactic perspective and rarely provides rich service descriptions. This hinders the direct application of traditional document clustering approaches. In order to attack this central challenge, the proposed framework applies Non-negative Matrix Factorization (NMF) to the WSDL corpus for service community discovery. NMF has demonstrated its effectiveness in clustering high-dimensional sparse data while offering intuitive interpretability of the clustering result. NMF-based community discovery is further augmented via semantic extensions of the WSDL descriptions. The extended semantics are first computed based on the information sources outside the WSDL corpus. They are then seamlessly integrated with NMF, which makes the semantic extensions fit in the context of the original services. The experiments on real world Web services are presented to show the effectiveness of the proposed framework.

1 Introduction

Web services are increasingly being adopted to access data and applications across the Web [19]. This has been largely the result of the huge investment in Web application development and the many standardization efforts to describe, advertise, discover, and invoke Web services [3]. The emergence of cloud infrastructure also offers a powerful yet economical platform that greatly facilitates the development and deployment of a large number of Web services. Based on the most recent statistics, there are 28,593 Web services being provided by 7,728 distinct providers over the world and these numbers keep increasing in a fast rate [1]. Despite the abundance of various supporting technologies to facilitate the access to these Web services, there currently lacks a meaningful organization of the large and diverse Web service space. Most current Web services exist on the Web in a disorganized manner, which poses significant challenges for users to fully leverage the wealthy computing resources offered by these services.

[1] http://webservices.seekda.com/

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 188–203, 2011.

Discovery of service communities that group together related services is a key enabler to address a set of fundamental issues in service computing that include service discovery, service composition, and quality based service selection:

- **Service discovery:** Searching a service with a desired functionality can be performed solely within the service communities that offer relevant functionalities. This not only increases the searching accuracy but also significantly reduces the searching time because services from irrelevant communities are directly filtered out.
- **Service composition:** Grouping together relevant services into communities facilitates the discovery of potentially composable services. Service composition can be (semi-)automated in such a controlled environment to generate value-added composite services.
- **Service selection:** As competing Web services that offer "similar" functionalities will be categorized into the same service communities, service users are provided with a one stop shop to get the service with required functionality and the best desired quality.

Existing efforts in constructing service communities can be categorized into either *top-down* or *bottom-up* approaches. A top-down approach usually starts with a set of predefined template services and bootstraps the communities by grouping together the related template services. It then relies on the services to register to the corresponding service communities based on the similarity with the template services. A top-down strategy may only be applicable to a limited number of Web services (e.g., within an organization), where a centralized control on the services can be enforced. Unfortunately, when a large scale of Web services from an open environment (e.g., the Web) are considered, the top-down strategy presents key challenges. One the one hand, as Web services are expected to be *autonomous* (i.e., provided by independent service providers) and *a priori unknown*, it is infeasible to predefine the template services that match the functionalities of these services. On the other hand, it is also unreasonable to rely on the independent service providers to register their services with the predefined service communities.

Bottom-up approaches directly infer service communities from the Web service descriptions. Most existing Web services are described using the standard Web service description language, WSDL. However, WSDL primarily describes a service from the syntactic perspective and rarely provides rich service descriptions [7]. This hinders the direct application of traditional document clustering approaches. Some recent efforts have been devoted to break the limitations of WSDL for improving the accuracy of service search and community discovery. These approaches can be divided into two categories, both of which, however, suffer some major issues.

- **The first category** aims to fully exploit the information carried by the WSDL service descriptions [7,8,13,12]. For example, a key premise behind the Woogle Web service search engine is that terms that co-occur frequently tend to share the same concept [7]. Nevertheless, WSDL descriptions usually

come with very limited number of terms. Hence, *semantically similar terms (e.g., car and vehicle) will have a slim chance to co-occur in a WSDL corpus and thus be deemed as irrelevant.*

– **The second category**, on the other hand, explores external information sources, such as WordNet, Wikipedia, and search engines, to extend WSDL with rich semantics [11,2]. However, *the external semantic extensions may not fit into the context of the original services.* For example, "apple" means different things for a computer hardware service and an online grocery store service. In this regard, *the semantic extensions are useful only when they can be leveraged in the context of the original service.*

We propose a novel framework to discover service communities that group together related services from diverse and large scale Web services. We adopt the bottom-up strategy so that the communities can be automatically discovered from the WSDL corpus. In order to attack the central challenges as highlighted above, the proposed framework exploits Non-negative Matrix Factorization (NMF) as a powerful tool for service community discovery. NMF-based community discovery is further augmented via semantic extensions of the WSDL descriptions. The **key contributions** of the proposed framework are summarized as follows.

Community Discovery via NMF. Service community discovery is to group together Web services with similar functionalities. As the functionalities of Web services are captured by the operations they offer, we construct an $m \times n$ matrix X, where the i-th row represents service $\mathbf{s}_i$, the j-th column represents operation $\mathbf{o}_j$, and the entry $\mathbf{X}(i, j)$ represents the association between $\mathbf{s}_i$ and $\mathbf{o}_j$. We exploit an augmented version of NMF, called Non-negative Matrix Tri-Factorization (NMTF), which factorizes matrix $\mathbf{X}$ into three low-rank non-negative matrices: a service cluster indicator matrix, an operation cluster indicator matrix, and a service-operation association matrix. NMTF in essence simultaneously clusters both services and operations. In this way, NMTF not only leverages the WSDL service descriptions but also exploits the "duality" relationship between services and operations [5,20]. Duality signifies that service clustering is determined by the functionalities of services (i.e., the operations they offer) while operation clustering is determined by the co-occurrence of operations in functionally similar services. Simultaneously clustering services and operations enables the two clustering processes to guide each other so that the overall clustering accuracy can be improved. Furthermore, the non-negative constraint of NMTF yields a natural parts-based representation of the data as it only allows addictive combinations [10]. Thus, the clustering result from NMTF is more intuitive to interpret.

Semantic Extension Integration. NMTF goes beyond the existing service and community discovery approaches by fully exploiting the information carried by the WSDL corpus, which includes not only the service descriptions but also the duality relationship between services and operations. Unfortunately, due to the

limited descriptive capacity of WSDL, terms that share similar semantics may be regarded as irrelevant if they do not co-occur in a WSDL file. This will lead to poor community discovery performance. To attack this challenge, we compute the semantic extensions of the WSDL corpus by leveraging external information sources. We then integrate the semantic extensions into the NMTF process, where the original service descriptions are used to discover the service communities. The amalgamation of the semantic extensions and NMTF has the effect of fitting the extended semantics obtained from external sources into the context of the original services. This enables the proposed the framework to effectively leverage the semantic extensions to benefit service community discovery.

Outline: The remainder of the paper is organized as follows. We propose a framework for service community discovery in Section 2. The cornerstone of the proposed framework is the usage of Non-negative Matrix Tri-Factorization (NMTF) to simultaneously cluster services and operations. We present a strategy for computing the semantic extensions of the WSDL corpus in Section 3. We then elaborate on how to integrate the extended semantics into the community discovery framework. We evaluate the effectiveness of the proposed service community discovery framework via real-world Web services in Section 4. We give an overview of related work in Section 5 and conclude in Section 6.

2 Framework for Service Community Discovery

Service community discovery aims to group together Web services that provide similar functionalities. Since the functionality of a Web service is reflected by its operations, it is desirable to evaluate the similarity between services based on the operations they offer. We consider two types of objects in a Web service space: services $\mathcal{S} = \{\mathbf{s}_1, ..., \mathbf{s}_m\}$ and operations $\mathcal{O} = \{\mathbf{o}_1, ..., \mathbf{o}_n\}$. The association (or similarity) between a service s and an operation $\mathbf{o}$ is denoted by a scalar value $x(\mathbf{s}, \mathbf{o})$. Thus, we can use a m-by-n two dimensional matrix $\mathbf{X}$ to denote the association between each pair of service and operation if we map the row indices into $\mathcal{S}$ and the column indices into $\mathcal{O}$. Each entry $\mathbf{X}(i, j) \in \mathbf{X}$ denotes the association between service $\mathbf{s}_i$ and operation $\mathbf{o}_j$. We refer to the matrix $\mathbf{X}$ as the service-operation contingency matrix. Once matrix $\mathbf{X}$ is constructed, the similarity between services $\mathbf{s}_i$ and $\mathbf{s}_j$ can be computed as the dot-product of the i^{th} and j^{th} row vectors of $\mathbf{X}$:

$$sim(\mathbf{s}_i, \mathbf{s}_j) = \mathbf{X}(i, :) \cdot \mathbf{X}(j, :) \tag{1}$$

To complete the construction of matrix $\mathbf{X}$, we also need to compute the association between each pair of service and operation. This can be achieved by representing both services and operations as N-dimensional term vectors, where N is the number of distinct terms in the WSDL corpus. More specifically, if the k^{th} term appears in the description of service $\mathbf{s}_i$ (or the signature of operation

Table 1. Notations

Notation	Description
$\mathcal{S}, \mathcal{O}$	sets of services and operations
$\mathbf{s}_i, \mathbf{o}_j$	the i^{th} service and j^{th} operation
$W_{\mathbf{s}_i}$	the WSDL description of service $\mathbf{s}_i$
$E(W_{\mathbf{s}_i})$	the semantic extension of $W_{\mathbf{s}_i}$
$\hat{s}_p$, $\hat{o}_q$	the p^{th} service community and q^{th} operation community
$\mathbf{X}, \mathbf{S}, \mathbf{R}, \mathbf{O}$	matrices
$\mathbf{X}^T$	the transpose of matrix $\mathbf{X}$
$\mathbf{X}(i,j)$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{X}$
$\mathbf{X}(i,:)$	the i^{th} row of matrix $\mathbf{X}$
$\mathbf{X}(:,j)$	the j^{th} column of matrix $\mathbf{X}$

$\mathbf{o}_j$), the corresponding entry in the term vector will be set as the frequency of this term [2]. Otherwise, the corresponding entry is set to 0. Hence, the association between service $\mathbf{s}_i$ and operation $\mathbf{o}_j$ can be computed as the dot-product of their term vectors. Table 1 lists the notations that are used throughout this paper.

2.1 Community Discovery via NMTF

In this section, we propose to use a Non-negative Matrix Tri-Factorization (NMTF) process to discovery service communities based on the service-operation contingency matrix $\mathbf{X}$ constructed above. In particular, NMTF factorizes $\mathbf{X}$ into three low-rank matrices, i.e.,

$$\mathbf{X} \approx \mathbf{SRO}^T \tag{2}$$

where $\mathbf{S} \in \mathbb{R}^{m \times k}$ is the cluster indicator matrix for clustering services (i.e., rows of $\mathbf{X}$), $\mathbf{O} \in \mathbb{R}^{n \times l}$ is the cluster indicator matrix for clustering operations (i.e., columns of $\mathbf{X}$), $\mathbf{R} \in \mathbb{R}^{k \times l}$ is the cluster association matrix that captures the association between service clusters and operation clusters. NMTF in essence simultaneously clusters $\mathcal{S}$ into k disjoint service communities and $\mathcal{O}$ into l disjoint operation communities. In this way, it effectively exploits the *duality* between services and operations to improve the overall community discovery accuracy.

To further demonstrate how NMTF works, we use a collection of real-world WSDL files obtained from [9]. This dataset consists of over 450 services from 7 different domains. For a clear illustration, we select 5 services, where three of them are from the eduction domain and two are from the medical domain. Each service offers one operation and thus there are altogether five operations. Through some preprocessing of the WSDL files (refer to Section 5 for details), we identify 33 distinct terms. Hence, all the services and operations can be represented as 33-dimensional vectors. Then, we construct a 5×5 contingency matrix $\mathbf{X}$ where each row represents a service and each column represents an operation. Applying NMFT on $\mathbf{X}$, we obtain the following result in Equation (3). It is

[2] Other values, such as the TFIDF score [1], can also be used.

easy to tell that the first three rows of $\mathbf{X}$, which represent three education services, are grouped into the first service community $\hat{s}_1$ (because $\mathbf{S}(i,1) > \mathbf{S}(i,2)$, where $i \in \{1,2,3\}$). The last two rows, representing two medical services are grouped into the second service community $\hat{s}_2$ (because $\mathbf{S}(i,1) < \mathbf{S}(i,2)$, where $i \in \{4,5\}$). Similarly, columns 1, 2, and 3, which represent three operations from the education domain are grouped into the first operation community $\hat{o}_1$ and the fourth and fifth operations are grouped into the second operation community $\hat{o}_2$.

$$\underset{\mathbf{X}}{\begin{pmatrix} 81 & 3 & 22 & 0 & 0 \\ 3 & 68 & 30 & 0 & 4 \\ 22 & 30 & 71 & 0 & 4 \\ 0 & 0 & 0 & 42 & 22 \\ 0 & 6 & 6 & 54 & 257 \end{pmatrix}} \approx \underset{\mathbf{S}}{\begin{pmatrix} 0.3069 & 0.0000 \\ 0.2878 & 0.0042 \\ 0.3834 & 0.0017 \\ 0.0000 & 0.0824 \\ 0.0000 & 0.7045 \end{pmatrix}} \underset{\mathbf{R}}{\begin{pmatrix} 307.7633 & 8.4288 \\ 10.9841 & 612.4139 \end{pmatrix}} \underset{\mathbf{O}}{\begin{pmatrix} 0.3418 & 0.0000 \\ 0.3206 & 0.0064 \\ 0.4274 & 0.0029 \\ 0.0000 & 0.1347 \\ 0.0000 & 0.5936 \end{pmatrix}^T} \tag{3}$$

2.2 Result Interpretation

Under NMTF, a row vector $\mathbf{X}(i,:) \in \mathbf{X}$, which corresponds to the i^{th} service in the service space, can be represented as follows:

$$\mathbf{X}(i,:) = \sum_{p=1}^{k} \mathbf{S}(i,p)\mathbf{V}(p,:) \tag{4}$$

where $\mathbf{V} = \mathbf{R}\mathbf{O}^T$. Each entry $\mathbf{V}(p,j)$ captures the association of operation $\mathbf{o}_j$ with service community $\hat{s}_p$. $\mathbf{V}(p,:)$, a row vector of $\mathbf{V}$, captures the association of service community $\hat{s}_p$ with all operations. In this regard, $\mathbf{V}(p,:)$ can be regarded as the centroid vector of service community $\hat{s}_p$. Recall that NMTF enforces a non-negative constraints on matrices $\mathbf{S}, \mathbf{R}, \mathbf{O}$. In addition, $\mathbf{S}$ is the cluster indicator matrix with $\mathbf{S}(i,p) \in \mathbf{S}$ representing the cluster membership of $\mathbf{s}_i$ in service community $\hat{s}_p$ Therefore, a service $\mathbf{X}(i,:)$ is essentially formulated as the *additive combination* of all the service community centroids weighted by the memberships of $\mathbf{s}_i$ in these communities.

2.3 Objective Function

NMTF aims to find three low-rank non-negative matrices to approximate the original service-operation contingency matrix $\mathbf{X}$. A good approximation requires that values in $\mathbf{S}\mathbf{R}\mathbf{O}^T$ be close to the original values in $\mathbf{X}$. Considering the non-negative constraints, it is equivalent to solve the following optimization problem:

$$\min_{\mathbf{S}\geq 0, \mathbf{R}\geq 0, \mathbf{O}\geq 0} ||\mathbf{X} - \mathbf{S}\mathbf{R}\mathbf{O}^T||_F^2 \tag{5}$$

where $||\cdot||_F$ denotes Frobenius norm.

3 Semantic Extension Integration

The NMTF process proposed in Section 2 aims to fully leverage the WSDL descriptions to discover service communities. Due to the autonomous nature of Web services, it is common that different WSDL files use distinct terms to describe similar functionalities (e.g., AirlineReservation and BookFlight). Existing document clustering techniques rely on the co-occurrence of terms to identify semantically similar terms [7]. Unfortunately, most WSDL descriptions are generated from program source code written in certain programming languages. This implies that WSDL files rarely provide rich service descriptions. Due to the limited terms used in the WSDL descriptions, the semantically similar terms may have a low chance to co-occur in the WSDL corpus.

To attack this challenge, we propose to explore external information sources to extend WSDL descriptions with rich semantics. We then exploit these extended semantics to improve the accuracy of service community discovery. Some recent efforts have been devoted to leverage semantic extensions of the WSDL files to improve service discovery [11,2]. In these approaches, the semantic extensions are directly used to match users' queries or compute the semantic distances between terms. However, as motivated in Section 1, using external sources may lead to semantic extensions that are irrelevant to the original services. Using irrelevant semantics to match users' queries or compute the similarity between terms will negatively affect the service discovery accuracy.

We propose to integrate the semantic extensions of the WSDL corpus into the NMTF process, in which the original services are clustered to discover the service communities. The amalgamation of the semantic extensions and NMTF places the extended semantics into the context of the original services to improve community discovery accuracy.

3.1 Computing the Semantic Extensions of the WSDL Corpus

A number of external information sources, such as WordNet and Wikipedia, may be used to compute the semantic extensions of the WSDL corpus. However, as most WSDL descriptions originate from program source code, a lot of terms may not be proper English words. For example, the concatenation of a number of words is typically used to describe the names of operations (e.g., GeocodeByZip). Abbreviations are also commonly used in the parameters of the operations (e.g., temp for temperature). This significantly limits the effectiveness of traditional lexical references, such as WordNet, which do not include WSDL terms that are not proper English words.

One useful and powerful information source that we plan to leverage is the large volume of documents on the Web. This also allows us to exploit web search engines to effectively process the irregular and misspelled terms, which are quite common in WSDL files. We follow a procedure, which is similar to the one proposed in [16] to compute the semantic extensions of the WSDL corpus:

1. Preprocess each WSDL file ($W_{\mathbf{s}_i}$) in the corpus to identify the *functional* terms (refer to Section 4 for the details of WSDL file preprocessing). A functional term describes the functionality provided by a service.
2. Submit each functional term $t \in W_{\mathbf{s}_i}$ to a search engine and retrieve the top-k documents, $d_1, ..., d_k$.
3. Rank the terms in documents, $d_1, ..., d_k$ based on their TFIDF scores and select the top-r terms.
4. The semantic extension of $W_{\mathbf{s}_i}$ is a vector $E(W_{\mathbf{s}_i})$, which consists of the TFIDF scores of the selected top-r terms.

3.2 Semantic Extension Integration

We propose a *graph based approach* to achieve semantic extension integration. The first step is to construct a semantic similarity graph, $G = (V, E)$, which captures the semantic similarity between different services. Each vertex v_i represents the semantic extension of a service $\mathbf{s}_i$. Two vertices are connected if the similarity $\mathbf{W}(i, j)$ between services $\mathbf{s}_i$ and $\mathbf{s}_j$ is larger than a certain threshold. The edge is weighted by $\mathbf{W}(i, j)$, which is obtained via the dot-product between $E(W_{\mathbf{s}_i})$ and $E(W_{\mathbf{s}_j})$. Based on the semantic similarity graph, the underlying rationale of semantic extension integration can be specified as follows.

Rationale: If two services $\mathbf{s}_i$ and $\mathbf{s}_j$ share similar semantic descriptions (i.e., they have a large edge weight $\mathbf{W}(i, j)$ in the similarity graph), they are expected to provide similar functionalities. Hence, their corresponding cluster memberships (e.g., $\mathbf{S}(i, p)$ and $\mathbf{S}(j, p)$) are expected to be similar. ■

Therefore, $\mathbf{W}(i, j)(\mathbf{S}(i, p) - \mathbf{S}(j, p))^2$ is expected to be small for all i, j. This is equivalent to say that

$$\mathcal{R}_p = \frac{1}{2} \sum_{i,j=1}^{m} \mathbf{W}(i, j)(\mathbf{S}(i, p) - \mathbf{S}(j, p))^2$$

is small. If all k service communities are considered and through some algebra, we have

$$\mathcal{R} = \sum_{p=1}^{k} \mathcal{R}_p = \mathrm{Tr}(\mathbf{S}^T \mathbf{L} \mathbf{S}) \tag{6}$$

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{7}$$

$$\mathbf{D}(i, i) = \sum_{j} \mathbf{W}(i, j) \tag{8}$$

where $\mathbf{L}$ is the graph Laplacian of the semantic similarity graph and $\mathbf{D}$ is the degree matrix.

To integrate the semantic extensions with the NMTF process, we incorporate $\mathcal{R}$ as a regularizer into the original objective function specified in Equation (5).

Table 2. Domains of Web Services

Domain	#Service	Abbreviation
Communication	42	Comm
Education	139	Educ
Economy	83	Econ
Food	23	Food
Medical	45	Medi
Travel	90	Trav
Weapon	30	Weap

Thus, service community discovery with semantic extensions can be formulated as the following optimization problem:

$$\min_{\mathbf{S}\geq 0,\mathbf{R}\geq 0,\mathbf{O}\geq 0} ||\mathbf{X} - \mathbf{SRO}^T||_F^2 + \lambda \mathrm{Tr}(\mathbf{S}^T\mathbf{LS}) \tag{9}$$

where λ is the regularization parameter. The above optimization problem can be solved by using an iterative approach that exploits the auxiliary functions and the optimization theory [10].

4 Empirical Study

We conduct a set of experiments to assess the effectiveness of the proposed service community discovery framework. The experiments are performed based upon a real-world WSDL corpus obtained from [9]. The WSDL corpus consists of over 450 services from 7 different application domains. Table 2 lists the number of services from each domain.

We preprocess the WSDL corpus before applying the proposed service community discovery algorithm. The purpose of WSDL preprocessing aims to identify the *functional terms*, which describe the functionalities of the services. We follow a procedure which is similar to the one adopted in [20]. More specifically, preprocessing consists of four steps: *extraction, tokenization, stopword removal*, and *stemming*: (1) Extraction extracts the key components of a WSDL file including types, messages, operations, port types, binding, and port using path expressions. (2) Tokenization is to decompose the concatenated terms into simple terms (e.g., from AirlineReservation to Airline and Reservation). (3) Stopword removal removes the non-functional terms, which include not only the regular stopwords but also the WSDL specific stopwords, such as *url, host, http, ftp, soap, binding, type, get, set, request, response*, etc. (4) Stemming reduces different forms of a term into a common root form. After the functional terms are identified through preprocessing, we follow the procedure described in Section 2 to construct the service-operation contingency matrix $\mathbf{X}$.

4.1 Evaluation Metrics

The performance is assessed by comparing the community membership assigned by the proposed community discovery framework and the service domains provided by the WSDL corpus. We adopt two metrics to measure the community discovery performance: ACcuracy (i.e., AC) and Mutual Information (i.e., MI). Both AC and MI are widely used metrics to assess the performance of clustering algorithms [17,4].

AC **metric:** For a given service $\mathbf{s}_i$, assume that its assigned community membership is z_i and its domain label is y_i based on the WSDL corpus. The AC metric is defined as follows:

$$AC = \frac{\sum_{i=1}^{m} \delta(z_i, map(y_i))}{m} \tag{10}$$

where m is the total number of Web services in the WSDL corpus. $\delta(x, y)$ is the delta function that equals to one if $x = y$ and equals to zero if otherwise. $map(y_i)$ is the permutation mapping function that maps each assigned community membership to the equivalent domain label from the WSDL corpus. The Kuhn-Munkres algorithm is used to find the best mapping [14].

MI **metric:** Let D be the set of application domains obtained from the WSDL corpus and C be the service communities obtained from the proposed community discovery framework. The mutual information metric $MI(D, C)$ is defined as follows:

$$MI(D, C) = \sum_{\hat{d}_i \in D, \hat{c}_j \in C} p(\hat{d}_i, \hat{c}_j) \log_2 \frac{p(\hat{d}_i, \hat{c}_j)}{p(\hat{d}_i)p(\hat{c}_j)} \tag{11}$$

where $p(\hat{d}_i)$ and $p(\hat{c}_j)$ are the probabilities that a randomly selected service from the corpus belongs to domain $\hat{d}_i$ and community $\hat{c}_j$, respectively. $p(\hat{d}_i, \hat{c}_j)$ is the joint probability that the randomly selected service belongs to both domain $\hat{d}_i$ and community $\hat{c}_j$.

4.2 Experiment Design and Parameter Setting

We also implement two well-know clustering algorithms to compare with the proposed service community discovery framework. These algorithms are Singular Value Decomposition (SVD) based Co-clustering algorithm and k-means algorithm. The SVD based co-clustering algorithm leverages the duality between services and operations and has been demonstrated to be effective in clustering WSDL service descriptions [20]. We apply this algorithm to the service-operation contingency matrix to generate service communities. The k-means algorithm is applied to the semantic extensions of the WSDL corpus. The semantic extension of a WSDL file W_{s_i} is represented as a vector $E(W_{s_i})$, which consists of

Table 3. *AC* and *MI* Performance Comparison

Algorithm notation	*AC* (%)	*MI* (%)
NMTF + Semantics	**55.0**	**47.1**
NMTF	52.5	46.2
SVD Co-clustering	45.5	36.0
Semantic k-means	45.0	28.4

the TFIDF scores of the top-r terms returned by a web search engine. Refer to Section 3 for details about how to compute the semantic extension of a WSDL file. In addition, we also solely apply NMTF to the service-operation contingency matrix to generate service communities.

We plan to achieve the following objectives through the comparisons with the approaches described above:

- The comparison with the SVD based co-clustering algorithm and NMTF aims to justify the effectiveness of integrating external semantic information into the service community discovery process.
- The comparison with k-means clustering on the semantic extensions of the WSDL corpus aims to demonstrate that placing the extended semantics into the context of the original service can better leverage the semantics to benefit service community discovery.

We use the notation NMTF+Semantics to denote the proposed algorithm that integrates NMTF with the semantic extensions of the WSDL corpus. The notations for other algorithms are also listed in Table 3. The regularization factor λ is set to 10. We perform k-means clustering to initialize matrices $\mathbf{S}$ and $\mathbf{O}$. $\mathbf{R}$ is initialized as $\mathbf{S}^T\mathbf{XO}$ [6]. We run each algorithm 200 times and the average *AC* and *MI* are reported.

4.3 Performance Comparison

Table 3 compares the *AC* and *MI* performance of four different algorithms. NMTF+Semantics generates the best results on both *AC* and *MI* over all the algorithms. Thus, the results clearly demonstrate the effectiveness of the proposed service discovery framework. It is also worth to note that semantic k-means reports the lowest performance on both *AC* and *MI*. This also justifies that using semantic extensions without considering the context of the original services does not necessarily benefit community discovery.

To further illustrate the performance difference, Figure 2 shows the confusion matrices with the best *AC* performances from the four different algorithms. As can be seen, NMTF+semantics achieves a best *AC* of 64.4%. Figure 2 (a) shows the corresponding confusion matrix. The best *AC* achieved by NMTF, SVD Co-clustering and semantic k-means are 62.8%, 47.6%, and 52.9%, respectively. Figure 2 (b), (c), and (d) show the corresponding confusion matrices from these three algorithms, respectively. Among the four algorithms, NMTF+Semantics

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Comm	41	0	1	0	0	0	0
Econ	1	79	1	0	0	2	0
Educ	0	5	120	2	1	11	0
Food	1	0	19	0	0	3	0
Medi	0	0	16	6	8	10	5
Trav	0	0	47	0	0	43	0
Weap	0	0	30	0	0	0	0

(a)

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Comm	41	0	0	0	0	1	0
Econ	1	78	1	0	0	4	0
Educ	0	6	83	0	37	12	1
Food	1	10	0	0	0	12	0
Medi	0	0	5	10	16	9	5
Trav	0	0	24	0	0	66	0
Weap	0	0	29	0	0	1	0

(b)

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Comm	0	31	1	0	0	10	0
Econ	0	59	2	0	0	22	0
Educ	0	2	83	0	6	12	36
Food	0	0	9	0	0	13	1
Medi	0	1	13	0	4	19	8
Trav	13	3	20	5	3	41	5
Weap	0	0	2	0	0	0	28

(c)

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Comm	30	0	12	0	0	0	0
Econ	0	57	24	0	0	2	2
Educ	0	1	119	0	17	0	2
Food	1	0	22	0	0	0	0
Medi	0	0	26	6	13	0	6
Trav	0	0	60	9	0	20	1
Weap	0	0	30	0	0	0	0

(d)

Fig. 1. Confusion Matrices with the best AC performances. (a) NMTF+Semantics: $AC = 64.4\%$; (b) NMTF: $AC = 62.8\%$; (c) SVD Co-clustering: $AC = 47.6\%$; (c) Semantic k-means: $AC = 52.9\%$. Comm, Econ, Educ, Food, Medi, Trav, and Weap are the seven domains obtained from the WSDL corpus. C_1 to C_7 are the service communities discovered from the WSDL corpus.

correctly clusters the most number of services from three domains: Comm, Econ, and Educ. NMTF correctly clusters the most number of services from two domains: Medi and Trav. SVD Co-clustering correctly clusters the most number of services from the Weap domain.

One interesting observation from the confusion matrices is that none of the Food services has been correctly clustered by any of these algorithms. Most Food services are clustered as either Educ or Trav services. This may be because that the descriptions of the Food services share many common terms with Educ or Trav services. Another possible reason is due to the inappropriate definitions of the domains in the given WSDL corpus. For example, food and travel are two highly related domains and it may be hard to set a clear boundary to differentiate services that belong to these domains. In this regard, the community discovery result can provide guidance to improve the service domain definitions.

5 Related Work

We give an overview of existing works that are most relevant to the proposed approach in this section.

5.1 Service Community Discovery

A WSDL clustering technique is proposed in [8] to bootstrap the discovery of Web services. Five key features are extracted from WSDL descriptions to group Web services into functionality-based clusters. These features include content, types, messages, ports, and name of the Web service. Each feature is assigned an equal weight when computing the similarity between two services. Then, the Quality Threshold (QT) clustering algorithm is applied to cluster Web services. QT is a partitional clustering algorithm, like k-means, but does not require specifying the number of clusters. A similar service clustering algorithm is proposed by using four types of features to determine the similarity between services, including content, context, service host, and service name [12]. A weighting mechanism is used to combine these features to compute the relatedness measure between services. A service-operation co-clustering strategy is proposed in [20] to discovery homogeneous service communities from a heterogenous service space. A SVD based algorithm is adopted to achieve the co-clustering of services and operations. Experimental result on a set of real-world Web services shows that co-clustering generates communities with better quality than just applying one-side clustering (e.g., k-means or QT) on services. The proposed service community discovery framework adopts a NMTF process that also clusters services and operations simultaneously. NMTF is seamlessly integrated with the semantic extensions of the WSDL corpus to further improve the performance of service community discovery.

5.2 Service Search and Discovery

Woogle, a Web service search engine, is developed in [7] that helps service users discover their desired service operations and operations that may be composed

with other operations. Woogle exploits a clustering algorithm and association rule mining to group parameters of service operations into concept groups. The concept groups will then be used to facilitate the matching between users' queries and the service operations. Woogle aims to combine multiple sources of evidence, including description of services, description of operations, and input/output of operations, to measure similarity. A similar approach is developed in [13] for service discovery. A service aggregation graph is also proposed to facilitate service composition. A service discovery approach is proposed in [15] based on Probabilistic Latent Semantic Analysis (PLSA). This approach treats service descriptions as regular documents without considering the limited information available in these descriptions. A common issue with the above approaches is that they solely rely on the information carried by the WSDL service descriptions. The limited descriptive capacity of the WSDL files may limit the effectiveness of these approaches. Some recent efforts have investigated to exploit semantic extensions of the WSDL files to improve service discovery [11,2]. The semantic extensions are directly used to match users' queries or compute the semantic distance between terms. However, using external resources may lead to semantic extensions that are irrelevant to the original services, which may negatively affect the service discovery accuracy. This has also been justified through our experiment results.

5.3 Service Selection

Service selection aims to find a proper service provider with the best user desired quality of service (e.g., latency, fee, and reputation) [18,21,22]. The selection is conducted within a set of services that compete to offer similar functionalities. Most existing service selection approaches assume that services with similar functionalities have already been discovered. In this regard, the proposed service community discovery framework can be used to preprocess the Web service space before service selection can be performed.

6 Conclusion and Future Directions

We present a novel framework that amalgamates Non-negative Matrix Tri-Factorization (NMTF) and the semantic extensions of the WSDL corpus for service community discovery. NMTF in essence clusters services and operations simultaneously. In this way, it not only exploits the service descriptions but also leverages the duality relationship between services and operations to improve the performance of service community discovery. The amalgamation of NMTF and the semantic extensions of the WSDL descriptions places the extended semantics into the context of the service, which enable to more effectively leverage the semantics to benefit community discovery. We evaluate the proposed framework on a real-world WSDL corpus and the effectiveness has been clearly justified via the comparison with three other algorithms.

One interesting direction that we plan to explore is to include prior knowledge or background information to further improve the performance of service

community discovery. A useful type of prior knowledge is the pairwise constraint that specifies whether two services should belong to the same community or not. Such kind of prior knowledge is usually easier to get than relying on the domain experts to actually label a number of services. In this regard, it is worthwhile to investigate how to use this specific type of supervisory information to benefit service community discovery.

References

1. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc, Boston (1999)
2. Bose, A., Nayak, R., Bruza, P.: Improving web service discovery by using semantic models. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 366–380. Springer, Heidelberg (2008)
3. Bouguettaya, A., Yu, Q., Liu, X., Malik, Z.: Service-centric framework for a digital government application. In: IEEE Transactions on Services Computing, vol. 99(PrePrints) (2010)
4. Cai, D., He, X., Han, J.: Document clustering using locality preserving indexing. IEEE Trans. Knowl. Data Eng. 17(12), 1624–1637 (2005)
5. Dhillon, I.S.: Co-clustering documents and words using bipartite spectral graph partitioning. In: KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 269–274. ACM, New York (2001)
6. Ding, C.H.Q., Li, T., Peng, W., Park, H.: Orthogonal nonnegative matrix t-factorizations for clustering. In: KDD, pp. 126–135 (2006)
7. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: VLDB 2004: Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp. 372–383, VLDB Endowment (2004)
8. Elgazzar, K., Hassan, A.E., Martin, P.: Clustering wsdl documents to bootstrap the discovery of web services. In: ICWS, pp. 147–154 (2010)
9. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS 2006, pp. 915–922. ACM Press, New York (2006)
10. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791 (1999)
11. Liu, F., Shi, Y., Yu, J., Wang, T., Wu, J.: Measuring similarity of web services based on wsdl. In: ICWS, pp. 155–162 (2010)
12. Liu, W., Wong, W.: Discovering homogenous service communities through web service clustering. In: Kowalczyk, R., Huhns, M.N., Klusch, M., Maamar, Z., Vo, Q.B. (eds.) SOCASE 2008. LNCS, vol. 5006, pp. 69–82. Springer, Heidelberg (2008)
13. Liu, X., Huang, G., Mei, H.: Discovering homogeneous web service community in the user-centric web environment. IEEE T. Services Computing 2(2), 167–181 (2009)
14. Lovasz, L.: Matching Theory (North-Holland Mathematics Studies). Elsevier Science Ltd. (1986)
15. Ma, J., Zhang, Y., He, J.: Efficiently finding web services using a clustering semantic approach. In: CSSSIA 2008: Proceedings of the 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation, pp. 1–8. ACM, New York (2008)

16. Sahami, M., Heilman, T.D.: A web-based kernel function for measuring the similarity of short text snippets. In: Proceedings of the 15th International Conference on World Wide Web, WWW 2006, pp. 377–386. ACM, New York (2006)
17. Xu, W., Liu, X., Gong, Y.: Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, SIGIR 2003, pp. 267–273. ACM, New York (2003)
18. Yu, Q., Bouguettaya, A.: Framework for web service query algebra and optimization. TWEB 2(1) (2008)
19. Yu, Q., Liu, X., Bouguettaya, A., Medjahed, B.: Deploying and managing web services: issues, solutions, and directions. VLDB Journal 17(3), 537–572 (2008)
20. Yu, Q., Rege, M.: On service community learning: A co-clustering approach. In: ICWS, pp. 283–290 (2010)
21. Yu, T., Zhang, Y., Lin, K.-J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web 1(1), 6 (2007)
22. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30(5), 311–327 (2004)

WTCluster: Utilizing Tags for Web Services Clustering

Liang Chen[1], Liukai Hu[1], Zibin Zheng[2], Jian Wu[1], Jianwei Yin[1], Ying Li[1], and Shuiguang Deng[1]

[1] Zhejiang University, China
[2] The Chinese University of Hong Kong, China
{cliang,huliukai,wujian2000,zjuyjw,cnliying,dengsg}@zju.edu.cn, zbzheng@cse.cuhk.edu.hk

Abstract. Clustering web services would greatly boost the ability of web service search engine to retrieve relevant ones. An important restriction of traditional studies on web service clustering is that researchers focused on utilizing web services' WSDL (Web Service Description Language) documents only. The singleness of data source limits the accuracy of clustering. Recently, web service search engines such as Seekda! [1] allow users to manually annotate web services using so called tags, which describe the function of the web service or provide additional contextual and semantical information. In this paper, we propose a novel approach called *WTCluster*, in which both WSDL documents and tags are utilized for web service clustering. Furthermore, we present and evaluate two tag recommendation strategies to improve the performance of *WTCluster*. The comprehensive experiments based on a dataset consists of 15,968 real web services demonstrate the effectiveness of *WTCluster* and tag recommendation strategies.

1 Introduction

A service-oriented computing (SOC) paradigm and its realization through standardized web service technologies provide a promising solution for the seamless integration of single-function applications to create new large-grained and value-added services. SOC attracts industry's attention and is applied in many domains, e.g., workflow management, finances, e-Business, and e-Science. With a growing number of web services, the problem of discovering user required web services is becoming more and more important.

Web service discovery can be achieved by two main approaches: UDDI (*Universal Description Discovery and Integration*) and web service search engines. Recently, the availability of web services in UDDI decreases rapidly as many web service providers decided to publish their web services through their own website instead of using public registries. Al-Masri *et al.* show that more than 53% of the UDDI business registry registered services are invalid, while 92% of web services

[1] http://webservices.seekda.com/

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 204–218, 2011.

cached by web service search engines are valid and active [2]. Compared with UDDI, using search engine to search and discover web services becomes more common and effective.

Searching for web services using web service search engines is typically limited to keyword matching on names, locations, businesses, and buildings defined in the web service description file [14]. If the query term does not contain at least one exact word such as the service name, the service is not returned. It is difficult for users to be aware of the concise and correct keywords to retrieve the satisfied services. The keyword-based search mode suffers from low recall, where results containing synonyms or concepts at a higher (or lower) level of abstraction describing the same service are not returned. For example, a service named "*Mobile Messaging Service*" may not be returned from the query term "*SMS*" submitted by the user, even these two keywords are obviously the same at the conceptual level.

To handle the drawbacks of traditional web service search engines, some approaches are proposed. Lim *et al.* propose to make use of ontology to return an expand set of results including subclass, superclass and sibling classes of the concept entered by the user [16]. Elgazzar and Liu *et al.* proposed to handle the drawbacks of traditional search engine by clustering web services based on WSDL documents [6][12]. In their opinion, if web services with similar functionality are placed into the same cluster, more relevant web services could be included in the search result. In this paper, we propose to improve the performance of web service clustering for the purpose of more accurate web service discovery.

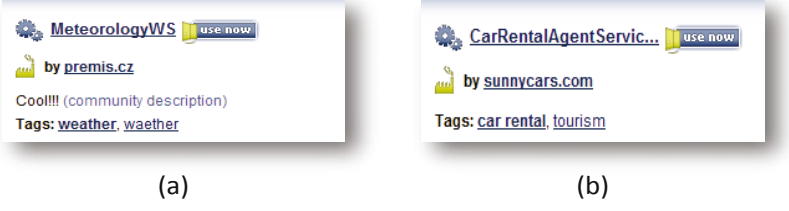


Fig. 1. Example of web services' tags

In recent years, tagging, the act of adding keywords (tags) to objects, has become a popular mean to annotate various web resources, e.g., web page bookmarks, online documents, and multimedia objects. Tags provide meaningful descriptions of objects, and allow users to organize and index their contents. Tagging data was proved to be very useful in many domains such as multimedia, information retrieval, data mining, and so on. Recently, a real-world web services search engine Seekda! allows users to manually annotate web services using tags. Figure 1 shows two examples of web services' tags in Seekda!. *MeteorologyWS* [2]

[2] `http://www.premis.cz/PremisWS/MeteorologyWS.asmx?WSDL`

in Fig. 1(a) is a web service which provides the function of weather forecasting. It has two tags, *weather* and *waether*. However, there is no word *weather* in its service name or WSDL document. Therefore, if a user uses *weather* as his query term, this service will not be retrieved without utilizing the tag information. Besides, the tag *waether* is also useful as some users may make a mistake in the typing process and use *waether* as his query term. Figure 1(b) shows another web service providing car rental information, which is very important for tourists. If we utilize the tag *tourism* in the search engine, this service will be included in the search result about tourism. From these two examples, we can find that the tagging data can help to retrieve more relevant web services.

In this paper, we don't simply use tags to match query terms, but use these tags to improve the performance of web service clustering for the purpose of more accurate web service discovery. In traditional web service clustering, features (e.g., *service name, operation, port*) are extracted from the WSDL document to form a vector, and the similarity between two web services is computed by comparing their corresponding vectors. As the words matching is still needed in the process of similarity computation, the web service *MeteorologyWS* in Fig. 1(a) can hardly be placed into the same cluster with other weather report services which have the word *weather* in their names or WSDL documents. As a service provider, he may have different naming convention and prefers to use *Meteorology* instead of *weather*. However, as a service user, he is likely to annotate the same tag to the services with similar function. Therefore, if we use the tags as part of the vectors to compute the similarities between web services, the performance of web service clustering could be improved. In our proposed *WTCluster* approach, we utilize both WSDL documents and tags, and cluster web services according to a composite similarity generated by integrating tag-level similarity and feature-level similarity between web services. Specifically, we extract 5 features from WSDL document, i.e., *Content*, *Type*, *Message*, *Port*, *Service Name*. To the best of our knowledge, this paper is the first paper to utilize the tagging data to cluster web services.

To evaluate the performance of *WTCluster*, we crawl 15,968 real web services from Seekda!. Through our observation, we find that the performance of *WTCluster* is limited by the web services which have few tags. To handle this problem, we propose two strategies to recommend some relevant tags to the services with few tags. The experiment results in Section 5 show that our tag recommendation strategies improve the performance of *WTCluster*.

In particular, the contribution of this paper can be summarized as follows:

1. We propose a novel web service clustering approach *WTCluster*, in which both WSDL documents and tags are utilized.
2. We propose two tag recommendation strategies to improve the performance of *WTCluster*.
3. We crawl 15,968 real web services to evaluate the performance of *WTCluster* and two tag recommendation strategies.

The rest of this paper is organized as follows: Section 2 highlights the related work of web service discovery and clustering. The detailed calculation of

WTCluster is introduced in Section 3, while two tag recommendation strategies are presented in Section 4. Section 5 shows the experimental results. Finally, Section 6 concludes this paper.

2 Related Work

With the development of service computing and cloud computing, web service discovery is becoming a hot research topic. A lot of work have been done to handle this problem. The approaches for discovering semantic web services and non-semantic web services are different. The semantic-based approaches adopt the formalized description languages such as OWL-S and WSMO for services and develop the reasoning-based similarity algorithms to retrieve the satisfied web services [1][3][10]. High level match-making approaches are usually adopted in the discovery of semantic web services. As non-semantic web services are more popular and supported by the industry circle, we focus on the discovery of non-semantic web services in this paper.

Some non-semantic approaches are proposed to handle the problem of web service discovery in recent years. Xin Dong *et al.* propose to compute the similarity between web services employing the structures of web services (including name, text, operation descriptions, input/output description, etc) [5]. They also propose a search engine called Woogle which supports similarity search for web services. Nayak attempts to handle the service discovery problem by suggesting the current user with other related search terms based on what other users had used in similar queries by using clustering techniques [14]. Nayak proposes to cluster web services based on search sessions instead of individual queries. Songlin Hu *et al.* make use of the content-based publish/subcribe model to handle service discovery problem [7]. Fangfang *et al.* try to reflect the underlying semantics of web services by utilizing the terms within WSDL fully [11]. In Fangfang's work, some external knowledge are firstly employed to compute the semantic distance of terms from two compared services, and then the similarity between two services is measured upon these distances.

Recently, web service clustering is presented as a novel solution to the problem of service discovery. Liu *et al.* propose to extract 4 features, i.e., *content, context, host name, and service name*, from the WSDL document to cluster web services [12]. They take the process of clustering as the preprocessor to discovery, hoping to help in building a search engine to crawl and cluster non-semantic web services. Khalid *et al.* also propose to extract features from WSDL documents to cluster web services [6]. Different from Liu's work, Khalid extracts *content, types, messages, ports, and service name* from WSDL documents.

Although these techniques are relevant, the performances of these approaches are limited by the singleness of source information as they utilize the information in WSDL documents only. In this paper, we propose to utilize both tagging data and WSDL documents to improve the performance of web service clustering. Moreover, we propose two tag recommendation strategies to handle another performance limitation caused by the web services with few tags.

3 WTCluster

In this section, we first describe our proposed framework for web service discovery in Section 3.1, and then introduce feature extraction, similarity computation and integration of *WTCluster* in Section 3.2 and Section 3.3, respectively.

3.1 Framework for Web Service Discovery

Figure 2 shows our proposed framework for web service discovery. This framework consists of two parts: 1) Data Preprocess; 2) Service Discovery. In the first part, WSDL documents and tags of web services are crawled from the Internet and used for clustering. Similar to Khalid's work[6], we extract five important features from WSDL documents, i.e., *Content*, *Type*, *Message*, *Port*, and *Service Name*. After obtaining these five features and tags of web services, we employ our proposed *WTCluster* approach to cluster web services. Since the data preprocess and clustering process is done offline, the efficiency is not a big concern, whereas the accuracy is more important. In the process of service discovery, the user first sends a query term to the web service search engine, and then the search engine returns an expanded search result by retrieving the clustered results.

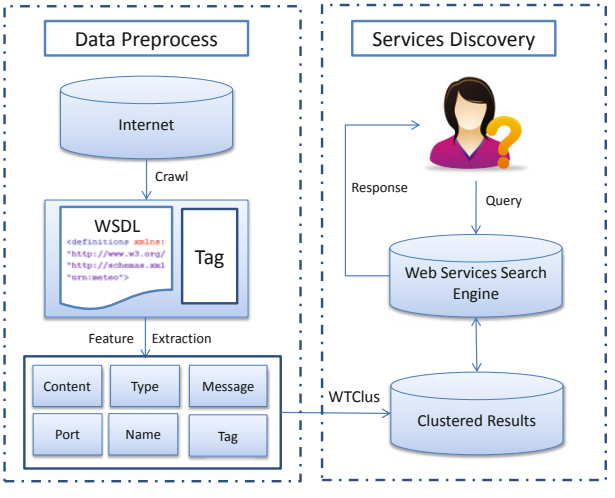


Fig. 2. Framework for web service discovery

3.2 Feature Extraction and Similarity Computation

As discussed above, we extract five features (i.e., *Content, Type, Message, Port, and Service Name*) from web service's WSDL document, and use these five features and tags to cluster web services. In this section, we describe the detailed process of feature extraction, feature-level similarity computation, and tag-level similarity computation.

Content. WSDL document, which describes the function of web service, is actually a XML style document. Therefore, we can use some IR approaches to extract a vector of meaningful content words which can be used as a feature for similarity computation. Our approach for building the content vector consists of four steps:

1. **Building original vector.** In this step, we split the WSDL content according to the white space to produce the original content vector.
2. **Suffix Stripping.** Words with a common stem will usually have the same meaning, for example, *connect*, *connected*, *connecting*, *connection*, and *connections* all have the same stem *connect* [12]. For the purpose of convenient statistics, we strip the suffix of all these words that have the same stem by using a Porter stemmer [15]. Therefore, after the step of suffix stripping, a new content vector is produced, in which words such as *connected* and *connecting* are replaced with the stem *connect*.
3. **Pruning.** In this step, we propose to remove two kinds of words from the content vector. The first kind of word to be removed is XML tag. For example, the words *s:element*, *s:complexType*, and *wsdl:operation* are XML tags which are not meaningful for the comparison of content vector. As the XML tags used in a WSDL document are predefined, it is easy to remove them from the content vector. Content words are typically nouns, verbs or adjectives, and are often contrasted with function words which have little or no contribution to the meanings of texts. Therefore, the second kind of word to be removed is function word. Church *et al.* stated that the function words can be distinguished from contents words using a Poisson distribution to model word occurrence in documents [9]. Typically, a way to decide whether a word w in the content vector is a function word is computing the degree of overestimation of the observed document frequency of the word w, denoted by n_w using Poisson distribution. The overestimation factor can be calculated as follows.
$$\Lambda_w = \frac{\hat{n_w}}{n_w}, \tag{1}$$
where $\hat{n_w}$ is the estimated document frequency of the word w. Specifically, the word with higher value of Λ_w has higher possibility to be a content word. In this paper, we set a threshold Λ_T for Λ_w ,and take the words which have Λ_w higher than threshold as content words. The value of threshold Λ_T is as follows.
$$\Lambda_T = \begin{cases} avg[\Lambda] & \text{if}(avg[\Lambda] > 1); \\ 1 & \text{otherwise} \end{cases} \tag{2}$$
where $avg[\Lambda]$ is the average value of the observed document frequency of all words considered. After the process of pruning, we can obtain a new content vector, in which both XML tags and function words are removed.
4. **Refining.** Words with very high occurrence frequency are likely to be too general to discriminate between web services. After the step of pruning, we implement a step of refining, in which words with too general meanings are

removed. Clustering based approaches were adopted to handle this problem in some related work [12][6]. In this paper, we choose a simple approach by computing the frequencies of words in all WSDL documents and setting a threshold to decide whether a word has to be removed.

After the above 4 steps, we can obtain the final content vector. In this paper, we use NGD (Normalized Google Distance) [4] to compute the content-level similarity between two web services. Given web services s_1, s_2, and their content vector $content_{s_1}$, $content_{s_2}$, the detailed equation for content-level similarity computation is as follows.

$$Sim_{content}(s_1, s_2) = \frac{\sum_{w_i \in content_{s_1}} \sum_{w_j \in content_{s_2}} sim(w_i, w_j)}{|content_{s_1}||contents_2|}, \tag{3}$$

where $|content_{s_1}|$ means the cardinality of $content_{s_1}$, the equation for computing the similarity between two words is as follows.

$$sim(w_i, w_j) = 1 - NGD(w_1, w_2) \tag{4}$$

In (4), we compute the similarity between two words using NGD based on the word co-existence in web pages. Due to space limitation, we don't introduce the detailed computation of NGD. As the number of words left in the content vector is limited after above 4 steps, the time cost for content-level similarity computation can be accepted.

Type. In a WSDL document, each input and output parameter contains a name attribute and a type attribute. Sometimes, parameters may be organized in a hierarchy by using complex types. Due to different naming conventions, the name of parameter is not always a useful feature, whereas the type attribute which can partially reflect the service function is a good candidate feature.

As Fig. 3 shows, the type of element *ProcessForm* (we name it $type_1$) is a complextype which has 5 parameters: *FormData* (string), *FormID* (int), *GroupID* (int), *szPageName* (string), and *nAWSAccountPageID* (int). If another service s_2 has a complextype $type_2$ which also contains 2 string type parameters and 3 int type parameters, we say $type_1$ and $type_2$ are matched. Specifically, in the process of type matching, the order of parameters in the complextype is not considered. We therefore extract the defined types, count the number of different types in the complextype, and compute the type-level similarity between two services using following equation.

$$Sim_{type}(s_1, s_2) = \frac{2 \times Match(Type_{s_1}, Type_{s_2})}{|Type_{s_1}| + |Type_{s_2}|}, \tag{5}$$

where $Type_{s_1}$ means the set of defined types in $s_1's$ WSDL document, $Match$ $(Type_{s_1}, Type_{s_2})$ means the number of matched types between these two services, and $|Type_{s_1}|$ means the cardinality of $Type_{s_1}$.

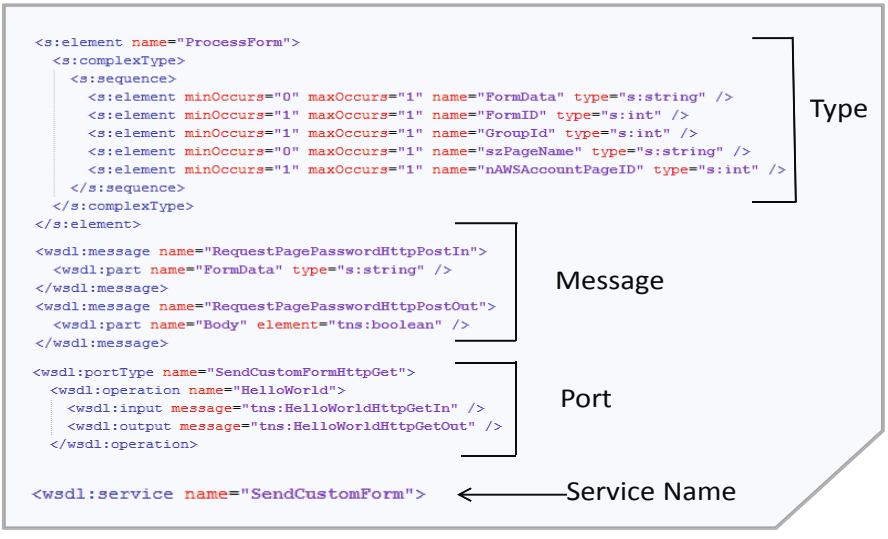


Fig. 3. Types, Message, Port, Service Name in WSDL document

Message. Message is used to deliver parameters between different operations. One message contains one or more parameters, and one parameter is associated with one type as we discussed above. Message definition is typically considered as an abstract definition of the message content, as the name and type of the parameter contained in the message are presented in the message definition. Fig. 3 shows two simple message definitions. In the first definition, the message named as *RequestPagePasswordHttpPostIn* contains one parameter *FormData* which is a *string* type. In the second definition, the message *RequestPagePasswordPostOut* contains one parameter *Body* whose type is a complextype named as *tns:boolean*. Similar to (5), we match the messages' structures to compute the message-level similarity between web services.

Port. The portType element combines multiple message elements to form a complete one-way or round-trip operation. Figure 3 shows an example of portType *SendCustomFormHttpGet* which contains some operations (due to space limitation, we only list one operation in this portType). As the portType consists of some messages, we can get the match result of portType according to the match result of messages. Similar to the computation of type-level and message-level similarity, we also use (5) to compute the port-level similarity.

Service Name. As the service name (*sname*) can partially reflect the service function, it is an important feature in WSDL document. Before computing the *sname*-level similarity, we first implement a word segmentation process to service name. For example, the service name *SendCustomForm* in Fig. 3 can be separated into three words *Send*, *Custom*, and *Form*. A simple version of word segmentation is splitting the service name according to the capital letters.

However, the performance of this simple version is not satisfied due to different naming conventions. In this paper, we first use this simple version to split the service name, and then manually adjust the final result. After the process of word segmentation, $s_1's$ name $SName_{s_1}$ can be presented as a set of words. And then we can use (3) and (4) to compute the *sname*-level similarity between web services.

Tag. The tagging data of web services describes the function of web services or provide additional contextual and semantical information. In this paper, we propose to improve the performance of traditional WSDL-based web service clustering by utilizing the tagging data. Given a web service s_i contains three tags t_1, t_2, t_3, we name the tag set of s_i as $T_i = \{t_1, t_2, t_3\}$. According to the Jaccard coefficient [8] method, we can calculate the tag-level similarity between two web services s_i and s_j as follows:

$$Sim_{tag}(s_i, s_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}, \tag{6}$$

where $|T_i \cap T_j|$ means the number of tags that are both annotated to s_i and s_j, and $|T_i \cup T_j|$ means the number of unique tags in set T_i and T_j, i.e., $|T_i \cup T_j| = |T_i| + |T_j| - |T_i \cap T_j|$.

3.3 Similarity Integration

In *WTCluster*, we use K-Means [13] clustering approach to cluster web services. K-Means is a widely adopted clustering algorithm which is simple and fast. The drawback of this algorithm is that the number of clusters has to be predefined manually before clustering. According to the six similarities calculated above, the composite similarity $CSim(s_i, s_j)$ between web services s_i and s_j is as follows:

$$CSim(s_i, s_j) = (1 - \lambda)Sim_{wsdl}(s_i, s_j) + \lambda Sim_{tag}(s_i, s_j), \tag{7}$$

where λ is the weight of the tag-level similarity, and the $Sim_{wsdl}(s_i, s_j)$ is the WSDL-level similarity which consists of five feature-level similarities between two services. The range of the value of λ is [0,1]. When $\lambda \in (0, 1)$, $CSim(s_i, s_j)$ is equal to 1 if the WSDL documents and tags of these two services are identical, and $CSim(s_i, s_j)$ is equal to 0 if both the WSDL documents and the tags of these two services are completely different. Specifically, *WTCluster* is equal to WSDL-based web service clustering approach when $\lambda = 0$, while *WTCluster* clusters web services only according to the tag-level similarity when $\lambda = 1$. We measure the WSDL-level similarity between web services s_i and s_j as follows:

$$\begin{aligned} Sim_{wsdl}(s_i, s_j) = {} & w_1 Sim_{content}(s_i, s_j) + w_2 Sim_{type}(s_i, s_j) + w_3 Sim_{message}(s_i, s_j) \\ & + w_4 Sim_{port}(s_i, s_j) + w_5 Sim_{sname}(s_i, s_j), \end{aligned} \tag{8}$$

where w_1, w_2, w_3, w_4, and w_5 are the user-defined weights of *Content, Type, Message, Port*, and *Service Name*, respectively. In particular, $w_1 + w_2 + w_3 + w_4 + w_5 = 1$.

4 Tag Recommendation

After examining the tagging data crawled from the Internet, we find the distribution of tags is not uniform. Some web services have more than 10 tags, while some ones have only 1 or 2 tags. As we compute the tag-level similarity by matching the common tags between two services, the web services with few tags lowers down the value of tag-level similarity. In this section, we propose to handle this problem by recommending a set of relevant tags to the web services with few tags.

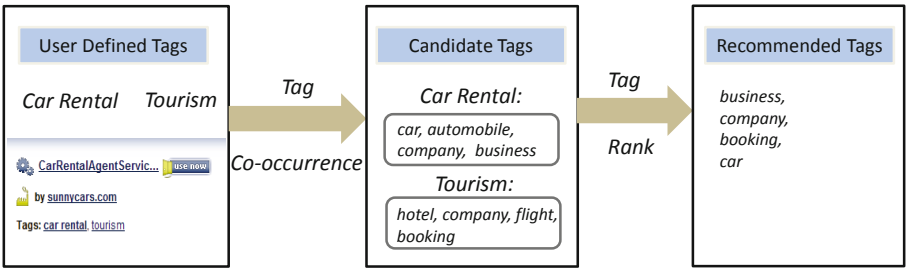


Fig. 4. Overview of Tag Recommendation Process

Figure 4 show the overview of tag recommendation process. From this figure, we can find that the process of tag recommendation can be divided into two steps. Specifically, we collect all annotated tags before the process of tag recommendation. In the first step, we first compute the co-occurrence between the user defined tags and any other tags, and then select the top-k co-occurrent tags of each user defined tag as the candidate tags. In Fig. 4, the number of k is set as 4, and the top-4 co-occurrent tags of *Tourism* are *hotel, company, flight,* and *booking*. There are some approaches to compute the co-occurrence, and we propose to use Jaccard coefficient method[8] in this paper. The detailed equation is as follows.

$$Co(t_i, t_j) = \frac{|t_i \bigcap t_j|}{|t_i \bigcup t_j|}, \tag{9}$$

where $|t_i \bigcap t_j|$ means the number of web services that have both t_i and t_j, and $|t_i \bigcup t_j|$ means the number of web services that have t_i or t_j. After the first step, for each user defined tag $u \in U$ (U is the set of user defined tags), we can get a list of candidate tags C_u.

In the second step, we rank the candidate tags and select the top-k tags as the recommended tags. In this paper, we propose two strategies to rank candidate tags.

Vote. In the *Vote* strategy, we use the idea of voting to compute a score for each candidate tag $c \in C$ (C is the set of all candidate tags). Given a candidate

tag c, we first use (10) to compute the value of $vote(u, c)$ between tag c and each user defined tag $u \in U$.

$$vote(u, c) = \begin{cases} 1 & \text{if c} \in C_u \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

After obtaining the voting result from each user defined tag, we count the voting results to get the final score by using (11).

$$score(c) = \sum_{u \in U} vote(u, c) \tag{11}$$

After obtaining all final scores, we rank the candidate tags to get the top-k recommended tags.

Sum. In the *Sum* strategy, we compute the score of the candidate tag c by summing the value of co-occurrence between c and each user defined tag u. The detailed equation is as follows.

$$score(c) = \sum_{u \in U} Co(u, c), \tag{12}$$

where the value of $Co(u, c)$ can be computed by using (9).

5 Experiment

In this section, we first compare the performances of different web service clustering approaches and then study the performances of two tag recommendation strategies .

5.1 Experiment Setup

To evaluate the performance of web service clustering approaches and tag recommendation strategies, we crawl 15,968 real web services form the web service search engine Seekda!. For each web service, we get the data of service name, WSDL document, tags, availability, and the name of service provider.

All experiments are implemented with JDK 1.6.0-21, Eclipse 3.6.0. They are conducted on a Dell Inspire R13 machine with an *2.27 GHZ Intel Core I5 CPU* and *2GB RAM*, running *Windows7 OS*.

5.2 Performance of Web Service Clustering

As the manual creation of ground truth costs a lot of work, we randomly select 200 web services from the dataset we crawled to evaluate the performance of web service clustering. We perform a manual classification of these 200 web services

to serve as the ground truth for the clustering approaches. Specifically, we distinguish the following categories: "HR", "On Sale", "Tourism", and "University". There are 31 web services in "HR" category, 26 web services in "On Sale" category, 32 web services in "Tourist" category, and 27 web services in "University" category. Due to the space limitation, we don't shows the detailed information of these web services. To evaluate the performance of web service clustering, we introduce two metrics (Precision and Recall) which are widely adopted in the Information Retrieval domain.

$$Precision_{c_i} = \frac{succ(c_i)}{succ(c_i) + mispl(c_i)}, Recall_{c_i} = \frac{succ(c_i)}{succ(c_i) + missed(c_i)}, \quad (13)$$

where $succ(c_i)$ is the number of services successfully placed into cluster c_i, $mispl(c_i)$ is the number of services that are incorrectly placed into cluster c_i, and $missed(c_i)$ is the number of services that should be placed into c_i but are placed into another cluster.

In this section, we compare the performances of three web service clustering approaches:

1. ***WCluster.*** In this approach, web services are clustered only according to the WSDL-level similarity between web services (calculated in (8)). This approach was adopted in some related work [6][12].
2. ***WTCluster*^{1}.** In this approach, we utilize both the WSDL documents and the tagging data, and cluster the web services according to the composite similarity calculated in (7).
3. ***WTCluster*^{2}.** In this approach, we first implement the tag recommendation process and then cluster web services using $WTCluster^1$ approach. In addition, we use the *Vote* strategy in this experiment.

Figure 5 shows the performance comparison of above 3 web service clustering approaches. For simplicity, we set $w_1 = w_2 = w_3 = w_4 = w_5 = 0.2$ and $\lambda = 0.5$. From Fig. 5, we can observe that our proposed *WTCluster* approaches ($WTCluster^1$, $WTCluster^2$) outperform the traditional *WCluster* approach both in the comparison of precision and recall. As we discussed above,

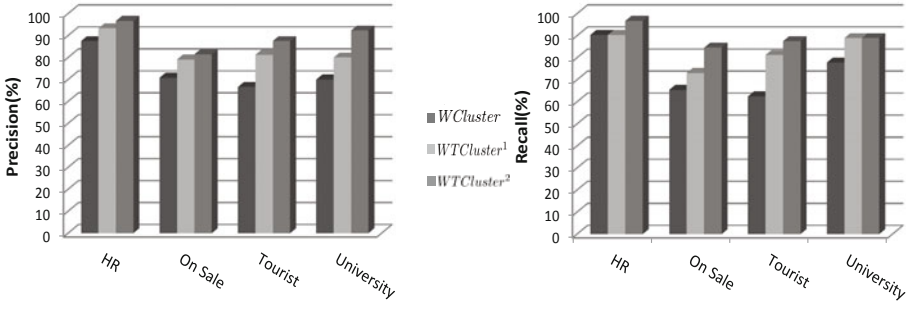


Fig. 5. Performance comparison of three web service clustering approaches

the tags of web services contains a lot of information, such as service function, location, and other semantical information. Utilizing these information improves the performance of web service clustering. Moreover, it can be observed that the approach $WTCluster^2$ which contains the process of tag recommendation outperforms the $WTCluster^1$ approach. It demonstrates that adding relevant tags to web services which have few tags can improve the performance of *WTCluster* approach.

5.3 Evaluation of Tag Recommendation Strategies

Before evaluating the performance of tag recommendation, we select 1,800 web services which contain 1254 unique tags as the dataset for evaluation. The ground truth is manually created through a blind review pooling method, where for each of the 1800 web services, the top 10 recommendations from each of the two strategies were taken to construct the pool. The volunteers were then asked to evaluate the descriptiveness of each of the recommended tags in context of the web services. We provide the WSDL documents and web service descriptions to volunteers to help them. The volunteers were asked to judge the descriptiveness on a three-point scale: *very good*, *good*, *not good*. The distinction between *very good* and *good* is defined to make the assesment task conceptually easier for the user. Finally, we get 212 *very good* judgements (16.9%), 298 *good* judgements (23.7%), and 744 *not good* judgements (59.4%).

To evaluate the performance of tag recommendation, we adopt two metrics which capture the performance at different aspects:

- **Success at rank K (S@K).** The success at rank K is defined as the percentage of *good* or *very good* tags take in the top K recommended tags, averaged over all judged web services.
- **Precision at rank K (P@K).** Precision at rank K is defined as the proportion of retrieved tags that is relevant, averaged over all judged web services.

Table 1 shows the S@K comparison of our proposed two recommendation strategies, where the *Given Tag* means the number of tags that the target web service has. Take the Sum strategy as example, when *Given Tag* varies from 1 to 2, the average value of S@K is over 0.7, which means that more than 70% recommended tags have *good* or *very good* descriptiveness. From Table 1, it can be observed that when *Given Tag* vary from 1 to 2, the performance of *Sum* strategy is better than the performance of *Vote* strategy in terms of S@K, while the performance of *Vote* strategy is better when *Given Tag* is larger than 5.

Table 2 shows the comparison of two tag recommendation strategies in terms of P@K. From Table 2, it can be observed that the value of P@K decreases when *Given Tag* increases. This is because the number of relevant tags to one certain web service is limited. When *Given Tag* increases, the number of left relevant tags decreases, which leads to the decrease of P@K. In addition, P@K achieves its largest value when K=1, and decreases when the value of K increases. It can be found that the *Vote* strategy basically outperforms the *Sum* strategy in terms of P@K.

Table 1. S@K comparison of two tag recommendation strategies

Given Tag	Method	K=1	K=2	K=3	K=4	K=5
1-2	Sum	**0.8132**	**0.7081**	**0.6738**	**0.7087**	**0.7181**
	Vote	0.6392	0.5949	0.6737	0.7005	0.6972
3-5	Sum	0.7534	0.7143	0.7380	0.6852	0.6720
	Vote	0.7867	0.6646	0.7042	0.7022	0.7103
>5	Sum	0.7632	0.7211	0.6944	0.6975	0.6647
	Vote	**0.8136**	**0.7769**	**0.7749**	**0.7262**	**0.6973**

Table 2. P@K comparison of two tag recommendation strategies

Given Tag	Method	K=1	K=2	K=3	K=4	K=5
1-2	Sum	0.6933	0.5083	0.4277	0.3788	0.3562
	Vote	**0.7879**	**0.5495**	**0.4503**	**0.3947**	**0.3689**
3-5	Sum	0.6512	0.4857	0.4171	0.3654	0.3345
	Vote	**0.7415**	**0.5414**	**0.4496**	**0.3925**	**0.3494**
>5	Sum	0.5894	0.4656	**0.4365**	0.3451	0.3508
	Vote	**0.7148**	**0.5478**	0.4105	**0.4026**	**0.3658**

6 Conclusion

In this paper, we propose to utilize the tagging data to improve the performance of web service clustering for the purpose of more accurate web service discovery. In our proposed *WTCluster* approach, we first extract five features from the WSDL document and compute the WSDL-level similarity between web services. Then, we use K-means algorithm to cluster web services according to the composite similarity which is integrated by WSDL-level similarity and tag-level similarity. To evaluate the performance of web service clustering, we crawl 15,968 real web services from the web service search engine Seekda!. The experimental results show that *WTCluster* outperforms the traditional WSDL-based approach.

Moreover, we propose two tag recommendation strategies to attack the performance limitation of *WTCluster* caused by the web services with few tags. The experiments based on real web services demonstrates that the tag recommendation process improves the performance of *WTCluster*.

Acknowledgements. This research is was partially supported by the National Technology Support Program under grant of 2011BAH15B05, the National Natural Science Foundation of China under grant of 61173176, Science and Technology Program of Zhejiang Province under grant of 2008C03007, National High-Tech Research and Development Plan of China under Grant No. 2009AA110302, National Key Science and Technology Research Program of China (2009ZX01043-003-003).

References

1. Agarwal, S., Studer, R.: Automatic matchmaking of web services. In: International Conference on Web Services, pp. 45–54 (2006)
2. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: International World Wide Web Conference, pp. 795–804 (2008)
3. Benatallah, B., Hacid, M., Leger, A., Rey, C., Toumani, F.: On automating web services discovery. The VLDB Journal 14(1), 84–96 (2005)
4. Cilibrasi, R.L., Vitnyi, P.M.B.: The google similarity distance. IEEE Transactions on Knowledge and Data Engineering 19(3), 370–383 (2007)
5. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: International Conference on Very Large Data Bases, pp. 372–383 (2004)
6. Elgazzar, K., Hassan, A.E., Martin, P.: Clustering wsdl documents to bootstrap the discovery of web services. In: International Conference on Web Services, pp. 147–154 (2009)
7. Hu, S., Muthusamy, V., Li, G., Jacobsen, H.A.: Distributed automatic service composition in large-scale systems. In: Proc. of Distributed Event-Based Systems Conference, pp. 233–244 (2008)
8. Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice Hall, New Jersey (1988)
9. Church, K., Gale, W.: Inverse document frequency (idf): a measure of deviations from poisson. In: Proceedings of the ACL 3rd workshop on Very Large Corpora, pp. 121–130 (1995)
10. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 915–922 (2006)
11. Liu, F., Shi, Y., Yu, J., Wang, T., Wu, J.: Measuring similarity of web services based on wsdl. In: International Conference on Web Services, pp. 155–162 (2010)
12. Liu, W., Wong, W.: Web service clustering using text mining techniques. International Journal of Agent-Oriented Software Engineering 3(1), 6–26 (2009)
13. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proc. of the Fifth Symposium on Math, Statistics, and Probability, pp. 281–297 (1967)
14. Nayak, R.: Data mining in web service discovery and monitoring. International Journal of Web Services Research 5(1), 62–80 (2008)
15. Porter, M.F.: An algorithm for suffix stripping. Program. 14(3), 130–137 (1980)
16. Lim, S.-Y., Song, M.-H., Lee, S.-J.: The Construction of Domain Ontology and its Application to Document Retrieval. In: Yakhno, T. (ed.) ADVIS 2004. LNCS, vol. 3261, pp. 117–127. Springer, Heidelberg (2004)
17. Zhang, Y., Zheng, Z., Lyu, M.R.: Wsexpress: A qos-aware search engine for web services. In: International Conference on Web Services, pp. 91–98 (2010)

Similarity Function Recommender Service Using Incremental User Knowledge Acquisition

Seung Hwan Ryu, Boualem Benatallah, Hye-Young Paik,
Yang Sok Kim, and Paul Compton

School of Computer Science & Engineering,
University of New South Wales, Sydney, NSW, 2051, Australia
{seungr,boualem,hpaik,yskim,compton}@cse.unsw.edu.au

Abstract. Similar entity search is the task of identifying entities that most closely resemble a given entity (e.g., a person, a document, or an image). Although many techniques for estimating similarity have been proposed in the past, little work has been done on the question of which of the presented techniques are most suitable for a given similarity analysis task. Knowing the right similarity function is important as the task is highly domain- and data-dependent. In this paper, we propose a recommender service that suggests which similarity functions (e.g., edit distance or jaccard similarity) should be used for measuring the similarity between two entities. We introduce the notion of "similarity function recommendation rule" that captures user knowledge about similarity functions and their usage contexts. We also present an incremental knowledge acquisition technique for building and maintaining a set of similarity function recommendation rules.

Keywords: Similarity Function, Recommendation, Entity Search, RDR.

1 Introduction

The community portals, such as DBLife, Wikipedia, are widely available for diverse domains, from scientific data management to end-user communities on the Web. In a community portal, data from multiple sources are integrated so that its members can search and query relevant community data. Community data typically contains different classes of entity instances, such as persons, documents, messages, and images, as well as relationships between them, such as *authorBy(person, document)*, *supervisedBy(person, person)*, and *earlyVersionOf(document, document)*. Each entity instance [1] is described by a set of attributes (e.g., person has name, title and address).

In this paper, we focus on similar entity search on such community data that is exposed as data services [7,8]. Unlike keyword based search, in similar entity search, entities are compared based on the similarity of entity attributes [2] [5,3,15]

[1] In this paper we use "entity instance" and "entity" interchangeably.
[2] We specify as attributes for short.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 219–234, 2011.

as well as entity relationships [3] [14,1], and a ranked list of entities is returned based on the degree of similarity.

A main challenge arises from the fact that entities may belong to different classes with potentially very different characteristics, and contain attributes of different data types. In such a situation, it is impractical to expect a *single* generic similarity function can work well for all attributes [3]. For example, in Figure 1(a), to measure the similarity between `q` and `m` from a discussion forum, we compare the attribute values individually: `Title` with `Title`, `Content` with `Content`, and `Size` with `Size`. When using one of the basic similarity functions (e.g., edit distance) [16,15] for all attributes and combining the scores, we obtain the final similarity score (0.55). However, the accuracy can be increased to 0.89 (Figure 1(b)) by choosing different similarity functions suited to each attribute. It is also possible to consider relationships with other entities if they exist, such as *repliedBy(message, person)*. The need for SES (Similar Entity Search) tasks is present in many application domains, such as product search, people search, document search, and data integration in business intelligence [3,20,15].

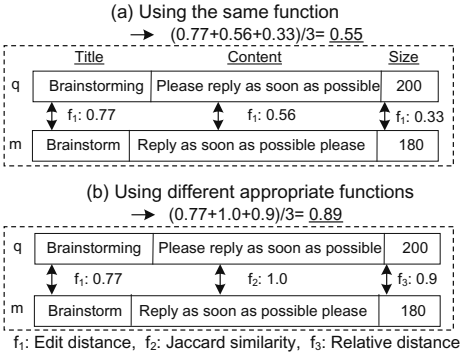


Fig. 1. Computing similarity between two messages `q` and `m`

Existing approaches for similarity analysis can be roughly divided into two groups. The first group computes attribute similarities using methods such as jaccard similarity, edit distance, or cosine similarity [16,9,5,21]. The second group exploits the relationships among entities [14,1] or machine learning-based techniques [4,5,24,23,10] for estimating the similarity of entire entities. For example, supervised machine learning techniques [5,10,24] train a model using training data pre-labeled as "matching" or "not matching" and then apply the trained model to identify entities that refer to the same real-world entity. Some of the machine learning-based techniques use positive and negative examples to learn the best combination of basic similarity functions [4,24,23].

While existing techniques have made significant progress, they do not provide a satisfactory answer to the question of which of the presented techniques should

[3] We specify as relationships for short.

be used for a given SES task. Even a technique that shows good performance for some data sets can perform poorly on new and different ones [5,15]. In real-world application domains, as in community portals, we observe that, community users, especially advanced users like programmers, administrators, and domain experts, often have valuable knowledge useful for identifying similar entities - the knowledge about which combination of similarity functions is most appropriate in which usage contexts. For instance, `edit distance` function [21] works well for comparing short strings (e.g., person names or document titles). We believe that this information is beneficial in terms of reuse and knowledge sharing as community users would choose similar functions in similar contexts.

Unfortunately, this information is not effectively exploited in existing approaches when measuring similarity. In this paper, we provide a recommender service that suggests most appropriate similarity functions for a given SES task by utilizing the knowledge collected from community users. It should be noted that our approach is complementary to the machine-learning based techniques in the sense that it allows adaptive knowledge "learning" over time as the application contexts change. Examples of such context changes are: application domain changes, dataset changes, continuous or periodic updates of datasets, and so on. In this paper we only consider the similarity of entities that belong to a same class/category. In particular, we make the following contributions:

- We introduce the notion of *similarity function recommendation rules* (henceforth recommendation rules). The recommendation rules represent the information about which similarity functions are considered most appropriate in which usage contexts (Section 3).
- We propose *incremental knowledge acquisition techniques* to build and update a set of recommendation rules. The continuous updates of recommendation rules enable the proposed recommender service to make more fine-tune recommendation (Selection 4).
- We present an implementation of the *recommender service* and provide the experimental results that show the feasibility and effectiveness of our proposed approach (Section 5).

2 Preliminaries

In what follows, we first explain the data model for representing entities and their relationships. We then describe how to measure the similarity of entities and present the overall architecture of the proposed recommender service.

2.1 Community Data Graph

We use a graph-based model, named "Community Data Graph", to represent entities and their relationships. We model the community data as a set of entities $E = \{E_1, E_2, ..., E_n\}$ and a set of relationships $R = \{R_1, R_2, ..., R_n\}$, where each E_i/R_i is an entity or a relationship category. Each entity category E_i (e.g., *Person*) has a set of entity instances (e.g., *John* and *Alice*).

Each relationship R_i (e.g., *authorBy*) has a set of relationship instances (e.g., $authorBy_{John\_firstDraft.doc}$[4]). Each entity/relationship instance consists of a set of attributes $A = \{A_1, A_2, ..., A_m\}$ and is denoted as e_i or r_i.

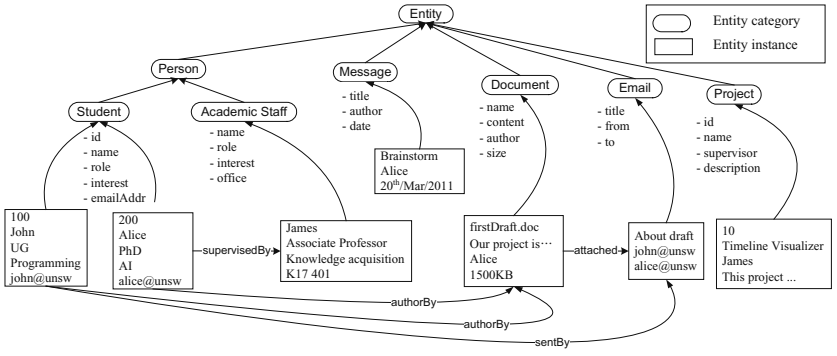


Fig. 2. Example excerpt of community data graph

Figure 2 shows a snapshot of a community data graph for an education community (called *courseWiki*)[5]. In courseWiki, the community users, such as supervisors, tutors, students, administrators, and even outside collaborators, could collaboratively work with each other and share their knowledge and experiences during the course. The courseWiki community data comes with a heterogeneous set of entities: project specifications (in Microsoft Word documents), wiki pages, emails exchanged, project reports (in PDF documents), and images/diagrams. In the graph, the nodes denote entity *categories/instances* and the edges denote *relationships* between them. For example, `John` and `Alice` are student instances of `Student` category. They can be also associated with a set of *attributes*, such as id `100` and email address `john@unsw`. The category `Student` denotes the collection of all students managed in the courseWiki community.

Definition 1. ***(Community Data graph)***
A community data graph is a direct labeled graph $G = < V, L_v, L_e, E >$*, where* V *is a set of nodes,* L_v *is a set of node labels,* L_e *is a set of edge labels and* $E \subseteq V^2 \times L_e$ *is a set of labeled edges. Each node represents an entity category or instance and each edge represents a relationship between two entities. Here, a node* $\in V$ *consists of a set of attributes* $\{A_1, A_2, ..., A_n\}$*.*

2.2 Measuring Entity Similarity

We now describe how to estimate the similarity between two entities:

Attribute-based Similarity: To measure the similarity between a query entity `q` and a same category of entity e_i, we compute the similarity between

[4] This can be specified as *authorBy* for short when there is no ambiguity.
[5] This is constructed from a project-based course "e-Enterprise Projects" in our school.

individual attributes and then produce the weighted similarity between the entities. In certain cases, a weight may be associated with each attribute, depending on its importance. Formally, we define the combined score *basic_sim* as follows:

- $basic\_sim(\mathtt{q}, e_i) = \sum_{k=1}^{N} \alpha_k f_k(\mathtt{q}.A_k, e_i.A_k)$

where f_k is the basic function being applied to a pair of attributes, α_k is its weight, and N is the number of basic functions.

Relationship-based Similarity: Apart from the attribute-based similarity, we exploit semantic relationships (called co-occurrence) between entities [14,1]. For example, two persons are likely to be similar (related), if they have co-occurring *authorBy* relationships. Like atomic attributes, relationships may have weights according to their importance (e.g., frequency of relationships). We adopt the weighted Jaccard distance to compute the co-occurrence coefficient between two entities. The weighted Jaccard distance is defined as:

- $co\_sim(\mathtt{q}, e_i) = \frac{\sum_{r \in A \cap B} w_r}{\sum_{r \in A \cup B} w_r}$

where A and B are the sets of relationships which the query entity `q` and the entity e_i have respectively, and w_r is a weight assigned to the relationship.

Composition-based Similarity: If entities have internal structures, such as XML schemas or process models, the entity similarity can be measured based on a complex process. Such a process is a directed graph that specifies the execution flow of several components [22]. The components could be a similarity estimator, a score combiner which computes a combined similarity value from results of other estimators, or a filter that identifies the most similar attribute pairs. We can integrate these measurement methods in our recommender service, if there is a need for finding correspondences between complex structures.

2.3 Overall Architecture

This subsection gives an overview of the recommender service architecture and describe components that support the concepts presented in our approach. The proposed architecture consists of the following three layers (see Figure 3).

Data service layer: To provide uniform and high-level data access to the data repository, we expose CoreDB [2] as a service by leveraging the data service technology [7,8]. CoreDB stores entities and their relationships extracted from community data sources, based on the entity-relationship model. For instance, `Person` entity table has attributes `name`, `role`, and `interest`, and stores all person entity instances. The data service layer also provides a set of CoreDB access open APIs [2], including basic CRUD operations, keyword search, rule creation, similarity functions recommendation, and so on.

Recommender service layer: This layer is composed of three main components: *function recommender*, *similarity computation* and *rule manager* components. The function recommender component takes as input `q` and

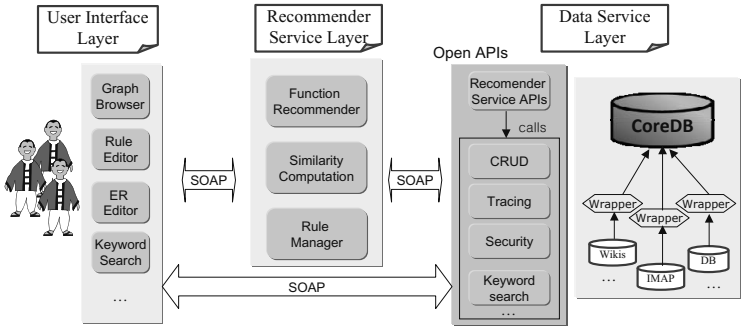


Fig. 3. Overall architecture

recommends as output a combination of similarity functions. This component accesses the recommendation rules managed in CoreDB via the open APIs. The similarity computation component asks users for a threshold (between 0.0 and 1.0) and computes the similarity scores using the recommended similarity functions. This component relies on multiple similarity functions that are represented as entities in CoreDB. The component returns and ranks the entities similar to `q` based on their final scores. The rule manager allows users to create and maintain recommendation rules. If a user is not satisfied with the returned result by the similarity computation component, she can create another recommendation rule using this rule manager. Then, she can immediately re-apply the newly added rule to get a different result.

User interface layer: At this layer, the community data accessible from data services is represented and visualized as a graph based on the mindmap metaphor [6] (the details will be described in Section 5.2). This layer is also responsible for providing various functionalities to enable users to intuitively browse and query the community data using the graph browser as well as to incrementally build recommendation rules using the rule editor.

3 Exploiting Community User Knowledge

In this section, we describe the notion of *recommendation rules* and their management. The rules represent user knowledge about similarity functions and their usage contexts. Then, we show how the recommender service makes recommendations on which functions to select, using the recommendation rules.

3.1 Recommendation Rule Representation Model

Community users, especially advanced users, often have their knowledge about the characteristics of individual similarity functions, such as usage purposes or function-specific parameters. For example, `edit distance` function [21] is expensive or less accurate for measuring the similarity between long strings (e.g., document or message contents). It is likely to be suitable for comparing short strings (e.g., document titles), capturing typographical errors or abbreviations.

Table 1. Examples of recommendation rules

RuleID	Usage Context	Function Combination	CS**
1	C* = "Message" ∧ exist(title) ∧ exist(author)	{(title, EditDistance), (author, Jaro)}	1
2	C= "Person" ∧ exist(interest) ∧ exist(role)	{(interest, Jaccard), (role, EditDistance)}	1
3	C= "Person" ∧ hasRelationship(sentEmail)	{(sentEmail, co-occurrence)}	2
4	C= "Product" ∧ exist(name) ∧ price ≤ 1500	{(name, Jaccard), (price, RelativeDistance)}	1

* C stands for Category, ** Confidence Score (see Section 3.3)

As another example, `relative distance` [3] is good for comparing numerical values, like weight and price, and `Hamming distance` [15] is used mainly for numerical fixed values, like postcode and SSN.

Thus, the effective leverage of this kind of knowledge is important to improve the accuracy of SES. In addition, community users may know which attributes/relationships play an important role in identifying similar entities. As an example, if a task is to find similar persons for a given person, attribute interest might be more useful than attributes `id`. To capture such user knowledge, we propose recommendation rules that consist of two components: *usage context* and *function combination.*

Usage context. Briefly stated, a usage context refers to the constraints that `q` should satisfy before the recommender service suggests similarity functions. It consists of a conjunction of predicates, each of which is specified by a unary or binary comparison involving entity's categories, attributes or its relationships, and constants. For example, in Table 1, the usage context of RuleID `1` states that `q` should belong to a `Message` category and have two attributes `title` and `author`. Table 2 shows some of operations that are used for specifying such usage contexts.

Function combination. For each usage context, the recommendation rule is associated with a list of pairs (attribute/relationship, similarity function) that indicates which functions are most appropriate to which attributes/relationships. For instance, in Table 1, the function combination of RuleID `1` suggests that the `edit distance` function, good for short string comparison, should be used to compare `title` and the `Jaro` function, good for name similarity detection, should be used for `author`.

Table 2. Usage Context Operations

- *exist(a_k)* checks whether `q` has an attribute a_k.
- *valueOf(a_k)* returns the value of an an attribute a_k.
- *hasRelationship(r_k)* checks whether `q` has a relationship r_k
- *length(a_k)* returns the length of a_k attribute value.
- *contain(a_k, V)* checks whether attribute a_k contains the value V.
- *belongTo(a_k, C)* checks whether the value of a_k belongs to a semantic concept C.

Definition 2. ***(Recommendation Rule)***
Let C_e be a set of entity categories supported in the recommender service. Recommendation rule is of the form: $q \in E_i,\ P(q.A_1, ..., q.A_n) \rightarrow \sum_{k=1}^{N} f_k(q.A_k)$

where P is a conjunction of predicates on the entity category and attributes A_1, ..., A_k *of q. Each predicate is either of the form* $q.category = C_i \in C_e$ *or unaryop (q.attribute/relationship) or q.attribute op value where* $op \in \{=, <, >, \leq, \geq, \neq, contain, belongTo\}$, $unaryop \in \{exist, valueOf, hasRelationship, length\}$.

3.2 Matching Recommendation Rules

When a user selects a certain entity as `q`, the recommender service identifies the potential combinations of functions being applicable to `q`. For this, the service provides an operation called `recommendFunctions()`, which takes as input `q` and produces as output a set of function recommendations that can be potentially applied to perform the corresponding SES task. The recommender service matches `q` against the usage contexts of a set of recommendation rules. The matching process relies on subsumption (containment) or equivalence between `q` and entity contexts. For example, given a query entity `q`: (category: `Student`, name: `John`, interest: `programming`, role: `undergraduate student`), the usage context of RuleID `2` is matched as the `Student` category is a sub-type of `Person` category (in Figure 2). The function combination of the matched recommendation rule is returned to the user. If no combination is found to be appropriate to `q`, the user might create a new recommendation rule with the help of rule editor.

3.3 Ranking Recommendation Rules

In our recommender service, each recommendation rule can be associated with a positive value, called *Confidence Score* (CS). The score indicates the level of credence in a corresponding recommendation rule. In fact, a CS reflects the user satisfaction level for the function combination in a recommendation rule. To obtain this score, we introduce a user feedback loop at the end of each SES task. After the user has applied the recommended function combination and examined the returned results by them, she can express the level of satisfaction with the recommendation rule (hence the function combination) by giving a score. The CS value for a recommendation rule is accumulated overtime, each positive feedback rating adding 1. The CS values are then used to show users a ranked list of the recommendation rules when there are more than one rule matching `q`. If no CS value is specified, we assume $cs = 1$. Also, completing the feedback loops is optional to users.

Definition 3. ***(Ranked recommendation rule)***
A ranked recommendation rule gives a confidence score $cs \geq 1$ *to a recommendation rule definition:* $q \in E_i, P(q.A_1, ..., q.A_n) \xrightarrow{cs} \sum_{k=1}^{N} f_k(q.A_k)$.

4 Incremental Knowledge Acquisition

In this section, we present how to incrementally obtain the recommendation rules from community users.

4.1 Knowledge Acquisition Method: Ripple Down Rule

To incrementally build and update recommendation rules, we adopt the knowledge acquisition method called Ripple Down Rule (RDR) [12] for several reasons: (i) RDR provides a *simple and easy* approach to knowledge acquisition and maintenance [18]; (ii) RDR works *incrementally*, in that users can start with an empty rule base and gradually add rules while processing new example cases.

In RDR, a rule has two components a condition and a conclusion: if [condition] then [conclusion]. Hence, the condition part of an RDR rule is mapped to the usage context in our recommendation rule, and the conclusion part to the function combination (with a confidence score). RDR organizes our recommendation rules as a tree structure. For example, Figure 4 shows an example rule tree, in which the rules are named *rule 0, rule 1, rule 2,* etc., according to their creation order. Rule 0 is the default root rule that is always fired for every query entity `q`. The rules underneath rule 0 are more specialized rules created by modifying rule conditions. The *rule inference* in RDR starts from the root node and traverses the tree, until there are no more children to evaluate. The conditions of nodes are examined as a depth-first traversal, which means the traversal result is the conclusion node whose condition is *lastly* satisfied.

4.2 Acquiring Knowledge through Different Rule Types

In what follows, we describe the incremental knowledge acquisition process using different rule types.

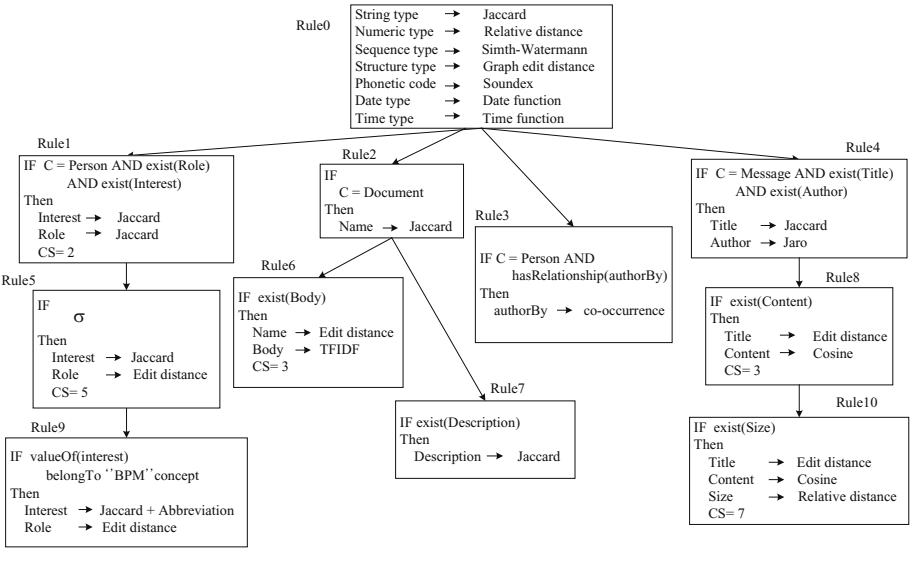


Fig. 4. Example RDR Tree: we omit duplicate conditions between parent and children rules for simplicity. If a confidence score (cs) is not available in the rule conclusion, $cs = 1$ is assumed. σ denotes the condition is the same as the parent one.

Attribute Type-Based Rule (Default Rule). The default rule contains no condition (i.e., it is always satisfied) and its conclusion consists of a list of pairs (attribute type, name of the suitable function). This rule only checks the types of attributes, meaning for example, all string-type attributes will be assigned the same string function (i.e., jaccard similarity function).

Example 1. In Figure 4, **Rule 0** specifies that `Jaccard` function is applied to string type attributes, `RelativeDistance` to numeric, `SimithWatermann` to attributes having sequence (e.g., biological sequence), and so on.

Key Attribute-based Rule. We note that there are some situations where choosing a few key attributes in an entity for comparison, rather than looking at all available attributes, may produce better results. For example, for `Person` entities, the attribute such as id may not be significant in determining similarity, but interest or role may provide a better clue. Using this rule, the user can identify any key attributes in an entity that she wants to compare. There are two possibilities in defining this rule:

Choosing key attributes only: the user identifies the attributes that play an important role for assessing the similarity without degrading the search accuracy. In this case, each attribute is paired with the default similarity function based on its type.

Example 2. **Rule 1:** $q \in E_i$, category= "Person" $\wedge$ exist(interest) $\wedge$ exist(role) $\rightarrow f_{Jaccard}(interest) \wedge f_{Jaccard}(role)$.

Choosing key attributes and functions: In this case, the user can choose key attributes as well as their similarity functions. Note that this may happen incrementally if the user determines later that a different function may produce better results. For example, in Rule 5, after looking at the results from Rule 1, she may realise that `edit distance` is better suited for `role` attribute, because the character-based function (`edit distance`) works better than the token-based function (`jaccard`) in detecting the similarity between strings including abbreviations, e.g., `Assoc. Professor` vs. `Associate Professor`.

Relationship-based Rule. Entity relationships can also contribute to analysing similarity. Out of all relationships linked with `q`, this rule allows the user to examine co-occurring ones only. Further, the user can specify the co-occurring relationships that are perceived more important. For example, the following example finds persons who have co-occurring relationship `authorBy`.

Example 3. **Rule 3:** $q \in E_i$, category= "Person" $\wedge$ hasRelationship(authorBy) $\rightarrow f_{co-occurrence}(authorBy)$

Lexical Relation-based Rule. This rule type takes into consideration the values of attributes where certain keywords/phrases or semantic information may play a role in determining similarity. For instance, consider two strings "BPM"

and "Business Process Modelling and Management" of `interest` attribute in person entities. When normal string comparison function may fail to see the similarity, it is certainly desirable to be able to match the two as 'similar'.

To handle the semantic relationships between attribute values, this rule allows the users to specify lexical relations between words (e.g., synonym, hyponym) or abbreviation. We use synonym and abbreviation tables, including domain-specific terms. An example of the abbreviation table entry is: (*BPM*: Business Processes, Business Process Management, Business Process Modelling and Management)), which takes the format of (concept name: list of terms). For instance, Rule 9 states that if `q` has `interest` attribute and its value belongs to concept name "*BPM*", then, for `interest`, apply $f_{Jaccard}$ function in comparing syntactical differences and use `abbreviation` function in comparing semantic differences.

Example 4. **Rule 9:** $q \in E_i$, category= "Person" $\wedge$ exist(interest) $\wedge$ exist(role) $\wedge$ interest belongTo `BPM` $\rightarrow$ $(f_{Jaccard}(interest) \vee f_{Abbreviation}(interest)) \wedge f_{EditDistance}(role)$.

5 Implementation, Usage, and Evaluation

This section describes our prototype implementation, usage scenario and experiment results.

5.1 Implementation

The prototype has been implemented using Java, J2EE technologies, and some generic services from existing Web services environments to implement specific functionalities of the services proposed in our approach. We extract coureWiki community data from multiple data sources and store the data into CoreDB. For this, we have implemented a number of wrappers in which each wrapper has a particular purpose and pulls the data from its original location to populate the community data graph. For instance, a special wrapper would analyse email exchange logs and build relationships such as *sentEmail*, *repliedBy*. We expose CoreDB as data services [7,8] that allow uniform data access via open APIs [2]. We have developed the recommender service supporting our proposed approach, which consists of three components: the function recommender, similarity computation, and rule manager components (Section 2.3). Table 3 shows a set of operations that such components can invoke to perform their specific functionalities. We also present a graphical user interface (Section 5.2) that allows users to interact with the recommender and data services.

5.2 Usage Scenario of the Recommender Service

We propose the following scenario as an illustration. Figure 5 presents a screenshot of the graph browser. Initially, in the visualization area, an *entity*

Table 3. The list of operations invoked by recommender service components

Function recommender/rule manager operations
- *recommendFunctions(q)* returns a list of similarity functions according to q' context.
- *createRule(c, d)* creates a rule with a condition c and a conclusion d.
- *refineRule(r, c, d)* refines a rule r with a condition c and a conclusion d.
- *rankRule(r)* increases the confidence score of a rule r by 1.
Similarity computation operations
- *computeSim(a_k,f_i)* computes similarity scores by applying function f_i to attribute a_k.
- *aggregateSim(q, e_i)* aggregates similarity values between individual attributes of q and e_i and computes a final score.

node (e.g., root entity) serving as a center node is displayed. The center node is directly connected with other nodes denoting entity categories or instances. A user can choose one of entity instances as q. The top left panel is used for navigating the graph according to the entity categories and the bottom left panel displays the details of the selected query entity (if any). After selecting q, she asks the recommender service for similarity functions. The service returns function combinations according to the recommendation rules that q satisfies. One recommendation rule example is:

– IF C= Person and exist(interest) and exist(role)
  THEN interest → Jaccard, role → Edit distance

Next, the user chooses one of function combinations, the similarity computation component calculates the similarity scores between q and the other entities, using the recommended functions, and then returns a list of similar entities. The right top panel shows the list of returned entities and the right bottom panel shows the details of a returned entity selected by the user. Here, the user can examine why the selected entity is similar to q.

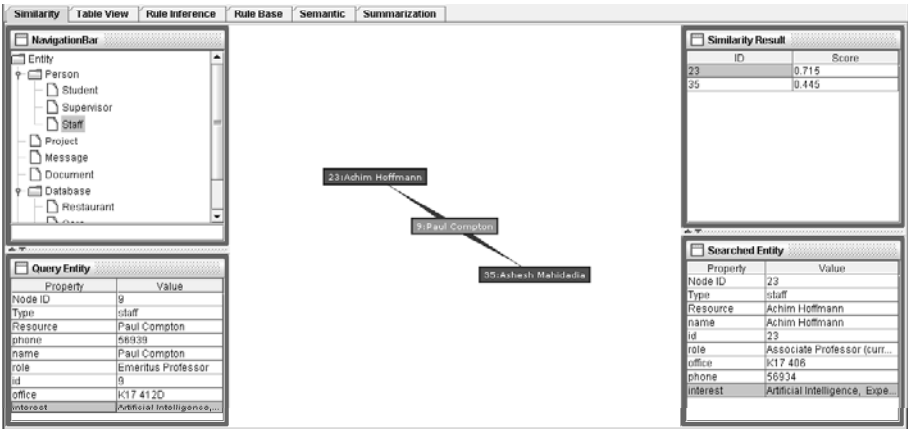


Fig. 5. Usage scenario of recommender service: the service returns entities similar to q using recommended functions

5.3 Evaluation

We now present the evaluation results that show how the recommender service can be effectively utilized in the courseWiki domain.

Dataset: We uses the dataset from our project-based course for the courseWiki community. The dataset is a collection of different categories of entities: 210 persons, 684 messages, 942 documents, 580 issues and 22874 events. For the evaluation, we applied our system on three categories of entities (i.e., `Person`, `Message`, and `Document`).

Evaluation Metrics and Methodology: We measured the overall performance with accuracy. Accuracy is the number of correctly identified similar pairs divided by the total number of identified similar pairs. Whether returned entities are similar to `q` is dependent on users' subjective decision. Therefore, we decided to manually pre-classify the entities into different groups (i.e., within a group, the entities are considered similar). All entities in one group is pre-labelled with the same groupID. For example, for `Person` entities, we grouped them according to their project/assignment work groups. `Document` entities were grouped based on their revision history.

Starting with a default rule, we began the knowledge acquisition process by looking at the different categories of entities in chronological order (instance creation date). For example, for 210 people entity instances, the acquisition process is defined as follows, note that this process is repeated for every entity instance: (i) an entity instance is selected as `q`, (ii) rules are applied, (iii) we examine the result[6], if the result is satisfactory (i.e., the groupID of `q` is the same as that of the returned entity), the rule is untouched, if not, the existing rule is refined (e.g., changing the function). The above steps are repeated until all entity instances are considered as `q`.

Results: Figure 6 shows that overall, across all categories, the accuracy of our system improves overtime as the number of entity instances being processed is increased. This is because there are more (refined) rules created. Some other observations we made about different categories are: (i) for `Person` entities, the relationships played an important role in improving the accuracy. As shown in Figure 6, with only about 60 number of entity instances considered, the system already performed at accuracy 0.93 (in this case, the number of created rules is 4). (ii) for `Document` entities, knowing the right function to use for a certain attribute was a particularly important factor. For example, comparing `title` attribute worked better with character-based string match function.

Discussion: It should be noted that the number of rules created depends on how well the users know about the characteristics of the dataset and available functions. In addition, the RDR approach for knowledge acquisition enables domain experts or users to build rules rapidly since there is no need for understanding the knowledge base as a whole or its overall structure [13].

[6] In fact, we consider the entity returned with the highest similarity score.

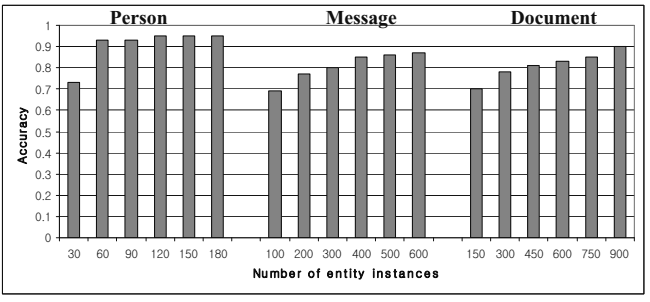


Fig. 6. Results of Evaluations

6 Related Work

Our recommender service is related to the efforts in basic similarity functions and record linkage.

Basic Similarity Functions. In this group of work, individual attributes are considered for similarity analysis. Many different basic functions for capturing similarity have been proposed in the last four decades [16,9,5,21]. For example, they are used for comparing strings (e.g., edit distance and its variations, jaccard similarity, and tf-idf based cosine functions), for numeric values (e.g., Hamming distance and relative distance), for phonetic encoding (e.g., Soundex and NYSIIS), for images (e.g., Earth Mover Distance), for assessing structural similarity (e.g., graph edit distance and similarity flooding), and so on. In contrast to those techniques, our work focuses on determining which similarity functions are most appropriate for a given similarity search task.

Record Linkage. The record linkage problem has been investigated in research communities under multiple names, such as duplicate record detection, record matching, and instance identification. The approaches can be broadly divided into three categories: supervised methods, unsupervised methods, rule-based methods. The supervised methods [5,24,11] train a model using training data pre-labeled as "match" or "no match" and later apply the model to identify records that refer to the same real-world object. The unsupervised methods [25] employ the Expectation Maximum (EM) algorithm to measure the importance of different elements in feature vectors. The rule-based approaches [19,17] enable domain experts to specify matching rules that define whether two records are the same or not. However, our work differs in that (i) we do not rely on the existence of training data. (ii) our recommender service helps users incrementally define recommendation rules and enables them to choose similarity functions suitable for the domain-specific similarity search task.

7 Conclusion and Future Work

In this paper, we presented a recommender service that suggests most appropriate similarity functions, which can be used when comparing two entities. Particularly, we introduced similarity function recommendation rules and their types. We also proposed an incremental knowledge acquisition process to build and manage the rules. In future work, we plan to investigate how to extend our approach to support large scale of SES tasks, such as identifying similar entities from millions of entities, using some high performance computing techniques.

References

1. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB, pp. 586–597 (2002)
2. Báez, M., Benatallah, B., Casati, F., Chhieng, V.M., Mussi, A., Satyaputra, Q.K.: Liquid Course Artifacts Software Platform. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 719–721. Springer, Heidelberg (2010)
3. Bilenko, M., Basu, S., Sahami, M.: Adaptive product normalization: Using online learning for record linkage in comparison shopping. In: ICDM, pp. 58–65 (2005)
4. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD, pp. 39–48. ACM (2003)
5. Bilenko, M., Mooney, R.J., Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: Adaptive name matching in information integration. IEEE Intelligent Systems 18(5), 16–23 (2003)
6. Buzan, T., Buzan, B.: The mind map book. BBC Active (2006)
7. Carey, M.: Data delivery in a service-oriented world: the bea aqualogic data services platform. In: SIGMOD 2006, pp. 695–705 (2006)
8. Castro, P., Nori, A.: Astoria: A programming model for data on the web. In: ICDE, pp. 1556–1559 (2008)
9. Christen, P.: A comparison of personal name matching: Techniques and practical issues. In: ICDM Workshops, pp. 290–294 (2006)
10. Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.: Efficient data reconciliation. Inf. Sci. 137, 1–15 (2001)
11. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: KDD, pp. 475–480 (2002)
12. Compton, P., Jansen, R.: A philosophical basis for knowledge acquisition. Knowl. Acquis. 2(3), 241–257 (1990)
13. Compton, P., Peters, L., Lavers, T., Kim, Y.S.: Experience with long-term knowledge acquisition. In: K-CAP, pp. 49–56 (2011)
14. Dong, X., Halevy, A.Y., Madhavan, J.: Reference reconciliation in complex information spaces. In: SIGMOD Conference, pp. 85–96 (2005)
15. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Trans. Knowl. Data Eng. 19(1), 1–16 (2007)
16. Hall, P.A.V., Dowling, G.R.: Approximate string matching. ACM Comput. Surv. 12, 381–402 (1980)
17. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Min. Knowl. Discov. 2, 9–37 (1998)

18. Ho, V.H., Compton, P., Benatallah, B., Vayssière, J., Menzel, L., Vogler, H.: An incremental knowledge acquisition method for improving duplicate invoices detection. In: ICDE, pp. 1415–1418 (2009)
19. Lee, M.L., Ling, T.W., Low, W.L.: Intelliclean: a knowledge-based intelligent data cleaner. In: KDD, pp. 290–294 (2000)
20. Li, Q., Wu, Y.-F.B.: People search: Searching people sharing similar interests from the web. J. Am. Soc. Inf. Sci. Technol. 59(1), 111–125 (2008)
21. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453 (1970)
22. Peukert, E., Eberius, J., Rahm, E.: Amc - a framework for modelling and comparing matching systems as matching processes. In: ICDE, pp. 1304–1307 (2011)
23. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: KDD, pp. 269–278 (2002)
24. Tejada, S., Knoblock, C.A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: KDD, pp. 350–359 (2002)
25. Winkler, W.E.: Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. In: Survey Research Methods Section, American Statistical Association, pp. 667–671 (2000)

Revealing Hidden Relations among Web Services Using Business Process Knowledge

Ahmed Awad and Mohammed AbuJarour

Hasso-Plattner-Institut, University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract. The wide spread of Service-oriented Computing and Cloud Computing has been increasing the number of web services on the Web. This increasing number of web services complicates the task of service discovery, in particular because of lack of rich service descriptions. Relations among web services are usually used to enhance service discovery. Formal service descriptions, logs of service invocations, or service compositions are typically used to find such relations. However, using such sources of knowledge enables finding simple relations only. In a previous work, we proposed to use business processes (BPs) to refine relations among web services used in the configurations of these BPs. That approach was limited to web services directly consumed by a *single* business process. In this paper, we generalize that approach and aim at predicting rich relations among web services that were not directly used together in any process configuration yet. To achieve this goal, we take all individual business processes (from a business process repository) and their configurations over web services (from a service registry) in the form of so-called extended behavioral profiles. These disparate profiles are then merged so that a single global profile is derived. Based on the aggregated knowledge in this global profile, we reveal part of the unknown relations among web services that have not been used together yet. We validate our approach through a set of experiments on a collection of business processes from SAP reference model.

Keywords: service discovery, behavioral profiles, business process configurations.

1 Introduction: Relations among Web Services

The increasing number of web services is one of the main factors that make *service discovery* a major challenge in Service-oriented Computing [3,4,11]. Finding relations among web services has been used to handle this challenge. The traditional techniques that have been proposed to achieve this task are input-output matching of web services using their technical service descriptions [8], semantic approaches [13], using compositions of web services [5], and using consumer-consumer similarity [15] (Sec. 2). The main goal of these approaches is finding any type of relations between web services, i.e., their outcome is whether two web services are related or not. No further refinement of these found relations is suggested. Additionally, these approaches are not able to find *indirect* relations among web services, because they use knowledge about web services that are used together only. Relations between web services that are not used together remain missing.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 235–249, 2011.

Missing relations between web services do not necessarily indicate their independence. Several reasons can lead to such hidden relations, such as lack of knowledge about web services and their functionalities, multiple web services with equivalent functionalities, and non-functional requirements (e.g., price, quality). We consider such missing relations as *hidden* ones and aim at revealing (part of) them. One approach of revealing such hidden relations is using knowledge concealed in the configurations of business processes that use these web services. Each configuration is considered as an identifier for its tasks and its web services. Multiple tasks that have different labels are similar if they are bound with the same web service. Similarly, web services that have different names are considered similar if they are bound with tasks that share the same label.

In our previous work [2], we introduced an approach to discover advanced relations among web services in the form of linkage patterns using business process knowledge represented as behavioral profiles [19]. Linkage patterns are generated among web services that are used in the same business process. As different consumers use web services in multiple business processes with different relations among them, multiple configurations over the same set of web services appear. These configurations are *local* to each individual process. In this paper, we develop an approach to derive a *global* behavioral profile over the entire set of web services in a service registry and reveal *hidden* relations among web services – that have never been used together by investigating indirect relationships among them – within this global profile.
The main contributions of this paper are:

- An approach to merge behavioral profiles of business processes into a global behavioral profile.
- An approach to reveal hidden relations among web services that have not been used together, yet.

To validate our approach, we use a set of business processes from the SAP reference model [7]. These models represent possibilities to configure SAP R/3 ERP systems. Thus, it is analogous to business process configurations over a service landscape.

The rest of the paper is organized as follows: We summarize related work in Sec. 2. After that, we give preliminary concepts and definitions in Sec. 3. Then, we introduce our definition of extended behavioral profiles in Sec. 4. In Sec. 5, we show our approach to derive a global behavioral profile and resolve its unknown relations. We present a set of experiments to evaluate our approach in Sec. 6. In Sec. 7, we summarize this paper and show our future work.

2 Related Work

Finding relations among web services has been considered by several researchers in the community. Approaches that tackle this problem can be grouped roughly in four groups:

- **Input/output matching approaches:** These approaches match inputs and outputs of operations of web services to find relations among them [8]. They assume the existence of complete, rich, and correct service descriptions, e.g., `WSDL`. However, such assumption is not always realistic [10]. Additionally, experiments have shown that `WSDL` descriptions only are not sufficient [17]. Additionally, these approaches may lead to unrealistic/unusable relations, and misses relations between web services with *manual* tasks in between. The main goal of these approaches is to investigate composability among web services [14,16].
- **Semantic approaches:** These approaches apply Artificial Intelligence planning techniques to find valid compositions of web services [12,13]. They are based on the assumption that web services are described formally using ontologies, such as `OWL-S`, `WSMO`, etc. In practice, semantic web services are not common [6]. Our approach does not necessarily require the existence of such formal descriptions. However, their existence could enable it to find additional relations.
- **Service compositions-based approaches:** These approaches are based on the idea that web services used in a service composition are related [5,20], i.e., they give only binary decisions. However, these approaches are not able to specify the type of relations between web services based on their usage in service compositions. These approaches depend on the co-occurrence of web services in service compositions to decide that there is a relation among them. On the other hand, Eshuis et al. [9] discover related web services based on structural matches of BPEL processes. To find related composite services with respect to a query, the behavioral relations among component services are compared to those in the query and ranked according to some heuristics. Compared to our approach, Eshuis et al.'s approach is unable to reveal hidden relations among web services that were never used in the same process model.
- **Consumer-consumer similarity approaches:** These approaches use the idea that similar service consumers usually use the same web services [15]. However, they assume the ability to track web services used by service consumers. This setting is not the typical one in practice. Moreover, it has been shown that similar service consumers do not necessarily use the same web services.

3 Preliminaries

A business process model consists of a set of tasks and theie execution ordering. Such execution ordering specifies generally which tasks are executed in sequence, concurrent or alternative to each other. Tasks can be *manual* or *service* tasks. The former are performed by humans, whereas the latter are executed by web services. Process models have special types of tasks that determine the start and end points of a process instance, respectively. Moreover, explicit control routing nodes are used to describe the above mentioned execution ordering. Thus, we represent a process model as a graph whose nodes are typed. The definition of a process model is given in Definition 1.

Definition 1 (Process Model). *A process model P is a connected, directed graph* (N, E) *where* $N = T \cup G \cup \{n_{in}, n_{out}\}$ *is a finite set of nodes, with Tasks T, Gateways G, and exactly one start and end event* n_{in} *and* n_{out}*, and* $E \subseteq (N \setminus \{n_{out}\}) \times (N \setminus \{n_{in}\})$ *is a set of edges connecting the nodes.*

A behavioral profile represents an abstract description of a business process, where it identifies a behavioral relationship between any pair of nodes within that process [19]. This relationship can be: (1) strict order $\rightsquigarrow$, (2) concurrent $\|$, (3) exclusive # or (4) inverse order $\leftsquigarrow$. The formal definition of behavioral profiles is given in Definition 2.

Definition 2 (Behavioral Profile). *Let N be the set of nodes within a business process model. The* behavioral profile *of a business process model is a function* $bhp_{bp} : N \times N \rightarrow \{\rightsquigarrow, \leftsquigarrow, \|, \#\}$ *that assigns a behavioral property, strict order, inverse order, parallel, or exclusive, between each pair of nodes within the business process model.*

If two tasks a, b appear in strict order within a business process x, $bhp_x(a, b) = \rightsquigarrow$, then task a executes before task b. Similarly, if two tasks are concurrent then they can be executed in any order. Exclusiveness means that at most one of the two tasks can execute within a process instance. To derive useful behavioral properties between tasks of a BP, we remove cyclic edges from such BPs because their existence makes all connected tasks concurrent.

Transforming business process models into executable processes is done through a *configuration* step. In this step, business engineers assign web services to *service* tasks in the considered model. This assignment represents a service discovery and selection task. The formal definition of BP configuration is given in Definition 3.

Definition 3 (BP Configuration). *A service registry usually contains a collection of web services*[1]*. The* configuration *of a business process model – containing tasks T – is a function* $conf : T \rightarrow WS_i$ *that assigns a web service* (WS_i) *for each* service *task in that business process model.*

Based on these behavioral profiles, we derive linkage patterns among web services [2]. Therefore, deriving a *consistent* global behavioral profile is a key to avoid inconsistencies among linkage patterns and to predict useful relations among web services that have not been used together yet, i.e., do not appear in the same BP.

4 The Extended Behavioral Profile

Revealing hidden relations among web services requires a global behavioral profile, where all services in the considered registry are involved. A global profile is the result of *merging* all individual behavioral profiles of available business processes. Merging two relations from two profiles results in *unknown* relations between web services that do not appear together in one business process. Moreover, this merging step might result in *contradicting* relations, e.g., merging $a\#_x b$ and $a \rightsquigarrow_y b$. Therefore, the four basic

[1] For simplicity, we assume that each web service contains a single operation.

behavioral relations of the original behavioral profile in Definition 2 are not sufficient. We *extend* the four basic relations to capture such situations when merging individual profiles by introducing two additional relations: unknown (?) and contradicts (※). These two relations do not appear on the level of individual raw profiles. They appear only when profiles are merged as we show in Sec. 5. Additionally, we record the *distance* between tasks bound to web services in the process configuration. This distance is used in the derived linkage patterns among web services to rank services during service discovery. We obtain this distance by counting the edges on the shortest path between the nodes representing the tasks in the process graph.

In this section, we present the formal notion of *extended* behavioral profiles (Sec. 4.1). In Sec. 4.2, we introduce a business process with its extended behavioral profile that is used as a running example in the rest of this paper.

4.1 Formal Model

The original definition of behavioral profiles is concerned with behavioral relations among tasks within a business process. However, in our approach, we are interested in discovering relations among web services used to configure such business processes.

Definition 4 (Extended Behavioral Profile). *Let $\mathcal{W}$ be the set of web services within a service registry. The* extended behavioral profile *of web services in $\mathcal{W}$ is a function $xbhp : \mathcal{W} \times \mathcal{W} \rightarrow \mathcal{P}(\{\rightsquigarrow, \leftsquigarrow, \|, \#, ?, ※\} \times \mathbb{N})$ that assigns a set of pairs of a behavioral property (strict order, inverse order, parallel, exclusive, unknown or contradicts) and a distance between each pair of web services within the service registry.*[2]

Comparing Definition 4 of the extended behavioral profile with Definition 2, we notice that the behavioral relations are leveraged from the level of tasks within individual process models (configurations) to the level of web services within the service registry. Moreover, the extended profile records the distance between web services consumed within an individual profile. This distance is greater than zero if the behavioral relation is either $\rightsquigarrow$ or $\leftsquigarrow$ and zero otherwise. Finally, multiple behavioral properties can exist between two web services in the global behavioral profile (Sec. 5.1).

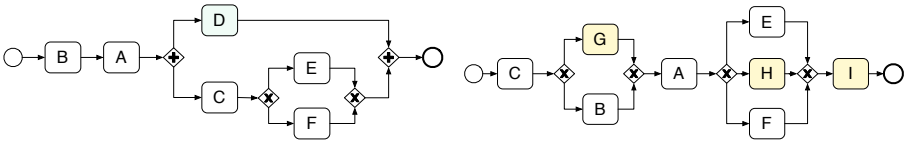
An individual behavioral profile (Definition 2) of a process can be turned into an extended profile by adding all web services in the service registry to the services consumed by that process where their behavioral relations are set to unknown. For simplicity, we ignore these unknown relations for input behavioral profiles.

Definition 5 (Projections over an Extended Behavioral Profile). *Let $\mathcal{W}$ be the set of web services within a serivce registry and let x be an extended behavioral profile. The function $rel_x : \mathcal{W} \times \mathcal{W} \rightarrow \{\rightsquigarrow, \leftsquigarrow, \|, \#, ?, ※\}$ projects the behavioral relation between two web services a and b in the registry with respect to profile x. Similarly $dist_x : \mathcal{W} \times \mathcal{W} \rightarrow \mathbb{N}$ projects the distance between the two services with profile x. For simplicity, we express $rel_x(a, b) = \{*\}$ as $a *_x b$ in the rest of the paper.*

[2] For simplicity, we assume that each web service has only one operation.

4.2 Running Example

In Fig. 1, we introduce two anonymized business processes from the SAP reference model that are used as a running example through this paper. We use the labels $A - I$ as identifiers for *both* tasks and web services. The common anonymized labels between both business processes indicate using the same service in their configuration. BP_1 has 6 tasks where only task D is not a common task with BP_2. On the other hand, BP_2 has 8 tasks among which 3 tasks are not common with BP_1, namely G, H, and I.



(a) Business Process BP_1 (b) Business Process BP_2

Fig. 1. Two anonymized Business Processes from SAP reference model

The extended behavioral profile of BP_1 is shown in Table 1 and that of BP_2 can be generated similarly, we omit it due to space limitations. It is worth mentioning that both BPs are configured such that each task is bound with a web service to execute it. According to Table 1, $xbhp_{BP_1}(E, F) = \{(\#, 0)\}$ and $xbhp_{BP_1}(A, D) = \{(\rightsquigarrow, 2)\}$. Relations that are not shown in this profile are implicitly *unknown*, e.g., $xbhp_{BP_1}(A, G) = \{(?, 0)\}$.

Table 1. The *extended* behavioral profile of BP_1 shown in Fig. 1(a)

	A	B	C	D	E	F
A	(‖, 0)	(⇜, 1)	(⇝, 2)	(⇝, 2)	(⇝, 4)	(⇝, 4)
B	(⇝, 1)	(‖, 0)	(⇝, 3)	(⇝, 3)	(⇝, 5)	(⇝, 5)
C	(⇜, 2)	(⇜, 3)	(‖, 0)	(‖, 0)	(⇝, 2)	(⇝, 2)
D	(⇜, 2)	(⇜, 3)	(‖, 0)	(‖, 0)	(‖, 0)	(‖, 0)
E	(⇜, 4)	(⇜, 5)	(⇜, 2)	(‖, 0)	(‖, 0)	(# , 0)
F	(⇜, 4)	(⇜, 5)	(⇜, 2)	(‖, 0)	(# , 0)	(‖, 0)

5 Deriving a Global Behavioral Profile

Knowledge about relations among web services is usually scattered in disparate profiles of business processes that are configured to use these web services. Collecting this knowledge into one single profile is essential to reveal hidden relations among these web services. We call the result of this step a *global* behavioral profile. We describe our approach to derive this global profile from individual profiles in Sec. 5.1. The global profile might include unknown or contradicting relations among some pairs of web

services. We inspect the gained knowledge in the global profile to *predict* possible resolutions for its unknown relations. We describe our approach to resolve unknown relations in the global profile in Sec. 5.2. Both steps, i.e., deriving a global profile and predicting unknown relations are of iterative nature. That is, at the point that a new process configuration is available, this new profile is merged with the global profile, to obtain a new global profile, and the prediction of unknown relations is rerun.

5.1 Merging Individual Behavioral Profiles

Given a set of behavioral profiles, we aim at deriving a global profile that contains pairwise relations between all used web services. We achieve this deriving by merging all individual profiles iteratively in a pairwise manner. The result of each merging iteration is an intermediate profile that is merged with another profile. This step is repeated until all individual profiles are incorporated. Merging individual profiles might result in unknown or contradicting relations among web services. Unknown relations appear between web services that are not used together in the same business process. Whereas contradicting relations appear due to *conflicting* relations in source profiles. For instance, the relations $(a\#_x b)$ (i.e., a and b are exclusive in profile x) and $(a \rightsquigarrow_y b)$ (i.e., a precedes b in profile y) includes at least one incorrect relation. Exclusiveness usually means that web services do similar or complementary jobs [2]. Currently, we propagate such conflicts to the resulting intermediate profile by adding two relations $(a \between_z b)$ and $(b \between_z a)$ that represent a contradiction to the resulting intermediate profile z.

We merge two relations between web services a and b that appear in both input profiles x and y into the global profile t according the set of rules that is summarized in Table 2. These rules can be grouped as follows:

1. Merging $(a *_x b)$ and $(a *_y b)$ gives $(a *_t b)$, where $*$ is the same type of relation.
2. Merging $(a \reverse{\rightsquigarrow}_x b)$ and $(a \rightsquigarrow_y b)$ gives $(a \parallel_t b)$ or $(a \between_t b)$.
3. Merging $(a *_x b)$ and $(a \bullet_y b)$ gives $(a \parallel_t b)$ or $(a \between_t b)$, where $* \in \{\rightsquigarrow, \reverse{\rightsquigarrow}\}$ and $\bullet = \parallel$.
4. Merging $(a *_x b)$ and $(a\#_y b)$ gives $(a\#_t b)$ if $* = \#$, and $a \between_t b$ otherwise.
5. Merging $(a?_x b)$ and $(a *_y b)$ gives $(a *_t b)$, where $*$ is a basic relation.
6. Merging $(a \between_x b)$ and $(a *_y b)$ gives $a \between_t b$.

Some merging rules are non-deterministic, i.e., produce multiple alternatives (See Table 2). For instance, merging $(a \rightsquigarrow_x b)$ and $(a \reverse{\rightsquigarrow}_y b)$ gives two options: $(a \parallel_t b)$ and $(a \between_t b)$. Parallelism means that there is no dependency between a and b and they can be used in any order. On the other hand, a dependency between a and b means that either profile x or y is incorrect, where it includes a data anomaly, e.g., missing data [18]. In this case, we conclude that there is a contradiction $(a \between_t b)$. To resolve such non-determinism, a human intervention is needed, which is out of scope of this paper.

The second component in the extended behavioral profile (besides the relation's type) is *distance* between services. This distance between two services in an intermediate profile is calculated as the shortest distance in the corresponding profiles unless one of both distances is zero, i.e., # or ||.

Table 2. Merging relationships from two profiles x and y into an intermediate profile t

Profile	$a \rightsquigarrow_x b$	$a \leftsquigarrow_x b$	$a \parallel_x b$	$a\#_x b$	$a?_x b$	$a\between_x b$
$a \rightsquigarrow_y b$	$a \rightsquigarrow_t b$	$a \parallel_t b$ or $a\between_t b$	$a \parallel_t b$ or $a\between_t b$	$a\between_t b$	$a \rightsquigarrow_t b$	$a\between_t b$
$a \leftsquigarrow_y b$	$a \parallel_t b$ or $a\between_t b$	$a \leftsquigarrow_t b$	$a \parallel_t b$ or $a\between_t b$	$a \between_t b$	$a \leftsquigarrow_t b$	$a\between_t b$
$a \parallel_y b$	$a \parallel_t b$ or $a\between_t b$	$a \parallel_t b$ or $a\between_t b$	$a \parallel_t b$	$a\between_t b$	$a \parallel_t b$	$a\between_t b$
$a\#_y b$	$a\between_t b$	$a\between_t b$	$a\between_t b$	$a\#_t b$	$a\#_t b$	$a\between_t b$
$a?_y b$	$a \rightsquigarrow_t b$	$a \leftsquigarrow_t b$	$a \parallel_t b$	$a\#_t b$	$a?_t b$	$a\between_t b$
$a\between_y b$	$a\between_t b$	$a\between_t b$	$a\between_t b$	$a\between_t b$	$a\between_t b$	$a\between_t b$

Example. Consider that the two processes from Fig. 1 are the only individual profiles in our knowledge base. By applying our merging rules shown in Table 2, we get the global profile shown in Table 3. This global profile has 9 web services that represent the union of all services in its source profiles, i.e., BP_1 and BP_2. For instance, merging relations ($\leftsquigarrow$, 1) and ($\leftsquigarrow$, 2) between web services A and B from BP_1 and BP_2, receptively, gives the relation ($\leftsquigarrow$, 1) in the global profile. The distance of the relation in the global profile is the minimum distance from input relations. Some merging rules produce multiple alternatives. For instance, A and C has the relation ($\rightsquigarrow$, 2) and ($\leftsquigarrow$, 4) in BP_1 and BP_2, respectively. Merging both relations gives two alternatives in the global profile between A and C: ($\parallel$, 0) and ($\between$, 0). The remaining relations can be derived in the same way. Merging extended behavioral profiles of BPs that do *not* have the same exact set of web services results in unknown relations between web services that do not appear in the same BP. For instance, relations between D from one side and G, H, and I on the other side in the global profile. In the sequel, we aim at using the knowledge gained from merging both profiles to reveal such unknown relations.

Table 3. Merging profiles of BP_1 and BP_2 (Fig. 1(a) &1(b)) in one global profile

	A	B	C	D	E	F	G	H	I
A	($\parallel$, 0)	($\leftsquigarrow$, 1)	($\parallel$, 0) ($\between$, 0)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 2)	($\leftsquigarrow$, 2)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 8)
B	($\rightsquigarrow$, 1)	($\parallel$, 0)	($\parallel$, 0) ($\between$, 0)	($\rightsquigarrow$, 3)	($\rightsquigarrow$, 4)	($\rightsquigarrow$, 4)	(# , 0)	($\rightsquigarrow$, 4)	($\rightsquigarrow$, 10)
C	($\parallel$, 0) ($\between$, 0)	($\parallel$, 0) ($\between$, 0)	($\parallel$, 0)	($\parallel$, 0)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 2)	($\rightsquigarrow$, 8)	($\rightsquigarrow$, 14)
D	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 3)	($\parallel$, 0)	($\parallel$, 0)	($\parallel$, 0)	($\parallel$, 0)	**(?, 0)**	**(?, 0)**	**(?, 0)**
E	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 4)	($\leftsquigarrow$, 2)	($\parallel$, 0)	($\parallel$, 0)	(# , 0)	($\leftsquigarrow$, 4)	(# , 0)	($\rightsquigarrow$, 2)
F	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 4)	($\leftsquigarrow$, 2)	($\parallel$, 0)	(# , 0)	($\parallel$, 0)	($\leftsquigarrow$, 4)	(# , 0)	($\rightsquigarrow$, 2)
G	($\rightsquigarrow$, 2)	(# , 0)	($\leftsquigarrow$, 2)	**(?, 0)**	($\rightsquigarrow$, 4)	($\rightsquigarrow$, 4)	($\parallel$, 0)	($\rightsquigarrow$, 4)	($\rightsquigarrow$, 10)
H	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 4)	($\leftsquigarrow$, 8)	**(?, 0)**	(# , 0)	(# , 0)	($\leftsquigarrow$, 4)	($\parallel$, 0)	($\rightsquigarrow$, 2)
I	($\leftsquigarrow$, 8)	($\leftsquigarrow$, 10)	($\leftsquigarrow$, 14)	**(?, 0)**	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 2)	($\leftsquigarrow$, 10)	($\leftsquigarrow$, 2)	($\parallel$, 0)

5.2 Predicting Potential Resolutions for Unknown Relations ($a?b$)

Merging two profiles that do not have the same set of web services results in a global profile with unknown relations among web services that do not appear in both source profiles. In this section, we describe our approach to reveal such *unknown* relations by predicting potential resolutions for them.

We predict potential resolutions for the unknown relation between web services a and b in the global profile g with the help of a common service between them, e.g., c. Our goal is to resolve the relation $(a?_g b)$ into $(a \leftsquigarrow_g b)$, $(a \rightsquigarrow_g b)$, $(a \parallel_g b)$, or $(a\#_g b)$ by investigating the relations between a and c on one hand and between b and c on the other hand. We select a common service c such that we can derive useful information from its relations with the considered services. For instance, selecting c such that $(a \divideontimes_g c)$ is not of value. Therefore, the common service c has to be in one of the basic four relations with both a and b. Furthermore, the predicted relation has to be consistent with existing relations in the global profile. Finding a useful resolution for unknown relations depends on the used knowledge, therefore it is not always possible to predict such a resolution. In such cases, the unknown relation between a and b $(a?_g b)$ in the global profile g remains and a human expert is informed about the situation to find a resolution manually.

We predict potential resolutions for each unknown relation between web services a and b in the global profile g – i.e., $(a?_g b)$ – using a common service, c, according to the set of rules that is summarized in Table 4. These rules can be grouped together as follows:

1. Order-order: resolving $(a *_g c)$ and $(b *_g c)$ gives $(a \rightsquigarrow_g b)$, $(a \leftsquigarrow_g b)$, $(a \parallel_g b)$, and $(a\#_g b)$, where $* = \rightsquigarrow$ or $* = \leftsquigarrow$. Each of these predicted relations still preserves the existing relations $(a \rightsquigarrow c)$ and $(b \rightsquigarrow c)$ or $(a \leftsquigarrow c)$ and $(b \leftsquigarrow c)$.
2. Transitivity: resolving $(a \leftsquigarrow_g c)$ and $(b \rightsquigarrow_g c)$ gives $(a \leftsquigarrow_g b)$. Any other relation, e.g., $(a \rightsquigarrow b)$, does not preserve the existing relation between a,b on the one hand and c on the other hand. For instance, $(a \rightsquigarrow b)$ means that b executes *before* a, that contradicts $(a \leftsquigarrow c)$. Similarly, we cannot deduce that $(a\#b)$ as it contradicts $(b \rightsquigarrow c)$, since that implies either $(c \leftsquigarrow b)$ or $(c\#b)$, which is not the case.
3. Branch-order: resolving $(a *_g c)$ and $(b \bullet_g c)$ gives $(b *_g a)$ and $(a \bullet_g b)$, where $* \in \{\leftsquigarrow, \rightsquigarrow\}$ and $\bullet \in \{\|, \#\}$.
4. Similar Branch-Branch: resolving $(a *_g c)$ and $(b *_g c)$ gives $(a *_g b)$, $(a \rightsquigarrow_g b)$, and $(a \leftsquigarrow_g b)$, where $* = \|$ or $* = \#$.
5. Different Branch-Branch: resolving $(a *_g c)$ and $(b \bullet_g c)$ gives $(a *_g b)$ and $a \bullet_g b$, where $* \in \{\|, \#\}$ and $\bullet \in \{\|, \#\}$, and $\bullet \neq *$.

Distances of the predicted $\rightsquigarrow$ and $\leftsquigarrow$ relations in the global profile are calculated according to the functions shown in Table 5. Distances are used to rank relevant web services during service discovery [2]. Additionally, we use them to prune possible resolutions. For some cases, the new distance is the absolute value of the difference of two distance. As an example, consider the case where we have $(a \rightsquigarrow c)$ and $(b \rightsquigarrow c)$. According to Table 4, all four basic relations are valid resolutions. For the predicted $(a \parallel b)$ and $(a\#b)$ we set distance to zero. However, for the two remaining cases, i.e. $(a \rightsquigarrow b)$ and $(a \leftsquigarrow b)$, the distance is the absolute value of the difference in input distances.

Table 4. Resolving the unknown relation $a?_g b$ via a common service c

Relation	$a \rightsquigarrow c$	$a \lessdot c$	$a \parallel c$	$a\#c$
$b \rightsquigarrow c$	$a \rightsquigarrow b$ $a \lessdot b$ $a \parallel b$ $a\#b$	$a \lessdot b$	$a \parallel b$ $a \lessdot b$	$a\#b$ $a \lessdot b$
$b \lessdot c$	$a \rightsquigarrow b$	$a \lessdot b$ $a \rightsquigarrow b$ $a \parallel b$ $a\#b$	$a \parallel b$ $a \rightsquigarrow b$	$a\#b$ $a \rightsquigarrow b$
$b \parallel c$	$a \parallel b$ $a \rightsquigarrow b$	$a \parallel b$ $a \lessdot b$	$a \parallel b$ $a \rightsquigarrow b$ $a \lessdot b$	$a \parallel b$ $a\#b$
$b\#c$	$a\#b$ $a \rightsquigarrow b$	$a\#b$ $a \lessdot b$	$a \parallel b$ $a\#b$	$a\#b$ $a \rightsquigarrow b$ $a \lessdot b$

When we have no information to calculate the distance, we set it to infinity, e.g., the case of $(a \parallel c)$ and $(b \parallel c)$. For the cases where there is no order in the predicted relation between a and b, we express this using N/A in the table.

We calculate the distance of the predicted $\rightsquigarrow$ and $\lessdot$ relations in the global profile according to these rules:

1. Order-order: The distance is defined as $|diff()|$ between distances of input relations.
2. Transitivity: The distance is defined as $sum()$ of distances of input relations.
3. Branch-order: The distance is defined as distance between web services b and c.
4. Similar Branch-Branch: The distance is defined as ∞, i.e., an artificial value.
5. Different Branch-Branch: The distance is not applicable, i.e., N/A.

Table 5. Distances of the predicted $a?b$ via a common service c

Relation	$a \rightsquigarrow c$	$a \lessdot c$	$a \parallel c$	$a\#c$
$b \rightsquigarrow c$	$\|diff()\|$	$sum()$	$dist(b,c)$	$dist(b,c)$
$b \lessdot c$	$sum()$	$\|diff()\|$	$dist(b,c)$	$dist(b,c)$
$b \parallel c$	$dist(a,c)$	$dist(a,c)$	∞	N/A
$b\#c$	$dist(a,c)$	$dist(a,c)$	N/A	∞

According to our rules of resolution shown in Table 4, possible resolutions to an unknown relation $a?b$ can include both $a \rightsquigarrow b$ and $a \lessdot b$. We use the distance information to prune one or both of these resolutions according to the following rules. Consider two relations $(a *_x b)$ and $(b \bullet_y c)$ with distances d_x and d_y, respectively, where $*$ and $\bullet$ are either $\rightsquigarrow$ or $\lessdot$, and Δd is defined as $d_x - d_y$, we identify three cases:

- $\Delta d = 0$: The unknown relation $(a?b)$ cannot be predicted to $(a \rightsquigarrow b)$ or $(a \leftsquigarrow b)$.
- $\Delta d > 0$: The unknown relation $(a?b)$ can be predicted to $(a \rightsquigarrow b)$, but not to $(a \leftsquigarrow b)$.
- $\Delta d < 0$: The unknown relation $(a?b)$ can be predicted to $(a \leftsquigarrow b)$, but not to $(a \rightsquigarrow b)$.

Table 4 shows possible resolutions of $a?b$ using one common service c. However, a and b might have a set of common services, which includes services that have useful behavioral relations ($\rightsquigarrow$,$\leftsquigarrow$,||, or #) with both a and b. We use each element in this set to predict the unknown relation between a, b according to the rules in Table 4. After that, we intersect all possible resolutions deduced from each element in that set. The result of this intersection is then used as a resolution to that unknown relation between a and b. If this intersection gives an empty set (e.g., due to contradictions), we are unable to predict resolutions for $a?b$. These steps are shown in Algorithm 1.

Algorithm 1. Predicting unknown relations in the global profile

Input: g the global profile
Output: g' the global profile with some unknown relations revealed

```
pred ← ∅
for all a?b ∈ g do
    CT ← getCommonTasks(a, b)
    for all c ∈ CT do
        ac ← rel_g(a, c)
        bc ← rel_g(b, c)
        tmp ← predictRelaton(ac, bc) {According to Tables 4 and 5}
        if pred = ∅ then
            pred ← tmp
        else
            pred ← intersect(pred, temp)
            if pred = ∅ then
                break
            end if
        end if
    end for
    g ← merge(g, pred) {According to Table 2}
end for
g' ← g
return g'
```

Example. In Table 3, we show the global profile that we get by merging the extended global profiles of BP_1 (Fig. 1(a)) and BP_2 (Fig. 1(b)). That global profile has three unknown relations between service D on the one hand and services G, H, and I on the other hand, because these services are not used in the same BP. However, BP_1 and BP_2 have other common web services, e.g., A, B and C. We use such common services to predict resolutions for (part of) these three unknown relations.

To predict ($D?G$), we select the set of common tasks among them. In this example, this set is {A, B, C, E, F}. Because ($D \leftsquigarrow A$) and ($G \rightsquigarrow A$), we deduce that ($D \leftsquigarrow G$) according to the transitivity rule. Similarly, we deduce all potential relations between D and G using their common services as shown in Table 6. The intersection of these alternatives is ϕ, i.e., there is no common relation among potential relations. Therefore, the relation between D and G in the global profile remains unknown.

We follow the same steps to predict the relation ($D?H$). The set of common tasks is the same. Intersecting all potential relations between D and H gives the new relation ($D \parallel H$). Again, the same set of common tasks is used to reveal the ($D?I$). In this case, the intersection of all potential relations between these tasks gives two alternatives: ($D \parallel I$) and ($D \rightsquigarrow I$). The distance in the new strict order relation is the minimum distance between I and the common tasks. In this case, the distance is 2.

6 Experiments and Evaluation

In this section, we show a set of experiments that we did to evaluate our approach of predicting potential relations among web services using business process knowledge. We use a set of business processes from the SAP reference model [7], because these models represent possibilities to configure SAP R/3 ERP systems. Thus, it is analogous to business process configurations over a service landscape. We use 18 BPs with related missions from the SAP reference model. In particular, they are concerned with purchase order/requisition processing. These processes include 146 tasks in total. On average, each BP has about 8 tasks. Among the 146 tasks, 81 tasks are distinct, i.e., bound (configured) with distinct web services. We did this configurations manually and verified the results manually, as well. We analyzed the labels of the tasks and decided which labels (tasks) that can be bound to the same web service. Additionally, we had to manually restructure the models to have a single start and a single end nodes so that the behavioral profile calculation algorithm can be applied to them. Moreover, we excluded loops to obtain useful behavioral relations among tasks. A loop yields relations among all nodes within that loop concurrent.

The baseline approach is predicting relations among tasks of BPs *without* using their configurations information, i.e., only identical labels of tasks in different BPs are considered similar. Following this approach, we are able to predict resolutions for (54.8%) of all unknown relations in the generated global behavioral profile. The ratio of resolved relations using labels of tasks only depends considerably on the degree of similarity and cohesion among labels. Using the configurations of these BPs where semantically

Table 6. Possible relations between services D & G via common services {A, B, C, E, F}

Common Task	**A**	**B**	**C**	**E**	**F**
Relation with D	D $\leftsquigarrow$ A	D $\leftsquigarrow$ B	D $\parallel$ C	D $\parallel$ E	D $\parallel$ F
Relation with G	G $\rightsquigarrow$ A	G # B	G $\leftsquigarrow$ C	G $\rightsquigarrow$ E	G $\rightsquigarrow$ F
Deduced Relation	D $\leftsquigarrow$ G	D $\leftsquigarrow$ G D # G	D $\rightsquigarrow$ G D $\parallel$ G	D $\leftsquigarrow$ G D $\parallel$ G	D $\leftsquigarrow$ G D $\parallel$ G

Table 7. Revealing hidden relations in the global profile of BP_1 and BP_2

	A	B	C	D	E	F	G	H	I
A	(‖, 0)	(⇜, 1)	(‖, 0) (※, 0)	(⇝, 2)	(⇝, 2)	(⇝, 2)	(⇜, 2)	(⇝, 2)	(⇝, 8)
B	(⇝, 1)	(‖, 0)	(‖, 0) (※, 0)	(⇝, 3)	(⇝, 4)	(⇝, 4)	(# , 0)	(⇝, 4)	(⇝, 10)
C	(‖, 0) (※, 0)	(‖, 0) (※, 0)	(‖, 0)	(‖, 0)	(⇝, 2)	(⇝, 2)	(⇝, 2)	(⇝, 8)	(⇝, 14)
D	(⇜, 2)	(⇜, 3)	(‖, 0)	(‖, 0)	(‖, 0)	(‖, 0)	**(?, 0)**	**(‖, 0)**	**(‖, 0) (⇝, 2)**
E	(⇜, 2)	(⇜, 4)	(⇜, 2)	(‖, 0)	(‖, 0)	(# , 0)	(⇜, 4)	(# , 0)	(⇝, 2)
F	(⇜, 2)	(⇜, 4)	(⇜, 2)	(‖, 0)	(# , 0)	(‖, 0)	(⇜, 4)	(# , 0)	(⇝, 2)
G	(⇝, 2)	(# , 0)	(⇜, 2)	**(?, 0)**	(⇝, 4)	(⇝, 4)	(‖, 0)	(⇝, 4)	(⇝, 10)
H	(⇜, 2)	(⇜, 4)	(⇜, 8)	**(‖, 0)**	(# , 0)	(# , 0)	(⇜, 4)	(‖, 0)	(⇝, 2)
I	(⇜, 8)	(⇜, 10)	(⇜, 14)	**(‖, 0) (⇜, 2)**	(⇜, 2)	(⇜, 2)	(⇜, 10)	(⇜, 2)	(‖, 0)

similar tasks are bound to a single web services, we are able to predict resolutions for around (72%) of all unknown relations among tasks used in our experiments.

We are able to reveal different types of relations among web services. In Table 8, we show the ratio of each type of relations with respect to the total number of relations in source raw profiles, their derived global profile, and after revealing part of the hidden relations in that global profile. Note that percentages in this table are local to each column. The total number of relations is not the same in global profile and revealed global as an unknown relation can be predicted in multiple resolutions.The majority of relations in the revealed global profile are parallel (34%). Additional knowledge about such tasks and their bound web services can be used to resolve such relations in more concrete ones. This further resolution is part of our future work. Conflicting relations appear due to inaccurate configurations of BPs or due to lack of sufficient knowledge about tasks and web services. Unknown relations are still in the global profile even after applying our resolutions approach. Either the used knowledge is not sufficient to reveal such relations or there are no such useful relations. For instance, a music web service and a web service for Gene analysis.

7 Discussion

In this paper we introduced an approach to reveal hidden relations among web services by exploiting process configurations over these services. The relations we address are of four basic types; strict order, inverse order, exclusive, and concurrent. To reveal such relations, we extended the notion of behavioral profiles by adding two additional relations, namely, contradicts and unknown, and distance between activity nodes, within a process, that are bound to the web services under investigation. In practice, several process configurations exist with an organization. Therefore, we had to merge these individual profiles into a single global profile. After that, unknown relations within the

Table 8. Types of relations and their ratios in raw profiles, derived global profile, and resolved profile

Type	Raw Processes	Global Profile	Revealed Global Profile
Strict Order $\rightsquigarrow$	33.25%	3.87%	14.48%
Inverse Order $\leftsquigarrow$	33.25%	3.87%	14.48%
Parallel $\|\|$	9.7%	1.60%	34.03%
Exclusive #	23.8%	3.42%	17.97%
Conflict $\divideontimes$	0%	0.15%	0.12%
Unknown ?	0%	87.10%	18.91%

global profiles were input to our prediction approach in order to reveal possible behavioral relations that might exist among them. To reveal these relations, we use common services between the two services with an unknown relation. We applied our approach to a subset of the SAP reference models and our experiments show that we could reveal about 72% of the unknown relations in the global profile.

Our prediction algorithm is sensitive to the input global profile. If the global profile is derived from widely different processes that have the least in common, the prediction algorithm cannot reveal much. However, one nice result we found about our algorithm is the tendency to find clusters of related web services.

We limited our experiments to process models without loops, because of the sensitivity of the behavioral-relation computation algorithm. For instance, if two tasks A and B are in sequence and that sequence is nested in a loop, the behavioral-relation computation algorithm would indicate that $A \,||\, B$ instead of $A \rightsquigarrow B$. Knowing that $A \rightsquigarrow B$ is more decisive and yields more useful results by the prediction algorithm. One possibility to overcome the loop-sensitiveness limitation is to employ other behavioral-relation computation algorithms, e.g., those of the α-algorithm for process mining [1]. The results of our prediction algorithm is *independent* from the behavioral-relation computation algorithm as all of these algorithms provide the four basic relationships.

There are several directions for future work. First, we plan to tackle the problem of revealing conflicting relations identified during the merge of individual profiles. Second, we intended to extend the discovery beyond binary relations. That is, discover relations among fragments of web services. Also, we aim at applying further experiments on other collections of process configurations. Also, we intend to use more information from the process configurations artifacts in order to enhance our prediction algorithm. In particular, we aim at exploiting data dependencies among web services to help refine our predictions. Data dependencies can also be exploited to reveal conflicts that arise while deriving the global behavioral profile.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
2. AbuJarour, M., Awad, A.: Discovering Linkage Patterns among Web Services using Business Process Knowledge. In: Proceeding of the 8th International Conference on Services Computing. IEEE Computer Society (2011)

3. AbuJarour, M., Naumann, F.: Dynamic Tags For Dynamic Data Web Services. In: Workshop on Enhanced Web Service Technologies. ACM, Ayia Napa (2010)
4. Al-Masri, E., Mahmoud, Q.H.: Investigating Web Services on the World Wide Web. In: Proceeding of the 17th International Conference on World Wide Web, WWW 2008, pp. 795–804. ACM, New York (2008)
5. Basu, S., Casati, F., Daniel, F.: Toward Web Service Dependency Discovery for SOA Management. In: Proceedings of the 2008 IEEE International Conference on Services Computing, vol. 2, pp. 422–429. IEEE Computer Society (2008)
6. Bose, A.: Effective Web Service Discovery using a Combination of a Semantic Model and a Data Mining Technique. Master's thesis, Queensland University of Technology, Queensland, Australia (2008)
7. Curran, T.A., Keller, G., Ladd, A.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model, 1st edn. Prentice Hall (1997)
8. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity Search for Web Services. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, vol. 30, pp. 372–383. VLDB Endowment (2004)
9. Eshuis, R., Grefen, P.: Structural Matching of BPEL Processes. In: Proceedings of the Fifth European Conference on Web Services, pp. 171–180. IEEE Computer Society (2007)
10. Fensel, D., Keller, U., Lausen, H., Polleres, A., Toma, I.: WWW or What is Wrong with Web Service Discovery? In: Proceedings of the W3C Workshop on Frameworks for Semantics in Web Services (2005)
11. Hagemann, S., Letz, C., Vossen, G.: Web Service Discovery - Reality Check 2.0. In: NWESP 2007: Proceedings of the Third International Conference on Next Generation Web Services Practices, pp. 113–118. IEEE Computer Society, Washington, DC, USA (2007)
12. Lecue, F., Leger, A.: Semantic Web Service Composition Based on a Closed World Assumption. In: Proceedings of the European Conference on Web Services, pp. 233–242. IEEE Computer Society (2006)
13. Lin, L., Arpinar, I.B.: Discovery of Semantic Relations Between Web Services. In: Proceedings of the IEEE International Conference on Web Services, pp. 357–364. IEEE Computer Society, Washington, DC, USA (2006)
14. Omer, A.M., Schill, A.: Web Service Composition Using Input/Output Dependency Matrix. In: Proceedings of the 3Rd Workshop on Agent-Oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing, AUPC 2009, pp. 21–26. ACM (2009)
15. Rong, W., Liu, K., Liang, L.: Personalized Web Service Ranking via User Group Combining Association Rule. In: Proceedings of the 2009 IEEE International Conference on Web Services, ICWS 2009, pp. 445–452. IEEE Computer Society (2009)
16. Segev, A.: Circular Context-Based Semantic Matching to Identify Web Service Composition. In: Proceedings of the 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation, CSSSIA 2008, pp. 7:1–7:5. ACM (2008)
17. Sharma, S., Batra, S.: Applying Association Rules For Web Services Categorization. International Journal of Computer and Electrical Engineering 2(3), 465–468 (2010)
18. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the Data-Flow Perspective for Business Process Management. Info. Sys. Research 17(4), 374–391 (2006)
19. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient Computation of Causal Behavioural Profiles Using Structural Decomposition. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 63–83. Springer, Heidelberg (2010)
20. Winkler, M., Springer, T., Trigos, E.D., Schill, A.: Analysing Dependencies in Service Compositions. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 123–133. Springer, Heidelberg (2010)

Towards a Service System Ontology for Service Science

Elisah Lemey and Geert Poels

Center for Service Intelligence,
Faculty of Economics and Business Administration,
Ghent University,
Tweekerkenstraat 2, 9000 Gent, Belgium
{Elisah.Lemey,Geert.Poels}@UGent.be

Abstract. Service Science is a new interdisciplinary approach to the study, design, implementation, and innovation of service systems. However due to the variety in service research, there is no consensus yet about the theoretical foundation of this domain. In this paper we clarify the service systems worldview proposed by Service Science researchers Spohrer and Kwan by investigating its foundational concepts from the perspective of established service theories and frameworks. By mapping the proposed service system concepts on the selected service theories and frameworks, we investigate their theoretical foundations, examine their proposed definitions and possible conflicting interpretations, discover their likely relationships and general structure, and identify a number of issues that need further discussion and elaboration. This analysis is visualised in a multi-view conceptual model (in the form of a UML class diagram) which we regard as a first step towards an explicitly and formally defined service system ontology.

Keywords: Service Science, SSME, service system, conceptual analysis, conceptual modelling, ontology, service-dominant ontology, systems theory.

1 Introduction

Service Science is a new interdisciplinary field that studies the structure and behaviour of service systems. A lot of suggestions have been made about which theories can serve as a basis for Service Science research or which frameworks can be used to conceptualize the object of study of Service Science, but few consensus exists among different authors [1-3]. This lack of agreement may become an obstacle for the further development of the Service Science research field.

Recently, Spohrer and Kwan [4] proposed the service systems worldview as a candidate shared conceptualization for Service Science researchers. Although the concepts of this worldview are not new, their theoretical foundation was not clarified by Spohrer and Kwan.

To foster the discussion of the appropriateness of the proposed service system conceptualisation this paper investigates the proposed concepts from the perspective of established service theories and frameworks from traditional service research areas as service marketing, service management, service operations and service computing. Based on this investigation, we also identify relationships between the concepts

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 250–264, 2011.

proposed for this worldview and visualise the identified structures in a conceptual model (in the form of a UML class diagram).

The basis for the model are the ten foundational concepts of the service systems worldview. The definitions of these concepts are compared with alternative definitions originating in six other service frameworks and theories. We aim to identify which additional concepts from these theories and frameworks can be incorporated in the model to further refine and extend the service systems worldview. By mapping the foundational concepts to the concepts used in traditional service research areas we identify commonalities and differences in interpretation which may help to find a common understanding of the service systems worldview. Also, if we want to create one scientific basis for Service Science research it is crucial that established service frameworks and theories connect to this scientific basis.

Our contribution to the emerging research area of Service Science is twofold. First of all, a UML class diagram for the ten foundational concepts is presented. This diagram is aimed at facilitating the presentation and discussion of the foundational concepts as it also uncovers and shows their relationships. The diagram provides the basis for elaborating a service systems ontology and a meta-model for modelling of service systems. Second, the investigation of the theoretical foundation (if any) and the search for additional concepts which can be marked as foundational, can be seen as a theoretical evaluation of the completeness and relevancy of the set of foundational concepts proposed by Spohrer and Kwan [4]. It provides elements for the further discussion, enhancement, and ultimately (and hopefully) consensual agreement of a service systems conceptualisation for Service Science.

Section 2 presents the service systems worldview as proposed by Spohrer and Kwan [4]. Section 3 gives an overview and motivates our choice of the service theories and frameworks used in the research. Section 4 presents the mapping of the foundational concepts of the service systems worldview to the concepts of the chosen service theories and frameworks. It also explains the development of the conceptual model based on the mapping results and discovery of concept relationships. Section 5 then discusses the main findings of our theoretical investigation of the service systems worldview. Section 6 presents conclusions and future work.

2 Foundational Concepts of the Service Systems Worldview

Spohrer and Kwan [4] use ten foundational concepts to explain the diversity and complexity of service systems: *entity*, *resource*, *access right*, *ecology*, *interaction*, *value proposition based interaction*, *governance mechanism based interaction*, *outcome*, *measure*, and *stakeholder*. To introduce these concepts in this paper we developed a UML class diagram (Figure 1). The use of a visual conceptual model will also facilitate the analysis of the foundational concepts (see section 4).

A service system *entity* is a dynamic configuration of *resources*. There are four types of resources: *physical with rights* (people), *physical without rights* (technology, natural resources), *non-physical with rights* (organisations), and *non-physical without rights* (shared information). In each service system entity there is at least one *focal resource* that holds access rights to the other resources in the configuration. Different types of access rights are *owned outright*, *leased-contracted*, *shared access*, and *privileged access* [5].

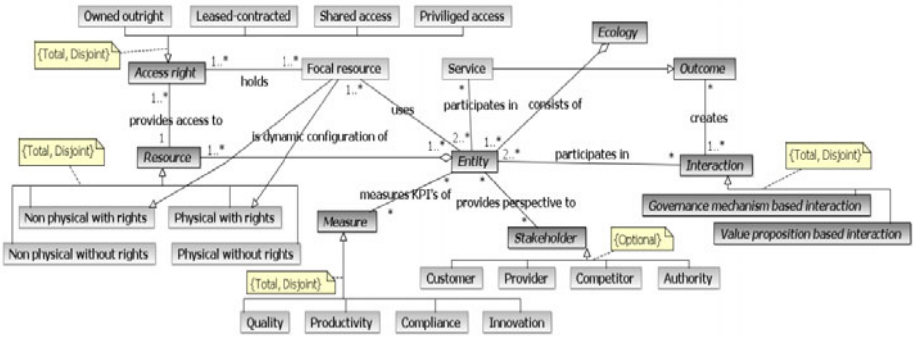


Fig. 1. UML class diagram of service systems worldview

The environment in which service system entities operate is the service system *ecology*. It is the sum of all macro-scale interactions of the population of all service system entities. It is characterized by both the diversity and the amount of types of service system entities [6]. Two main types of service system entities can be distinguished: formal entities that exist within a legal and economic context and are formed by contracts and informal entities that are based on promises and can be situated in a social or political context. Service system entities can thus be people, families, companies, universities, organisations, cities or anything else that can interact with other service system entities [7].

Service system entities participate in *interactions*. It is through these interactions that service system entities improve their state. Thus, the purpose of these interactions is to create value [5]. The service systems worldview advocates mutual value co-creation [8]. Co-creation means that both entities are needed to create value. Mutual means that, service system entities will only create value for another entity if they are getting value out of it themselves. There are two types of interactions. When entities interact through value propositions, this is called a *value proposition based interaction*. A value proposition is a type of shared understanding between service system entities about how interactions between them can lead to mutual value co-creation [5]. Contrary to this type of interaction which is focused on voluntary reciprocal action of individual entities, the second type of interaction, *governance mechanism based interaction,* occurs in the context of collective interest, i.e., when the interaction is regulated by a governing body. Auctions and court cases are typical examples of this second type of interaction [4].

Interactions between service system entities lead to *outcomes*. When two or more service system entities interact, the outcome will be judged by each entity to determine whether value for that entity was created or not. The service systems worldview proposes the Interact-Service-Propose-Agree-Realize (ISPAR) model that models ten possible outcomes for interactions [8]. The ISPAR model indicates that the service systems worldview is not a happy path theory but also takes into account other outcomes that deviate from mutual value co-creation.

The service systems worldview further reckons that the interaction outcomes and their effect on the state of the participating service system entities have to be measured. Therefore, four types of *measures* are defined: *quality*, *productivity*, *legal*

compliance, and *sustainable innovation*. These four measures serve as key performance indicators (KPI's) for a service system entity [5].

Next, all service system entities can view themselves and can be viewed from multiple *stakeholder* perspectives. The four main types of stakeholder perspectives are *customer*, *provider*, *authority*, and *competitor*. These types refer to roles that service system entities play in service system networks that are determined by the patterns of interactions between the participating service system entities.

We added *service* as an additional concept to the model. It seems only logical that a model of service systems includes the notion of service even if service is not one of the ten foundational concepts of the service systems worldview. According to the ISPAR model in the service systems worldview, mutual value co-creation that is the preferred outcome of interactions between service system entities is what is called service. For a service the resources of at least two service system entities are needed [8].

3 Overview of Service Theories and Frameworks

The large variety in service literature provides us with a rich network of conceptual pieces for the constructs described above. However, this variety at the same time brings along a complexity dimension because there is a lack of agreement between the different service theories. Note that in the remainder of this section (and the paper) we will use the term 'theory' to refer also to proposals that are more appropriately called 'framework' as they offer interrelated concepts to define and organise the service domain without purporting to have explanatory or predictive power.

Our choice of theories was mainly guided by previous Service Science research. In a joint white paper of IBM and Cambridge University's Institute for Manufacturing the worldview of Service Dominant Logic (SDL) is indicated as a possible theoretical basis for Service Science [1]. Furthermore, other proponents of Service Science propose the Unified Service Theory (UST), the work system method and the service quality gaps model as interesting theories to draw from for the elaboration of the Service Science discipline [9-11]. As recent Service Science research indicates the need to introduce a system focus in the study of service systems, we also included the system theoretic view of service systems of Mora et al. [12-13]. Finally, we included a service ontology based on the DOLCE upper-level ontology [14]. Although this ontological theory may not be as well-known in service research as the other theories, we believe that the insights of this ontological analysis are valuable. As it shows how a service ontology can be defined as a specialisation of a philosophical ontology describing the world in general, it can serve as an example for our own (future) research. We will give a description of each theory as an introduction to the reader.

First, *SDL* advocates that service is the fundamental unit of exchange. A service is defined as "the application of specialized competences through deeds, processes and performances for the benefit of another entity or the entity itself". This implies that SDL does not focus on the products that are produced but on the value creating processes that accompany the consumption of the product and that deliver the actual value to the customer [15-17].

Second, *UST* focuses on the service process and states that every provider process in which an individual customer input can be identified is a service. Customers thus

act as suppliers in the service process. They can offer themselves, their property or information as input. The unique contribution of this theory is in defining the concept of customer input [18].

Third, *work system method* and related concepts form the basis of a business-oriented system analysis and design tool. The work system framework uses nine basic elements to provide a system view of the organisation. The service value chain framework elaborates the work system framework with service-oriented insights. It presents a two-sided view of the service process as the service is coproduced by customer and provider. The work system life cycle model evaluates the change of work systems over time [2, 19].

Fourth, the *service quality gaps model* is designed to measure the quality of services. The starting point of the model is the assumption that there exists a gap between the quality perception of a company's management team and that of the customers. The model identifies four gaps on the side of the provider which combine into a fifth gap on the side of the customer: a gap between the expected service quality and the perceived service quality [20].

Fifth, the *system theoretic approach of Mora et al*. shows a systems view in which service systems are part of larger supra-systems and are composed themselves of service sub-systems [13]. There are two types of service sub-systems defined. The service facilitator represents the original service provider and the service appraiser represents the initial user's system. The supra-system further contains all environmental elements such as competitors, customers, regulators, suppliers or partners. An important feature of the model is that the supra-system is influenced by the value outcomes of its service (sub-)systems.

Sixth, the *service ontology based on the DOLCE upper-level ontology of Ferrario and Guarino* provides a general ontological foundation for service systems [14]. A service is conceived as a complex event with five main parts: service commitment, service presentation, service acquisition, service process and service value exchange. A key concept in this service ontology is the commitment of an agent to guarantee the execution of a service at a certain place and time.

4 Analysis

For the analysis of the foundational concepts in the light of the above described theories, we start from Figure 1. The concepts of this model will be mapped subsequently on each service theory identified in the previous section. We have developed UML class diagrams for representing the concepts (and their relationships) underlying each theory to facilitate the mapping process. These diagrams can be interpreted as different views that overlay the conceptual model represented by the UML class diagram in Figure 1, where each view defines, refines and/or extends the foundational concepts from the perspective of the concerned theory. Although the foundational concepts can be compared to any of these theories in arbitrary order, we present our analysis results in an incremental manner focusing on the most remarkable interpretations and additions brought about by the different theories.

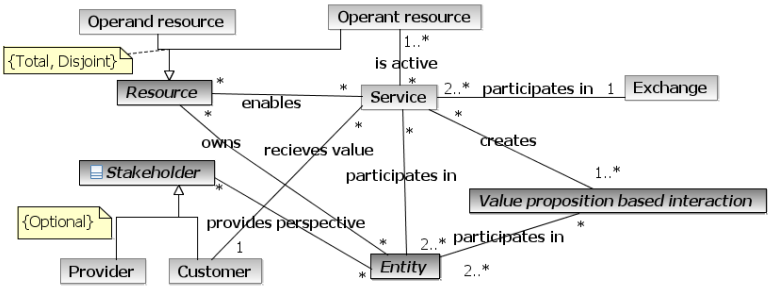


Fig. 2. UML class diagram of Service Dominant Logic

First of all, the service systems worldview embraces the ideas of SDL (Figure 2) which is proposed as its philosophical foundation [1]. However, not all foundational concepts can be grounded in SDL as this theory has no definition of the concepts ecology, governance mechanism based interaction, measure, and access right. Moreover, SDL is a happy path theory and therefore the concept outcome is narrowed down to value co-creation. As value co-creation is the only possible service outcome, it can be equated with service in Figure 2. SDL also indicates that there is only one beneficiary of the service, i.e., the customer [17]. Both SDL and service systems worldview state that a service implies the participation of at least two entities but according to SDL this doesn't imply mutual value creation. As explained, the value is co-created but only for the customer and thus economic exchange is needed for mutuality [15]. Therefore exchange is shown as a separate concept in the UML class diagram in Figure 2 whereas it was embedded in the service concept in Figure 1.The specification of the concept resource is also different. SDL differentiates between operand and operant resources. Operand resources such as commodities, buildings or tools are passive resources on which an operation or activity is performed to produce an effect. Operant resources such as competences or knowledge are employed to act upon operand resources and other operant resources [15]. SDL implies that at least one operant resource is active in the service.

Second, UST takes a service operations perspective in which there is a strong focus on the concepts related to the service process (Figure 3). A service is perceived as a

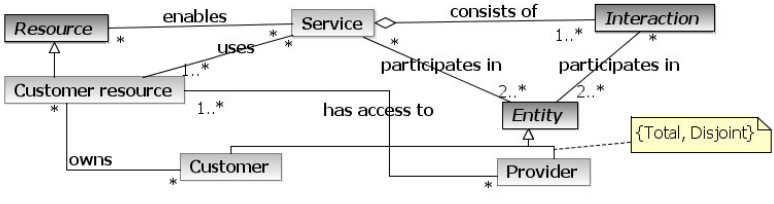


Fig. 3. UML class diagram of the Unified Service Theory

process that consists of (at least one) interactions between customer and provider. Furthermore, the theory introduces the concept customer resources which include customer-self inputs, tangible belongings and customer-provided information [18]. According to UST a service is a production process in which the customer provides at least one resource that is an essential input for the production of the service. This implies an extra constraint on the relationship between customer resource and provider. The provider has to have access to the customer resource(s) that are used as production input. This customer involvement also manifests itself in the notion of co-production. Co-production implies not only participation of two service system entities (i.e., co-creation) but also the active involvement of the customer in the production process by contributing at least one customer resource.

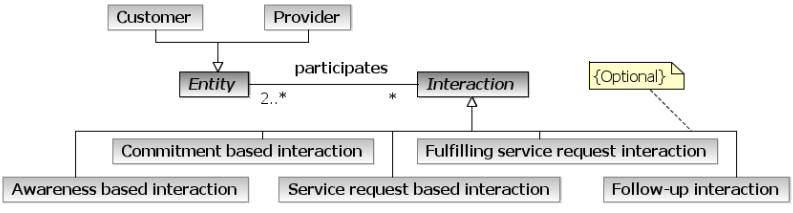


Fig. 4. UML class diagram of the work system method

Third, Alter's work system method also implies a process focus. The added value of this theory is in the service value chain. It shows the different steps that should be executed in a service process [2]. Alter shows the different kinds of interactions that emanate from a value proposition (Figure 4). First, customer-provider contact is based on awareness. The provider should create awareness among customers about the existence of the service. The customer should become aware of a need that has to be filled. Next, provider and customer will engage in a negotiation about commitment to the service. The ISPAR model of the service systems worldview also recognizes these two types of interaction (i.e., proposal and agreement) which show how the service process is initiated, but does not further distinguish between different types of interaction that occur when the rest of the service process is executed (i.e., realisation). The service value chain model further defines these interactions or 'service encounters'. The customer makes a service request which is handled and fulfilled by the provider and the customer participates in this fulfilment. Finally, both customer and provider follow up the handling of the service.

Fourth, the service quality gaps model was designed as an instrument to measure service quality. The contribution of the model is in the definition of five determinants of quality that serve as measures: reliability, assurance, tangibles, empathy, and responsiveness (Figure 5). It should be noted that the customer is the central stakeholder in the service quality gaps model. Therefore only the expected and the perceived quality from the viewpoint of the customer are taken into account. This is similar to the service systems worldview where the quality measure should be evaluated by service system entities in the role of customer [5]. A difference with quality as a measure in the service systems worldview is that quality is considered in relationship to services (as manifested by interactions between customer and provider) whereas in the service systems worldview it is a KPI of the service system entity in the role of provider.

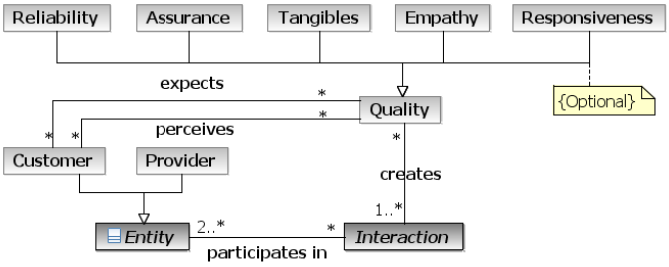


Fig. 5. UML class diagram of service quality gaps model

Fifth, the systems theoretic model of Mora et. al. [13] defines a supra-system which corresponds with the ecology foundational concept and a sub-system which corresponds with the service system entity foundational concept (Figure 6). The supra-system consists of all stakeholders such as competitors, customers, authorities, providers or partners. An original insight from this model is that the supra-system is also influenced by the outcome of the interactions between the appraiser and facilitator subsystems. This outcome is defined as a change in the service properties in both the appraiser and facilitator subsystems. The systems theoretic model also accounts for service failure. The vision of Mora et. al. on service is very similar to that of the service systems worldview. Especially the explicit recognition of the service system ecology as the supra-system in which interactions between appraiser and facilitator sub-systems take place, shows the significant resemblance between the two models. This recognition also indicates an outward focus of the model. When a service is performed, it brings along changes not only for the participants of that service but also for the surrounding system.

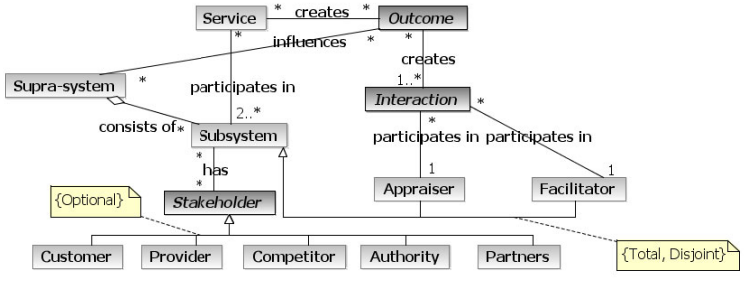


Fig. 6. UML class diagram of the systems theoretic model of Mora et al. [13]

Sixth, the service ontology of Ferrario and Guarino [14] also elaborates the concept of interaction (Figure 7). The authors strongly focus on the concept of commitment which they consider as the core of a service. However, commitment here has a different meaning than commitment defined by Alter. Here commitment implies more than just offering a value proposition. A value proposition is usually the result of

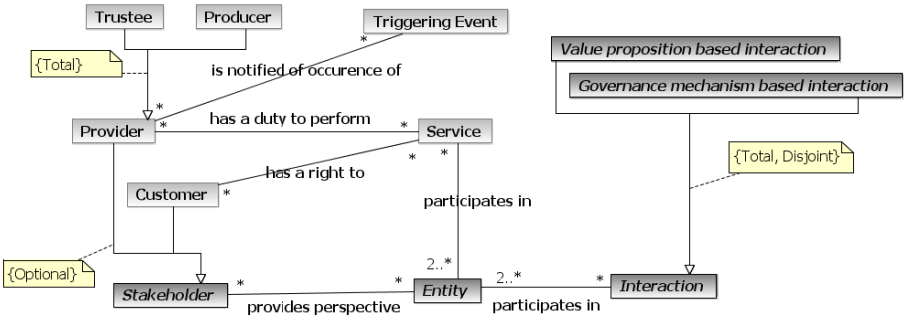


Fig. 7. UML class diagram of the service ontology of Ferrario and Guarino [14]

negotiations between provider and customer. Therefore the customer has already decided he wants a certain service. Service commitment according to Ferrario and Guarino means the willingness to perform a service on the side of the service provider without needing the involvement of the customer. This implies that service can exist even before the occurrence of interactions between provider and customer. Nevertheless, commitment is not sufficient to initiate the actual service execution as it only indicates willingness on the side of the provider. Hence, a triggering event is included. Provider entities should be notified of the occurrence of this event in order to know when the service should be executed. To the specification of the provider, the concept of trustee is added. The authors indicate that the entity that makes a commitment is not always the producer of the service. This distinction brings a sense of reality into the service ontology as real life services are often realized with more than two entities participating. Lastly, the service ontology provides us with a legal perspective on the service. It defines the right to a service which can be considered as an object and is, in contrast to the service, transferable. When a customer has a right to a service, the provider has a duty to perform this service. This duty to perform a service can also be transferred. The essence of the service process is not the service itself but the normative position of the participants in the service process.

5 Discussion

A core set of three foundational concepts is supported by all service theories used in the analysis: service system entity, interaction, and stakeholder perspective (or role played by the entity in the interaction). Although not all analysed service theories explicitly mentioned the concept of stakeholder, all theories support the view that it is essential for the service activity to have interactions between provider and customer entities. This implies that the ontological classification of service as an activity rather than an object is a feature that unites the service theories considered and the service systems worldview that is proposed for Service Science. The service ontology of Ferrario and Guarino provides us with a specification of the provider concept and distinguishes between service trustee and producer, where the former may delegate the service activity to the latter. This is an interesting perspective which does not

contradict the service systems worldview as Spohrer and Kwan do not claim to have enumerated all possible stakeholder perspectives. The recognition of this distinction may however account for phenomena such as service outsourcing or subcontracting with an associated legal perspective. This may prove useful for the study of service systems.

The other foundational concepts are supported by only some of the service theories. Given that each service theory considers a particular scope within the service domain, this is not a problem per se as long as theories do not explicitly reject the existence of the concepts falling outside their scope. For instance, SDL only recognizes value proposition based interactions, but does not refute governance mechanism based interactions. The interactions described in the service value chain framework of the work system method are also value proposition based and implicitly this type of interaction is assumed by the service quality gaps model, so there seems to be sufficient support for this type of interaction. The service commitment in the service ontology of Ferrario and Guarino can be a value proposition based interaction or a governance mechanism based interaction, e.g., a newly founded state university commits to the government of the state to provide education to its citizens once it becomes operational.

Another foundational concept for which support can be found in several of the discussed theories is resource. The service systems worldview defines service system entities as configurations of resources. Service entails interactions between a provider and customer entity, implying that resources of both entities are involved (e.g., consumed, used, applied, employed,..). For the provider resources in the service this implies that at least one focal resource of the provider entity is involved as this focal resource has access rights to other provider resources that might be needed. The service systems worldview distinction of resources 'with rights' versus 'without rights' is largely similar to the distinction 'operant' versus 'operand' in SDL so the involvement of a focal, hence operant resource of the provider in the service thus conforms to SDL. What is less clear in the service systems worldview is the involvement of customer resources. Here, both SDL and UST can shed some light, though in different and possibly contradicting ways. UST requires the involvement of at least one customer resource, which can be physical or non-physical, operand (without rights) or operant (with rights), as an input to the production of the service. SDL does also require the involvement of at least one customer resource, though not necessarily as an input to the service production. This difference, which can also be captured by the distinction between co-production (UST) and co-creation of value (SDL), is important and leads to a different view of service.

UST recognizes the existence of other processes than service processes, e.g., production processes aimed at mass production of commodities, whereas SDL rejects any other economic activity than service activity. Even the buying of mass produced commodities without providing any individual input to the production process is considered as service exchange because the value that the customer gets results from using these commodities (i.e., integrating it with other customer resources in SDL speak). This difference has important consequences for the service systems worldview. As SDL is proposed as the philosophical foundation of Service Science [1], the service systems worldview tends towards the co-creation view. If SDL is followed then the logical consequence is that all economic activity must be service.

In this case a further specification of the nature of the involvement of customer resources and the implications for value is desired. If, however, value co-creation is interpreted as co-production as in UST then not all economic activity would qualify as service activity. If this would be agreed upon, the concept of value co-creation needs to be redefined. Even in that case, the nature of the involvement of customer resources needs to be specified further.

The foundational concept of access right seems to be a derivative concept (hence we can question its qualification as foundational), but is certainly useful for further describing what is meant by an entity as a configuration of resources. If value co-creation would be redefined using UST (so more as co-production), then the access right concept can be further founded on this theory, specifically with respect to the provider entity having access to customer resources that are input to the service process.

The concept of ecology can be founded on the notion of supra-system as in the system theoretic model of Mora et al. This concept is not explicitly present in the other theories discussed, although the service ontology of Ferrario and Guarino recognizes the existence of a broader societal and legal context in which service takes place (hence the presence of service commitment as a governance mechanism based interaction). Interesting in the system theoretic model is the link between interaction outcomes and the supra-system showing that the favourable or unfavourable outcomes of service interactions affect and are affected by the ecology in which the participating service system entities reside. The explicit recognition of such a relationship may be useful for the study of service systems.

The favourable outcome according to the service systems worldview is mutual value co-creation, which is also the view of SDL. However, in SDL mutual value co-creation is the only outcome explicitly recognized. The system theoretic model recognizes that there might be other, unfavourable outcomes. The service quality gaps model does also given that a gap can exist between the expected and perceived outcome, which should be quality according to this model.

This brings us to the last foundational concept to discuss, measure. The measure concept and its specialisation into four kinds of performance indicators for a service system entity is not directly supported by the service theories used in the analysis. The service quality gaps model focuses on quality measures where quality is one of these four dimensions. However, it considers quality in relation to individual or sets of service interactions and not to the performance of the service system entity as a whole. Of course, quality assessments of service interactions do provide an indication of the quality of the service provider itself. For the other three dimensions (productivity, compliance, innovation) a theoretical foundation must be found elsewhere.

An interesting difference and possible point of discussion for the service systems worldview is what is not explicitly defined in any of the different model views and that is the concept of service system itself. One point of view is that the service system is described by the entirety of the concepts and their relationships. But even then, important differences between the service theories pop up. Table 1 tries to answer the question 'what is the service system?' for the discussed theories and the proposed service systems worldview by looking at different dichotomies that were derived from the analysis in the previous section. The table shows that the theories roughly fall apart in two categories: those with an outward focus implying that services are positioned within and have effect on a broader context, which is considered as the service

system, and those with an inward focus implying that services take place between and have an effect on their participants, which are considered as the service systems. The inward focus category is made up of SDL, which is clearly its prime representant, and further also the work system and service quality gaps model. The UST leans towards the inward focus, but takes a somewhat special position as it strongly emphasizes the service process happening within the service system that is in the provider role [21]. To the outward focus category belong the system theoretic model and to a lesser extent the service ontology. Although conceptual research in Service Science that preceded the proposal of the service systems worldview can clearly be characterized by an inward focus because of its embracing of SDL as philosophical foundation, our analysis shows that the set of foundational concepts as proposed by Spohrer and Kwan clearly tends towards an outward focus as it fits well with the system theoretic model of Mora et al. This result is an interesting point of discussion as it might imply that the service system conceptualisation put forward is focusing now on its systems foundation after having developed its SDL-based service foundation.

Finally, we already remarked that service itself is not defined as a foundational concept in the service systems worldview; we added it ourselves as an eleventh, potentially foundational concept to the model represented by the class diagram in Figure 1. It is remarkable that few service theories discussed provide an explicit definition of service; SDL and the service ontology by Ferrario and Guarino being notable exceptions. Nevertheless, the notion of service as viewed upon by a certain theory can mostly be derived from the definitions and relationships of the concepts that are explicitly defined. Based on our analysis we can distil three main perspectives: process-oriented, outcome-oriented and commitment-oriented which are presented in figure 8.

Four of the analyzed theories have a process orientation. First, the service value chain framework of the work system method distinguishes and sequences five different provider-customer types of interaction (see Figure 4). Also the service ontology of Ferrario and Guarino distinguishes between different phases that can be related to a service as a process view. Next, UST is clearly a process-oriented theory but focuses on customer inputs rather than on interactions between customer and provider. Finally, the ISPAR model related to the service systems worldview recognizes three types of service interaction i.e., proposal, agreement, and realisation.

Table 1. What is a service system?

	Service system worldview	SDL	UST	work system method	service quality gaps model	system theoretic model	service ontology
Closer to the notion of ecology (or a subset of it as the notion of ecology can be defined recursively) or closer to the notion of service system entity?							
Ecology						⇐⇒	
Entity		⇒⇐	⇒⇐	⇒⇐	⇒⇐		⇒⇐
Value creation or production interactions occur within the service system or between the service systems?							
Within	⇐⇒		⇐⇒			⇐⇒	
Between		⇒⇐		⇒⇐	⇒⇐		⇒⇐
Mutuality occurs within a service system or between service systems (implying economic exchange between service systems)?							
Within	⇐⇒					⇐⇒	
Between		⇒⇐					
Also governance mechanism based interactions or only value proposition based interactions?							
Governance mechanism	⇐⇒						⇐⇒
Only value based		⇒⇐	⇒⇐	⇒⇐	⇒⇐		
Many kinds of stakeholder perspectives or only explicit recognition of provider-customer roles?							
Many kinds	⇐⇒					⇐⇒	⇐⇒
Only provider-customer		⇒⇐	⇒⇐	⇒⇐	⇒⇐		
with ⇐⇒ representing an outward focus and ⇒⇐ representing an inward focus							

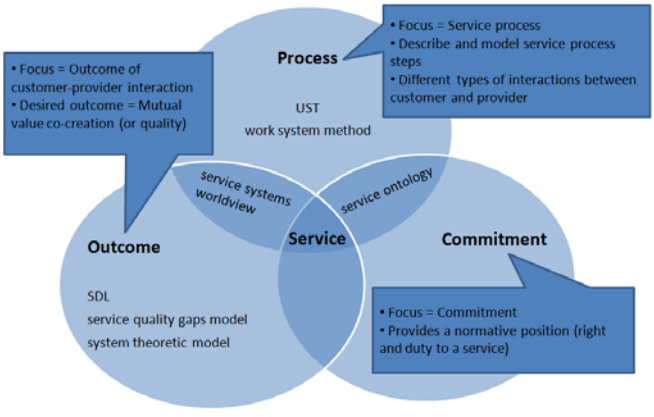


Fig. 8. Three perspectives on service

The service systems worldview, SDL, the service quality gaps model and the system theoretic model of Mora et al. are outcome-oriented. For SDL this outcome is mutual value co-creation where the mutuality is established through economic exchange (i.e., service-for-service) and the co-creation is realized through resource integration by the customer. For the service quality gaps model the outcome is the quality of the service as perceived by the customer. For the system theoretic model the desired outcome is the improvement of the state of the appraiser and facilitator subsystems (i.e., the consumer and provider entities), which affects the state of the service system and its supra-system (i.e., the ecology). The service systems worldview of Spohrer and Kwan shares the desired outcome of service as mutual value co-creation with SDL, though mutuality may here reside within the service (as in the system theoretic model of Mora et al.) rather than in the economic exchange of services. With the system theoretic model it shares the goal of service interactions as expressed by improvements in the state of the participating service system entities. Also the observation that the desired outcomes are not always achieved is shared with the systems theoretic model.

The commitment-oriented perspective refers to the service ontology of Ferrario and Guarino, which distinguishes between the service process (as discussed above under process-oriented) and the service itself. The service ontology takes a unique position by defining service as a commitment between a provider and a customer or a governance body that acts in the interests of (future) customers. The service as commitment view allows service to exist even without interactions taking place between the provider and the customer. The commitment is an interaction, but does not necessarily involve the customer and this position was not found in the other service theories discussed, neither is it part of the service systems worldview.

6 Conclusion and Future Work

In this paper we investigated whether the service systems worldview of Spohrer and Kwan can be founded on established service theories and frameworks. Our research points out that more or less all of the foundational concepts and their proposed specialisations are covered by one, many or in some cases even all reviewed service theories or frameworks. We identified a couple of issues that need further discussion and elaboration, e.g., because of conflicting views when mapping foundational concepts to the concepts of different service theories. Overall, however, our analysis shows that there is evidence of theoretical support for the proposed service systems worldview.

An interesting finding is that, although SDL was initially proposed as the philosophical foundation for the service systems worldview, our analysis indicates that the service system conceptualisation put forward by Spohrer and Kwan is developing beyond SDL. The resemblance with the system theoretic approach of Mora et al. shows a shift towards systems thinking which should be further explored in the future.

Future research may develop in two directions. First, the UML class diagrams developed in this paper can be used as a basis for the further formalisation of the service systems worldview into a service systems ontology. The availability of a consensually agreed ontology could take Service Science a big step forwards as the integrative nature of the research intended by this interdisciplinary field requires a common ground to succeed. Second, our analysis shows that the process orientation of the service systems worldview needs further development. More precisely, more elaborate service process models than ISPAR could be developed to account for the more detailed service interaction typologies proposed by some of the frameworks discussed in the paper.

References

1. IfM, IBM: Succeeding through service innovation: a service perspective for education, research, business and government. University of Cambridge Institute for Manufacturing, Cambridge (2008)
2. Alter, S.: Service system fundamentals: Work system, value chain, and life cycle. IBM Systems Journal 47, 71–85 (2010)
3. Vargo, S.L., Maglio, P.P., Akaka, M.A.: On value and value co-creation: A service systems and service logic perspective. European Management Journal 26, 45–152 (2008)
4. Spohrer, J., Kwan, S.K.: Service Science, Management, Engineering, and Design (SSMED): An Emerging Discipline-Outline & References. International Journal of Information Systems in the Service Sector (IJISSS) 1, 1–31 (2009)
5. Spohrer, J., Maglio, P.P.: Service Science: Towards a Smarter Planet. In: Salvendy, G., Karwowski, W. (eds.) Introduction to Service Engineering. John Wiley & Sons (2010)
6. Spohrer, J., Anderson, L., Pass, N., Ager, T.: Service science and service-dominant logic. In: Otago Forum 2: Academis papers, pp. 4–18 (2008)
7. Spohrer, J., Golinelli, G.M., Piciocchi, P., Bassano, C.: An Integrated SS-VSA Analysis of Changing Job Roles. Service Science 2(1/2), 1–20 (2010)
8. Maglio, P.P., Vargo, S.L., Caswell, N., Spohrer, J.: The service system is the basic abstraction of service science. Information Systems and E-Business Management 7, 395–406 (2009)

9. Chesbrough, H., Spohrer, J.: A research manifesto for services science. Communications of the ACM 49, 35–40 (2006)
10. Maglio, P.P., Srinivasan, S., Kreulen, J.T., Spohrer, J.: Service systems, service scientists, SSME, and innovation. Communications of the ACM 49, 81–85 (2006)
11. Spohrer, J., Maglio, P.P., Bailey, J., Gruhl, D.: Steps toward a science of service systems. Computer 40, 71–77 (2007)
12. Barile, S., Spohrer, J., Polese, F.: System Thinking for Service Research Advances. Service Science 2 (2010)
13. Mora, M., Rory, S.R.M., Gelman, O.C., Toward, O.: an Integrated Conceptualization of the service and Service system Concepts: A systems approach. International Journal of Information Systems in the Service Sector (IJISSS) 1, 36–57 (2009)
14. Ferrario, R., Guarino, N.: Towards an ontological foundation for services science. Future Internet–FIS 2008, 152–169 (2009)
15. Vargo, S., Lusch, R.: Evolving to a new dominant logic for marketing. Journal of Marketing 68, 1–17 (2004)
16. Vargo, S., Lusch, R.: From goods to service (s): Divergences and convergences of logics. Industrial Marketing Management 37, 254–259 (2008)
17. Lusch, R., Vargo, S., Wessels, G.: Toward a conceptual foundation for service science: Contributions from service-dominant logic. IBM Systems Journal 47, 5–14 (2010)
18. Sampson, S., Froehle, C.: Foundations and implications of a proposed unified services theory. Production and Operations Management 15, 329 (2006)
19. Alter, S.: The work system method: connecting people, processes, and IT for business results. Work System Press (2006)
20. Parasuraman, A., Zeithaml, V., Berry, L.: A conceptual model of service quality and its implications for future research. The Journal of Marketing 49, 41–50 (1985)
21. Wild, P.J.: A systemic framework for supporting cross-disciplinary efforts in services research. CIRP Journal of Manufacturing Science and Technology (2010)

Support for the Business Motivation Model in the WS-Policy4MASC Language and MiniZnMASC Middleware

Qinghua Lu[1,2], Vladimir Tosic[1,2], and Paul L. Bannerman[1,2]

[1] NICTA, Australian Technology Park, Sydney, NSW, Australia
[2] University of New South Wales, Sydney, NSW, Australia
{Qinghua.Lu,Vladimir.Tosic,Paul.Bannerman}@nicta.com.au

Abstract. The WS-Policy4MASC language and MiniZnMASC middleware for policy-driven management of service-oriented systems enable making IT system management decisions that maximize diverse business value metrics (e.g., profit, customer satisfaction). However, their past support for alignment with high-level business considerations was weak. Therefore, we introduce a new extension of WS-Policy4MASC that specifies the key concepts from the Business Motivation Model (BMM) industrial standard for modeling business intent. These concepts include hierarchies of ends (e.g., goals) and means (e.g., strategies). We also present and illustrate new decision making algorithms that leverage information in the extended WS-Policy4MASC to align run-time IT system management decisions with business considerations.

Keywords: Business-driven IT management, business motivation model, dynamic adaptation, policy-driven management, self-management, service-oriented computing, Web service management.

1 Introduction

Information technology (IT) systems are rarely ends in themselves. They usually execute in support of actions to fulfill operational and strategic objectives of an organization. Therefore, organizational performance includes a measure of the effectiveness of IT systems in meeting business drivers and expectations. This paper illustrates how the high-level business constructs in the Object Management Group's (OMG) Business Motivation Model (BMM) can guide the execution of service-oriented systems, through extensions to the WS-Policy4MASC language and MiniZnMASC middleware for policy-driven self-management. While the implementation of the presented system is for management of service-oriented systems and business processes, the underlying conceptual solutions can also be generalized to policy-driven self-management of other IT systems.

Due to the high complexity of management tasks and the cost of experienced human system administrators, it is more efficient and cost-effective for IT systems to be self-managing, directed by high-level policies at run-time. Self-management has been a research goal for several decades, but was made prominent by the vision of

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 265–279, 2011.

autonomic computing [1]. Additionally, while business users are typically interested in maximizing business value, prior IT system management solutions have mostly focused on optimizing technical quality of service (QoS) metrics and not directly on maximizing business value. The goal of business-driven IT management research (BDIM) is to determine mappings between technical and business metrics and leverage these mappings to make run-time IT system management decisions that maximize business value metrics [2]. Autonomic BDIM is the intersection area between autonomic computing and BDIM, where processing of business value metrics is added to the decision making components of autonomic computing. There are many open research challenges in this intersection area [2].

Service-oriented computing has become the dominant way of building distributed computing systems. While business-driven management of service-oriented systems is mentioned in prior research [3-5], there are still many open issues. One of the limitations of the past BDIM research is that most works have focused on maximizing profit. However, human business managers can have diverse business objectives. Maximizing short-term profit may not always be the only or best approach to achieve long-term or high-level business goals [4-5]. Business motivation (such as goals, strategies) is a major differentiator of companies in a market, so it is an ideal mechanism to incorporate in self-managing BDIM solutions to direct decision making in controlling and adapting IT systems in response to run-time changes.

BMM is an OMG standard for specification of high-level business motivation and intent as input into design, development and execution of IT systems [6]. Our WS-Policy4MASC language [4] for specification of policies for management of IT (particularly service-oriented) systems provides unique support for implementing autonomic BDIM solutions. Our algorithms using WS-Policy4MASC information, implemented in the MiniZnMASC middleware [7], make decisions for dynamic adaptation of service-oriented systems that maximize diverse financial and non-financial business value metrics. However, the alignment with high-level business value metrics was weak in our previous work. Therefore, we now present extensions to WS-Policy4MASC incorporating key BMM constructs and new run-time self-management algorithms to leverage these additional metrics.

In the next section, we present background information on WS-Policy4MASC, MiniZnMASC and BMM, and overview other major related work. The main section of the paper details our WS-Policy4MASC extensions with the key BMM constructs and our new BDIM algorithms that use these metrics. In the final section, we summarize conclusions and future work.

2 Background and Related Work

2.1 WS-Policy4MASC and MiniZnMASC

WS-Policy4MASC [4], our extension of the WS-Policy industry standard, is a policy language that can describe various adaptations and all information necessary for decision making. WSPolicy4MASC defines five types of WS-Policy policy assertions: 1) goal policy assertions (GPAs) prescribe conditions to be met; 2) action policy assertions (APAs) list adaptation actions; 3) utility policy assertions (UPAs)

contain business metrics for particular situations; 4) probability policy assertions (PPAs) specify probabilities of occurrence, and; 5) meta-policy assertions (MPAs) describe which values are important for adaptation decisions.

The specification of diverse business value metrics in UPAs and MPA specification of strategies for choosing among alternative adaptation actions are the main original contributions of WS-Policy4MASC and differentiators from the other WS-Policy extensions. Each adaptation approach is modeled as an APA. WS-Policy4MASC enables specification of both financial and non-financial business value metrics (BVMs) in UPAs. A number of UPAs can correspond to consequences of executing a particular APA or meeting a particular GPA. WS-Policy4MASC enables dealing with uncertainty through PPAs, e.g. by assigning probabilities that different estimates of the same BVM will be correct. It enables specification of business strategies in MPAs, as a means of deciding which among alternative adaptation approaches to take in policy conflict situations when several APAs could be applied, but only one can be chosen. In addition to policy assertions, WS-Policy4MASC also specifies details necessary for run-time management in auxiliary constructs: ontological meaning, monitored QoS metrics, monitored context properties, states, state transitions, events, schedules, applicability scopes, and various types of expression.

Our MiniZnMASC middleware [7] is a comprehensive framework for autonomic management of service-oriented systems and business processes. It implements novel decision-making algorithms that, at runtime, concurrently provide adaptation decisions, depending on different business strategies and operational circumstances, in a way that achieves maximum overall business value while satisfying all given constraints. These decision-making algorithms use information specified in WS-Policy4MASC policy assertions. They are triggered by monitored events, such as not meeting a GPA. When adaptation decisions are needed at the same time for a number of business process instances, MiniZnMASC uses the constraint programming language MiniZinc [8] to make such decisions. Although our past MiniZnMASC publications focus on autonomic BDIM support, this middleware can be used also for traditional decision-making that maximizes technical metrics. This is because WS-Policy4MASC can describe all information necessary for adaptation decision-making. Our evaluation using several prototype implementations showed that the proposed MiniZnMASC architecture and decision-making algorithms are feasible and easy to modify. Furthermore, our performance and scalability tests showed that MiniZnMASC does not introduce unforeseen performance or scalability problems.

2.2 Business Motivation Model (BMM)

BMM [6] comprises a set of abstractions that define elements of business plans integrated with high-level processes to accommodate business change. It provides the business motivational intelligence to frame operational system services within a context of ongoing business change. As depicted in Figure 1, BMM integrates four primary motivational elements in the model: end, means, influencers and assessment. *End* defines the organization's aspirations – what it wants to be or become. *Means* specify the actions the organization will undertake to achieve the desired ends. *Influencers* are internal or external causes of change that may influence the

organization's business motivation. An a*ssessment* is a judgment about the impact of an influencer on the organization's current end and/or means. Assessments may employ existing analysis techniques such as SWOT (strength, weakness, opportunity, threat) and consider *potential impacts* of influencers in terms of risks and *potential rewards*. Assessment decisions may result in changes to the current end and/or means.

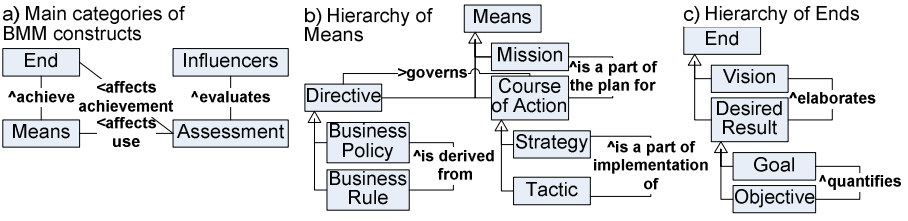


Fig. 1. Main concepts and interrelationships in BMM (based on [6])

The end and means comprise component elements. An end comprises a vision and desired result; a desired result comprises a goal and objective. *Vision* is "an overall image" of the aspiration (that is, it may not be fully or explicitly defined). Goals and objectives are more specific. A *Goal* is a qualitatively defined desired result, while an *Objective* is a quantitatively defined discrete step towards achieving a goal. Objectives provide metrics for measuring progress towards the stated goal. Means comprise mission, course of action and directive. *Mission* describes the broad ongoing operational activity of the organization. *Course of Action* define the actions the organization will undertake in terms of *Strategies* (broadly scoped actions) and *Tactics* (narrowly scoped actions). The BMM hierarchy of means also includes *Directives* and the overall model includes *Influencers* and *Assessment. Directives* specify the business policies and rules that frame courses of action. *Business Policies* define what can and cannot be done within means in a general sense, but are not directly actionable. By contrast, *Business Rules* provide specific actionable guidance to implement or fulfill business policies but are defined externally to BMM. Similarly, organization unit and business process have roles in BMM but are defined externally, in other OMG standards. *Influencers* are anything that can impact employment of means or achievement of ends; while an *Assessment* is a judgment about the impact of an influence.

The major value of BMM is in providing business plan-based structured 'intelligence' to guide the implementation and execution of software-based business services. It provides an abstraction of why businesses pursue particular ends through particular means and includes a basic assessment mechanism to consider emergent change impacts over time. However, BMM also has limitations, derived from simplifying assumptions that limit its application in practice. First, it assumes a predominantly stable operating environment. This may not be possible in organizations in highly dynamic and volatile environments. Second, it assumes that strategies and tactics are and can be made explicit. This is usually not possible in the incremental/emergent approach to strategy. Third, while the standard acknowledges that BMM may be applied at different organizational levels, it does not explicitly accommodate multi-unit modeling. Finally, several elements in the model, such as

vision, do not lend themselves to explicit definition or necessarily translate into tangible entities. The presence of intangibles can be a significant barrier in the implementation of autonomously managed systems. Notwithstanding these limitations, BMM does provide a standard definition of high-level business value constructs that can be mapped as drivers for IT systems management.

2.3 Other Related Work

The need to frame and integrate service-oriented systems with business level drivers, such as goals and strategies, is increasingly recognized in the literature. A common rationale for this is alignment of the organization's IT-enabled operations with its strategic motivations and directions. However, design-time integration is studied much more often than execution time integration.

A potential alternative implementation model is the balanced scorecard (BSC). Conceptually, the balanced scorecard has been influential in motivating alignment of operations with organizational strategies. Proposed as a mechanism to map strategy to implementation, a balanced scorecard can represent the cause and effect linkages through which specific improvement objectives can be realized [9]. Strategic maps reflect how business strategies can be achieved through supporting financial, customer, internal business process, and learning and growth implementation perspectives. While the balanced scorecard has been explicitly applied to IT (the *IT balanced scorecard*) to support alignment through IT governance [10], it has not yet been fully implemented at the systems level. A key difficulty in implementing the balanced scorecard in self-managed service-oriented systems is its high dependency on intangibles [11]. Consequently, its application has been primarily a manual process. Nevertheless, the balanced scorecard concepts inspire and influence researchers to align service-level operations to business strategies.

In particular, [12] proposes an approach for IT service management by business objectives called IT Management by Business Objectives (MBO). Their information model contains objectives, key performance indicators (KPIs), and perspectives, inspired by BSC. Decision support is provided through deployment of a reasoning engine, Aline, which computes the alignment of alternative courses of action to objectives, providing a measure of utility and basis for ranking the decision options and returning a recommendation. However, this novel approach is limited by the vagaries of assessing intangibles inherent in the perspectives, as in BSC. While there is other BDIM-related work, MBO is the one most closely related to our research.

Others have focused on design-time alignment of organizational goals, objectives, strategies or business requirements with process models. These include, for example, the *Tropos* development methodology based on *i** organizational modeling for requirements and design [13]; a framework for representing organizational strategies and goals in terms of business requirements using *Tropos* and *Formal Tropos* and implemented by activities in business processes through Web services using an early version of WSBPPEL [14]; co-evolution of operational business process models using the Business Process Modeling Notation (BPMN) and organizational models using the *i** modeling notation [15]; relating BPMN-based business process models to high level stakeholder goals modeled using KAOS [16]; an approach to process design and configuration management using requirements goal models to capture alternative

process configurations [17]; and a map-driven process modeling approach based on intentions and strategies abstracted from organizational tasks [18].

Furthermore, many languages and supporting tools have been developed by academia and industry for specification of policies, SLAs, and related IT system management constructs for service-oriented systems. They are often accompanied by corresponding run-time management middleware. While significant, these approaches tend to focus on monitoring of technical QoS metrics, providing limited business value metric-based control capabilities. The WS-Policy extensions WS-QoSPolicy [19] and WS-CoL [20], and the corresponding middleware, are closest to our research but they do not address self-managing BDIM.

3 Extension of WS-Policy4AMSC and MiniZnMASC with Key BMM Constructs

When changes (e.g., in system performance) occur during runtime, the affected Web service compositions should be adapted. If there is more than one available adaptation option, a decision is needed to resolve the 'conflict' to determine which adaptation option should be executed. It is often appropriate to maximize business value in such adaptation decision-making. To improve the support for business value considerations in adaptation of Web service compositions, we extended WS-Policy4MASC and MiniZnMASC with key constructs from the BMM industrial standard for modeling business intent. These constructs are from the End and Means hierarchies (but Directives are not included). In the BMM Means hierarchy (see Figure 1.b), a Mission is implemented through Strategies that, in turn, are implemented via Tactics. In a BMM End hierarchy (see Figure 1.c), a Vision is composed of Goals that are measured through Objectives. There are analogies between the BMM End hierarchy and the BMM Means hierarchy: Vision is the final end of the BMM hierarchy and is at the same abstraction level as Mission, Goal is at the same level as Strategy, and Objective is at the same level as Tactic.

A Vision can be achieved via a bottom-up approach by starting from Tactics. Contribution of particular lower-level constructs to the achievement of higher-level constructs (e.g., contribution of a Tactic to the achievement of a Strategy) and contribution of particular means to the achievement of ends (e.g., contribution of a Strategy to the achievement of a Goal) can be shown as directed arcs between nodes representing BMM constructs. These arcs can be in AND/OR relationships. The OR relationship means that any of the lower-level constructs can achieve the higher-level construct. The AND relationship means that all lower-level constructs must be achieved for achievement of the higher-level construct. To denote the strength of a particular contribution, we specify a Utility Contribution Weight (UCW) for each arc. A UCW is a relative value (between 0 and 1) that denotes how much of the business value produced at the higher-level BMM construct is due to the achievement of the lower-level BMM construct. In principle, the sum of all UCWs of arcs incoming into the same node should be 1. To denote probabilities in OR relationships, we specify Occurrence Probabilities (OPs). The sum of all OPs in an OR relationship should be 1. If an arc is not in any OR relationship, its OP is 1. If values for some of the UCWs or OPs are not given, default values are calculated based on the assumption that arcs with missing information have mutually equal contributions and probabilities. For example, if there are 2 arcs in an OR relationship and OPs are not given, the default values are OP=0.5 for both these arcs.

To illustrate how adaptation decisions are made using the BMM End and Means hierarchies, we provide an example of a business scenario. A loan broker company provides a loan brokering Web service composition shown in Figure 2. The BMM constructs and their values for the example are listed in Table 1. This company classifies its consumers into three classes according to the customers' previous loans record and credit history with the company: gold, silver, and bronze. The company provides different classes of consumer with different technical QoS guarantees, prices per year, and penalties if the guarantees are not met.

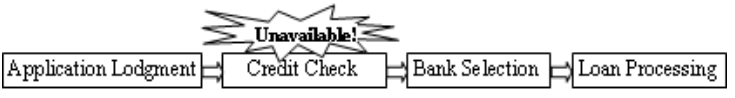


Fig. 2. A loan brokering Web service composition

Table 1. BMM constructs in the loan broker company

BMM Construct	Construct Value
Vision	To be a world-class company, helping people and organizations to grow
Mission	To be the provider of first choice across loan brokering services
Goal1	To increase product sales
Goal2	To improve customer satisfaction
Objective1	To have $300,000,000 turnover in this year of operation
Objective2	To build a reputation for quality, error-free services and products
Strategy1	Ensure that the service composition has a relatively high availability
Strategy2	Ensure that the completion time of the service composition is short
Tactic1	Use a replacement external credit service when the credit check service becomes unavailable
Tactic2	Skip the credit check service
Tactic3	Use an internal credit check service

The credit check service is a third-party service provided by an external credit check agency. During runtime, this service suddenly becomes unavailable for some reason. The adaptation system finds a replacement, service S, for the credit check service. It takes some time to set up the replacement service. Therefore, in addition to having an alternative tactic "use the replacement external service", the adaptation system also provides two other alternative tactics: "skip the credit check service" and "use an internal credit check service".

In this context, different tactics have different value contributions towards the business vision. To determine which Tactic to execute, business value contributions towards the business Vision have to be calculated according to policies. Figure 3 illustrates an example of BMM End and Means hierarchies of the loan-broking company. In this example, we identify three alternative tactics in the BMM hierarchies. The selected Tactics have to satisfy various constraints including cost limit constraints (in this example: $100 total cost) and other constraints.

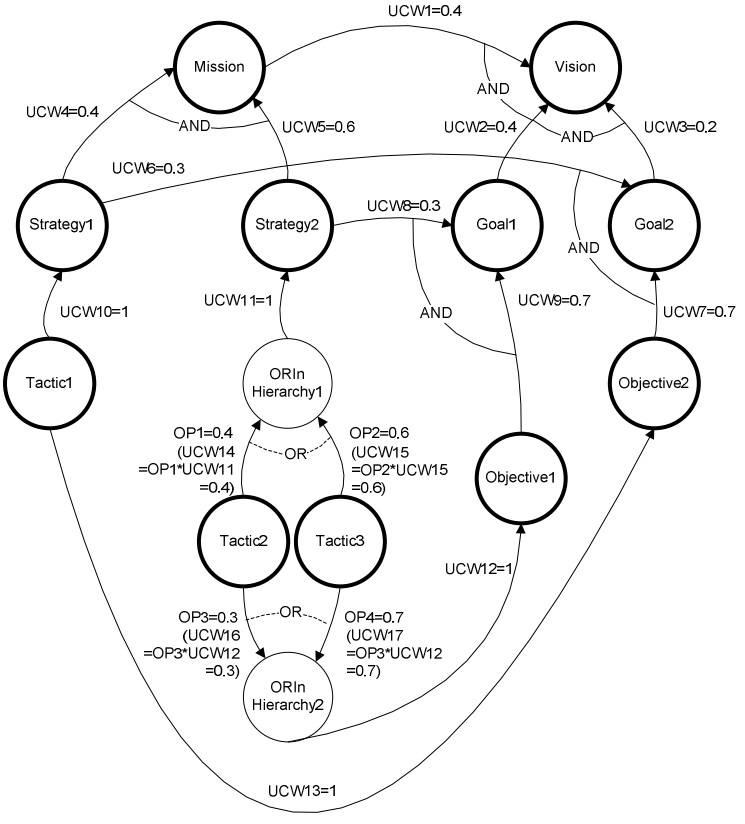


Fig. 3. BMM hierarchy in the loan broker company

Table 2. Business value for bronze consumers of each node in the BMM hierarchy

BMM hierarchy node	Business value
Vision	$10,000
Mission	$1,000
Goal1	$4,000
Goal2	$5,000
Strategy1	$200
Strategy2	$100
Objective1	$400
Objective2	$500
Tactic1	$10
Tactic2	$30
Tactic3	$20

In Figure 3, we illustrate how to use the BMM End and Means hierarchies to select one Tactic for each class of consumer, among several options. We assume that the same Tactic has different business values to different classes of consumers. To simplify the discussion, we only discuss how to determine which Tactic to select for bronze consumers. Table 2 shows the business value of each node for bronze consumers in the BMM End and Means hierarchies.

Starting from Tactic1, we have two options: Strategy1 and Objective2. Let us meet Strategy1 first (there is no difference in starting from Strategy1 or Objective2). We have two options to go after Strategy1: Mission and Goal2. Let's select Mission to meet first. After meeting Mission, we achieve Vision, which is the end point. After meeting Vision, we calculate the total business value of meeting each hierarchy node in the return path. The calculation for the loan broking example is shown in Figure 4. As it is the same way for other paths, we do not discuss repetitively here. V represents the business value of directly achieving a BMM hierarchy node. V' represents the total business value contribution a BMM hierarchy node makes and it includes the business value of directly achieving this hierarchy node and the sum of contributions this node makes to achieving each of the related higher-level BMM hierarchy nodes. As shown in Figure 4, Tactic3 brings the maximum high-level business value. Therefore, the algorithm selects Tactic3 as the adaptation action for bronze consumers.

1) **V**'(Vision)=V(Vision)=$10,000
2) V'(Mission)=V(Mission)+ UCW1*OP*V'(Vision)=$1,000+0.4*1*$10,000 = $5,000
3) V'(Goal1)=V(Goal1)+UCW2*OP*V'(Vision)=$4,000+0.4*1*$10,000=$8,000
4) V'(Goal2)=V(Goal2)+UCW3*OP*V'(Vision)=$5,000+0.2*1*$10,000=$7,000
5) V'(Strategy1)=V(Strategy1)+UCW4*OP*V'(Mission)+UCW6*OP*V'(Goal2)=$200 +0.4*1*$5,000+0.3*1*$7,000=$4,300
6) V'(Strategy2)=V(Strategy2)+UCW5*OP*V'(Mission)+UCW8*OP*V'(Goal1)=$100 +0.6*1*$5,000+0.3*1*$8,000=$5,500
7) V'(Objective1)=V(Objective1)+UCW9*OP*V'(Goal1)=$400+0.7*1*$8,000=$6,000
8) V'(Objective2)=V(Objective2)+UCW7*OP*V'(Goal2)=$500+0.7*1*$7,000=$5,400
9) V'(Tactic1)=V(Tactic1)+UCW10*OP*V'(Strategy1)+UCW13*OP*V'(Objective2) =$10+1*1*$4,300+1*1*$5,400=**$9,710**
10) V'(Tactic2)=V(Tactic2)+UCW11*OP1*V'(Strategy2)+UCW12*OP3*V'(Objective1) =$30+1*0.4*$5,500+1*0.3*$6,000=**$4,310**
11) V'(Tactic3)=V(Tactic3)+UCW11*OP2*V'(Strategy2)+UCW12*OP4*V'(Objective1) =$20+1*0.6*$5,500+1*0.7*$6,000=$**7,520**

Fig. 4. The process of calculating high-level business value in the loan-broker example

We extended our WS-Policy4MASC language and MiniZnMASC middleware with the key BMM concepts in four steps:

1) conceptually extended WS-Policy4MASC with all key BMM constructs;

2) coded the new extensions into the WS-Policy4MASC language schema and tested this on examples;

3) conceptually extended and improved the MiniZnMASC BDIM algorithms to use BMM-related features supported by WS-Policy4MASC, and;

4) implemented and tested the BDIM algorithm in Java.

Note that our solution is for IT system management of one operational business process; not the whole organization. The solution has to be modified and extended to be applied to the IT system management across the whole organization.

The starting point for our integration of BMM and WS-Policy4MASC was an examination of mappings between the BMM concepts and the WS-Policy4MASC concepts. Due to the space limitations, we focus in this paper only on the main aspects crucial for our algorithms that use BMM constructs in MiniZnMASC. The full mappings table and its discussion are available in [21].

```
<!-- Definition of the BMMHierarchy -->
<masc-bh:BMMHierarchy MASCID="BMMHierarchy1">
    <masc-bh:ToBeAchieved>
        <masc-bh:ActionPolicyAssertionRef To="Strategy1" />
    </masc-bh:ToBeAchieved>
    <masc-bh:ANDInHierarchy MASCID="ANDInHierarchy1">
        <masc-bh:ORInHierarchy MASCID="ORInHierarchy1">
            <masc-bh:UCWRef To="UCW7" />
            <masc-bh:AchievedBy>
                <masc-bh:OPRef To="OP3" />
                <masc-bh:ActionPolicyAssertionRef To="Tactic1" />
            </masc-bh:AchievedBy>
        </masc-bh:ORInHierarchy>
        <masc-bh:ORInHierarchy MASCID="ORInHierarchy2">
            <masc-bh:UCWRef To="UCW8" />
            <masc-bh:AchievedBy>
                <masc-bh:OPRef To="OP1" />
                <masc-bh:ActionPolicyAssertionRef To="Tactic2" />
            </masc-bh:AchievedBy>
            <masc-bh:AchievedBy>
                <masc-bh:OPRef To="OP2" />
                <masc-bh:ActionPolicyAssertionRef To="Tactic3" />
            </masc-bh:AchievedBy>
        </masc-bh:ORInHierarchy>
    </masc-bh:ANDInHierarchy>
</masc-bh:BMMHierarchy>
```

Fig. 5. An example of the BMMHierarchy definition

The corresponding concept for BMM Means in WS-Policy4MASC is the action policy assertion, while the corresponding concept for BMM End in WS-Policy4MASC is the

goal policy assertions. In order to specify relationships between hierarchies of BMM ends and means, we added the new "BMMHierarchy" construct to WS-Policy4MASC. The sub-elements under "BMMHierarchy" are "ToBeAchieved" and "ANDInHierarchy". "ToBeAchieved" is what the arc arrow between two nodes in the BMM hierarchy points to. The elements under "ToBeAchieved" can be "GoalPolicyAssertionRef" or "ActionPolicyAssertionRef". "ANDInHierarchy" has the sub-element "ORInHierarchy". There should be at least one "ORInHierarchy" bounded to an "ANDInHiearchy". "ORInHierarchy" has two sub-elements: "UCWRef" and "AchievedBy". "UCWRef" refers to utility contribution weights for the "AchievedBy". The sub-elements under "AchievedBy" are "ActionPolicyAssertionRef" or "GoalPolicyAssertionRef", and "OPRef". "ActionPolicyAssertionRef" or "GoalPolicyAssertionRef" refers to what the arrow between two nodes in the BMM hierarchy points from. "OPRef" refers to the occurrence probability of "AchievedBy". Figure 5 shows an example of the BMMHierarchy definition.

The extended algorithm for selection of the best options among conflicting action policy assertions is outlined in Figures 6 and 7. Here, the "best" means "highest overall business value, while meeting all constraints" and depends on which combinations of the business value metric categories are used in the calculations as well as on the contribution of particular conflicting action policy assertions to long-term business goals and strategies. This algorithm enables choosing an adaptation option that might not be the best in the short-term, but is the best one when longer-term high-level business considerations are taken into account.

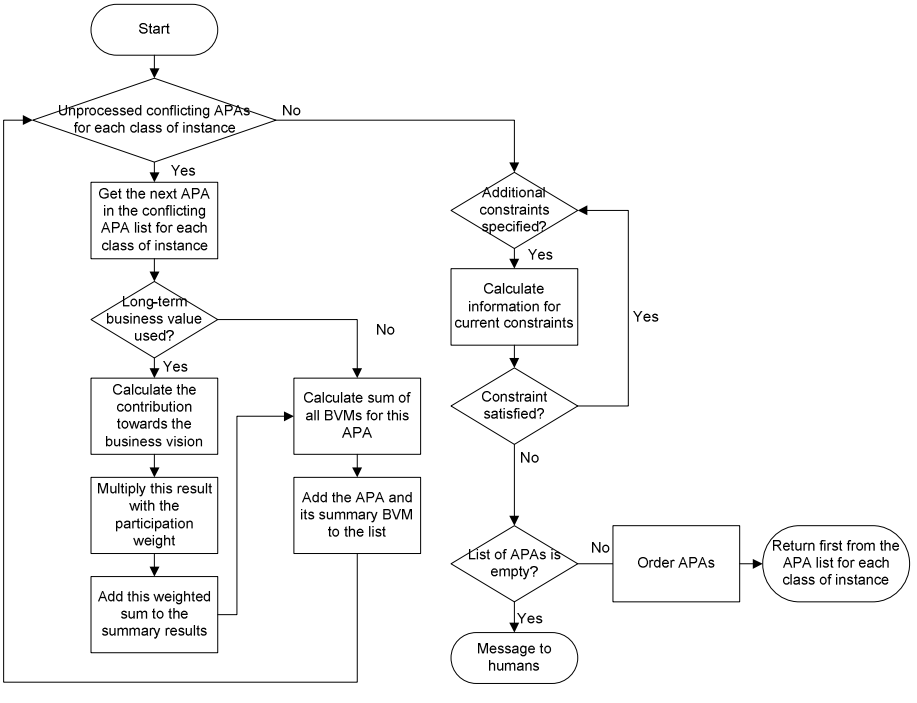


Fig. 6. The conflict resolution algorithm

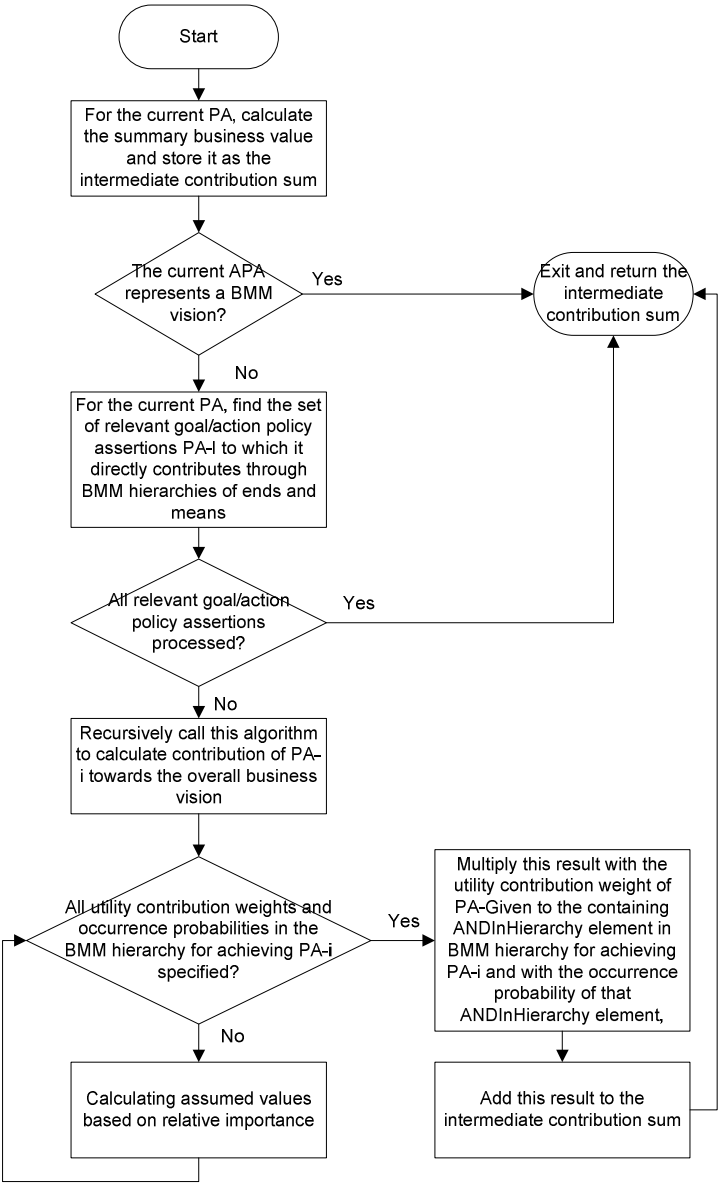


Fig. 7. Calculation of value contribution towards the business vision

The algorithm first loops through all conflicting action policy assertions (APAs) to check whether each of them satisfies all given constraints and, if yes, to calculate the sum of all business value metrics (BVMs) for these APAs. The APAs that satisfy all constraints are added to the list, along their summary BVMs. If none of the conflicting APAs satisfies the constraints, the resulting list will be empty and an

exception is thrown (e.g., human administrators are notified to resolve the issue). The APAs in the list are ordered based on the decreasing value of their summary BVMs. The first APA in the list will have the highest summary BVM while satisfying all constraints. This APA is returned as the adaptation action (which could contain several sub-actions) to be executed.

Notice in Figure 6 that the algorithm checks whether the long-term business contribution is considered in the adaptation decision-making. If yes, the algorithm calculates the contribution towards the BMM business vision, multiplies the result with the participation weights, and then adds this weighted sum to the currently examined action policy assertion as its long-term business value. As shown in Figure 7, if contribution towards long-term business vision is considered in decision-making, then when the summary business value metric is calculated for each adaptation option, the extended decision making algorithm adds weighted contribution towards the overall BMM business vision. The algorithm for calculation of the contribution to the overall business vision traverses the AND/OR hierarchy of BMM ends and means, calculates the summary business value for each hierarchy node, and applies utility contribution weights and occurrence probabilities.

4 Evaluation

We evaluated MiniZnMASC and the developed algorithms for long-term business-driven decision-making on four aspects: feasibility, functional correctness, performance overhead, and scalability. We evaluated feasibility through implementation of several proof-of-concept prototypes. We found no problems with feasibility. We implemented the motivating example from Section 3 and evaluated the functional correctness of MiniZnMASC by comparing the results calculated by MiniZnMASC and by hand. We also developed several other examples for this evaluation. The results showed that MiniZnMASC had been built correctly.

For the performance and scalability tests, we used a Hewlett-Packard laptop model HP EliteBook6930p with Intel Core 2 Duo CPU T900 2.53GHz processor and 4.00 GB of RAM memory, running 32-bit Windows Vista operating system. We measured the performance with increasing number of conflicting action policy assertions (BMM Tactics) to be examined. We started with 3 action policy assertions, then increased to

Table 3. Performance measurement results with increasing number of conflicting action policy assertions (BMM Tactics)

Test case	Execution time of decision making	Execution time of the whole conflict resolution algorithm
3 conflicting action policy assertions	Average: 141 ms Range: 125-157 ms	Average: 47 ms Range: 46-63 ms
10 conflicting action policy assertions	Average: 218 ms Range: 203-343 ms	Average: 78 ms Range: 78-78 ms
100 conflicting action policy assertions	Average: 1186 ms Range: 1029-1389 ms	Average: 437 ms Range: 434-453 ms

10, and then to 100. Using the Java "System.currentTimeMillis()" call, we measured the execution time of autonomic business-driven decision making and the execution time of the whole conflict resolution algorithm in MiniZnMASC. We repeated 100s of tests at different times of day and averaged their results.

Table 3 shows the measured results of the range (min, max) and average of the execution time. The overall execution time of decision making in MiniZnMASC rises because the execution time of the summation of business values for each conflicting action policy assertion increases with increasing number of conflict action policy assertions. The last test case (100 conflicting action policy assertions) is much more complicated than realistic scenarios in practice, so 1.186 sec is not an issue. It is important to note that in realistic scenarios of MiniZnMASC the number of conflicting action policy assertions will be low (typically 2, maybe a few more) while the overall number of action policy assertions can be huge. We also checked that the number of additional non-conflicting action policy assertions in the MiniZnMASC Policy Repository had no significant effect on performance, even when there were hundreds of action policy assertions.

5 Conclusions and Future Work

The main contribution of our research is in aligning organization-level business intentions with run-time execution of service-oriented systems by extending existing language (WS-Policy4MASC) and middleware (MiniZnMASC) for autonomic business-driven IT management with key concepts from the industrial standard for modeling business intent (BMM). These key concepts include hierarchies of ends (e.g., goals) and means (e.g., strategies). We also developed new decision-making algorithms for our MiniZnMASC middleware. The new algorithms leverage information in the extended WS-Policy4MASC to align run-time IT system management decisions with business concerns.

Our ongoing work is on extending WS-Policy4MASC and MiniZnMASC with the remaining BMM concepts (e.g., business policies, business rules, directives, influencers, and assessment). We already have conceptual solutions and WS-Policy4MASC extensions, but we still have to complete the new MiniZnMASC prototype and experiment with it.

Acknowledgments. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. We thank Nahid Ebrahimi Nejad, Shyam Sunder Iyer, and Cheng Li for their contributions to implementation of the extended WS-Policy4MASC and MiniZnMASC prototypes.

References

1. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. Computer 36, 41–50 (2003)
2. Bartolini, C., Sahai, A., Sauve, J.P.: Proceedings of the Second IEEE/IFIP Workshop on Business-Driven IT Management (2007)

3. Casati, F., Shan, E., Dayal, U., Shan, M.C.: Business-oriented management of Web services. Communications of the ACM 46, 55–60 (2003)
4. Tosic, V.: Autonomic business-driven dynamic adaptation of service-oriented systems and the WS-Policy4MASC support for such adaptation. Intl. J. of Systems and Service-Oriented Eng (IJSSOE) 1, 79–95 (2010)
5. Tosic, V.: On Modeling and Maximizing Business Value for Autonomic Service-Oriented Systems. In: Ardagna, D., Mecella, M., Yang, J. (eds.) Business Process Management Workshops. LNBIP, vol. 17, pp. 422–433. Springer, Heidelberg (2009)
6. OMG-BMM: Business Motivation Model Version 1.0, http://www.omg.org/spec/BMM/1.0/PDF
7. Lu, Q., Tosic, V.: Support for Concurrent Adaptation of Multiple Web Service Compositions to Maximize Business Metrics. In: IM 2011. IEEE, Dublin (2011)
8. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a Starndard CP Modelling Language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
9. Kaplan, R.S., Norton, D.P.: Having trouble with your strategy? Then map it. Focusing Your Organization on Strategy—with the Balanced Scorecard. 49 (2000)
10. Van Grembergen, W.: The balanced scorecard and IT governance (2000)
11. Kaplan, R.S., Norton, D.P.: Measuring the strategic readiness of intangible assets. Harvard Business Review 82, 52–63 (2004)
12. Bartolini, C., Sallé, M., Trastour, D.: IT service management driven by business objectives An application to incident management, pp. 45–55. IEEE (2006)
13. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems 27, 365–389 (2002)
14. Kazhamiakin, R., Pistore, M., Roveri, M.: A framework for integrating business processes and business requirements (2004)
15. Koliadis, G., Vranesevic, A., Bhuiyan, M., Krishna, A., Ghose, A.: A combined approach for supporting the business process model lifecycle. Citeseer (2006)
16. Koliadis, G., Ghose, A.: Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. In: Hoffmann, A., Kang, B.-h., Richards, D., Tsumoto, S. (eds.) PKAW 2006. LNCS (LNAI), vol. 4303, pp. 25–39. Springer, Heidelberg (2006)
17. Lapouchnian, A., Yu, Y., Mylopoulos, J.: Requirements-Driven Design and Configuration Management of Business Processes. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 246–261. Springer, Heidelberg (2007)
18. Nurcan, S., Etien, A., Kaabi, R., Zoukar, I., Rolland, C.: A strategy driven business process modelling approach. Business Process Management Journal 11, 628–649 (2005)
19. Rosenberg, F., Enzi, C., Michlmayr, A., Platzer, C., Dustdar, S.: Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL, p. 15. IEEE Computer Society (2007)
20. Baresi, L., Guinea, S., Plebani, P.: Policies and Aspects for the Supervision of BPEL Processes. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 340–354. Springer, Heidelberg (2007)
21. Iyer, S.S.: Analysis of Methods for Improving IT Support for Business. University of New South Wales (2009)

WS-Governance: A Policy Language for SOA Governance*

José Antonio Parejo, Pablo Fernandez, and Antonio Ruiz-Cortés

Universidad de Sevilla, Spain

Abstract. The widespread use of Service Oriented Architectures (SOA) is beginning to create problems derived from the governance of said structures. To date there is not a single effective solution to solve all existing challenges to govern this type of infrastructure. This paper describes the problems encountered when designing a SOA governance solution in a real e-Government scenario. More specifically, we focus on problems related to specification and automated analysis of government policies. We propose a novel SOA governance specification model as a solution to these problems. We have named this model WS-Governance. In order to ease its adoption by SOA practitioners it: i) shares WS-Policy guidelines and is compatible with it, ii) has XML serialization as well as a plain-text one and iii) has a semantics based on a mapping to Constraint Satisfaction Problems that provides a precise description as well as facilitating the automation of some editing and WS-Governance related activities such as consistency checking.

1 Introduction

SOA adoption brings an increase on the number of elements of the IT architecture, where proper management and control become capital issues. In this context, SOA Governance is defined as the management process aimed at delivering the SOA promise of reuse, business goals support and responsiveness [1,2]. According to [3] SOA Governance Lifecycle can be divided into six stages, from more abstract business levels to more concrete operational levels: Create a SOA strategy, Align Organization, Manage Service Portfolio, Control Service Lifecycle, Policy Definition and Enforcement and Service Level Management. In this paper we focus on the policy definition as the key stage that requires a deeper analysis in order to support an agile governance.

Effective governance requires policy management, including : (i) the definition of policies that encode governance rules and (ii) the establishment of appropriate conformance testing and enforcement mechanisms. Moreover, we have identified the need to incorporate the structure of the organization as essential information to take into account when the governance policies are designed. In our case study, the structure, size and departmental autonomy of the organization implies that multiple administrators could specify policies in a distributed and independent way, boosting the possibility of specifying inconsistent policies. In this context, the capability of automatic consistency

* This work has been partially supported by the European Commission (FEDER) project SETI (TIN2009-07366), and project ISABEL P07-TIC-2533 funded by the Andalusian local Government.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 280–296, 2011.

checking of policies is highly valuable, and governance tools should support it. However, the current governance tools market is vendor-driven and turbulent, where tools are based on proprietary technology and their features are guided by the specific aspects where their vendors have expertise [4,5].

The contribution of this paper is twofold: (i) First, a language for governance policies definition is presented. This language defines governance documents; which make policies unambiguous by providing a rich context for governance policies and their meta-data. This is done whilst maintaining their definition independently of the SOA elements to govern; their internal organization and the underlying infrastructure. (ii) Secondly, a formal definition of governance document is proposed, describing the elements to govern, their properties and the policies that govern them. This formal definition allows the automation of policies' consistency checking.To the best of our knowledge, this proposal provides a novel approach, paving the way for building more powerful and automated governance tools. This proposal has been developed and tested by creating two applications: a consistency analyzer and an on-line editor.

The rest of the paper is structured as follows: In sec. 2, the case study that motivates the research presented on this paper is described. Sec. 3 depicts the limitations of WS-Policy for governance policy specification and motivates the need of a governance document. Sec. 4 presents WS-Governance, our XML-based language proposal for governance policies specification, and its plain text equivalent WS-Gov4People. Sec. 5 presents a mapping of governance documents to Constraint Satisfaction Problems (CSPs) that allow the automated checking of consistency. Finally, in sections 6 and 7 related work is described and conclusions are drawn.

2 A Motivating Use Case

The motivation of our approach is derived from a case-study based on a real scenario we addressed on a research project, involving a regional-wide governmental organization. This organization has a complex structure divided into 16 governmental departments and thousands of end users using a shared IT infrastructure. This infrastructure is distributed in the different departments both logically and physically and is usually managed autonomously in each location. In recent years there has been a shift toward SOA in the organization, and currently there is an important number of core services replicated in the infrastructure with different QoS capabilities.

From an architectural point of view, the infrastructure is designed as a federated bus of services; in this context, each department represents a node with two main elements: an Enterprise Service Bus and a Management System that provide different horizontal functionalities (such as monitoring, transactions or security). All the different nodes are integrated conforming the global infrastructure. The different services are deployed in the bus and the consumer applications ask the bus for the appropriate provider.

Due to the structure of the organization, each department has developed a high autonomy in its IT infrastructure management. Consequently, the integration of applications and services amongst different departments has raised an important issue: the need to

specify a consistent normative framework on the whole organization for meeting business needs without breaching autonomy. Such a framework can be created by specifying a set of Governance Policies. A Governance Policy represents a capability, requirement or behavior that allows the SOA to achieve its goals, and whose meeting is quantifiable and monitorable through time [6,1,7,8]. Governance policies are as heterogeneous as the said governed elements, addressing the distributed, flexible, and heterogeneous nature of current SOAs. Moreover, governance policies originate from disparate sources, from legal regulations and their derived compliance issues to strictly technical details.

There is a real and urgent need of a language to define governance policies unambiguously with precise semantics; as a first step toward governance policy definition, enforcement and automatic consistency checking.

Figure 1 shows three excerpts of different real governance documents found in our case study -conveniently modified in order to preserve privacy and meet confidentiality clauses-. Currently, these fragments correspond to human-oriented policies that should be enforced by administrators by means of configurations of the IT-infrastructure. Each document is enacted by a different organization: the first document by the main authority so it should be enforced by all sub-organizations (departments); the second document represents an integration agreement amongst two departments (1 and 2) and finally, the last document is an internal governance document of department 1.

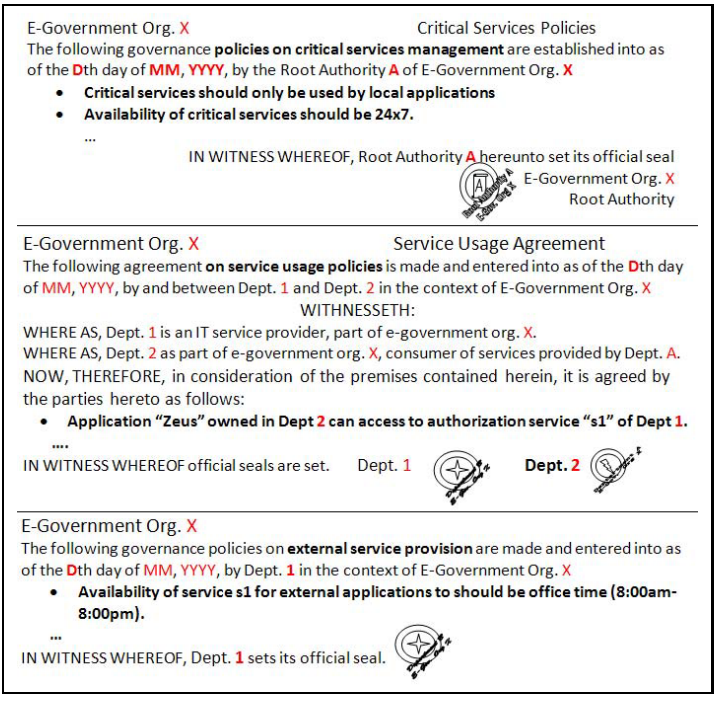


Fig. 1. Governance Documents

3 Using WS-Policy for SOA Governance

In working for a public administration, we were concerned with developing mainstream policies that would avoid ad-hoc solutions. For this reason, we decided to use the W3C recommendation WS-Policy for defining policies [9]. As can be seen in Fig. 3 where the UML metamodel of WS-Policy is shown[1], the building blocks of policies are assertions (*PolicyAssertion*) that are composed using *and*, *or* and *xor*–like compositors (*All*, *ExactlyOne*, and the top level compositor *PoliciAlternative*). Policy nesting is supported by meaning a logical AND operation of the assertions of the global policy and those of the nested one.

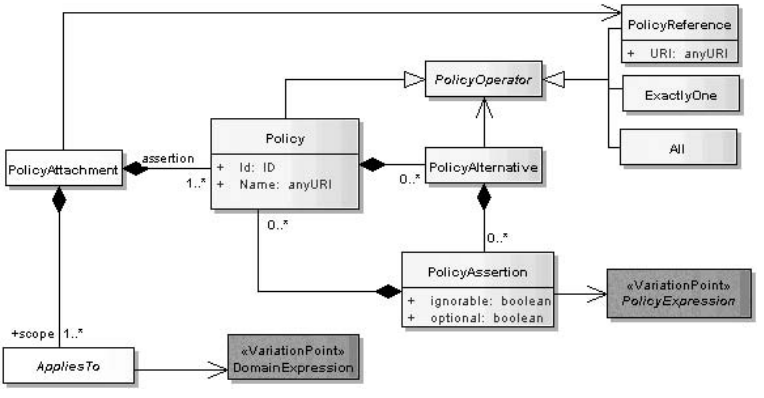


Fig. 2. UML metamodel of WS-Policy

Assertions represent domain-specific capabilities, constraints or requirements, where their grammar is left open by WS-Policy, thus allowing the use of XML-based Domain Specific Languages (DSLs) for that purpose (see the "VariationPoint" stereotype of PolicyExpression in Fig. 3). WS-Policy has been mainly focused on the definition of policies related to specific service capabilities such as security, reliability, etc. In fact, there are a number of DSLs for those purposes. Unfortunately, describing assertions for SOA governance policies is more complex and as far as we know there is currently no DSL to describe this kind of policies.

Two mechanisms are available in WS-Policy to associate policies with the subjects to which they apply, *i.e.* for defining their scope. The first one associates one or more policy definitions as a part of the subject definition. For example, if we want to apply a policy to a web service described in WSDL, the policy specified in WS-Policy has to be inserted into the WSDL code. We call this mechanism *endogenous attachment*, since the definition of the policy is internal to the element one. Left column in table 1 shows an example of this kind of attachment. In this case, two polices on the web service "StockQuote" are defined: one to ensure a reliable message and another to specify security mechanisms

[1] The UML class diagram in 3 represents our interpretation of the metamodel described by the XML schema specified in [9] and [10].

on web service binding. Notice that with endogenous attachment: i) attaching a set of policies to a set of subjects at the same time is not possible and ii) changing a policy requires modifying the definition of the subject, *i.e.* it is an *intrusive* mechanism.

The second mechanism associates one or more policy definitions to one or more subjects. The WS-PolicyAttachment recommendation [10] proposes the use of *PolicyAttachemt* and *AppliesTo* (shaded classes in Fig. 2). In this case, the WS-Policy is not encoded in the same file that the specification of the element, thus we call this mechanism *exogenous attachment*. Right column in table 1 shows an example of this kind of attachment. In this case, secure binding mechanisms are asserted on the "StockQuote" and "MortgageRisk" services using its endpoint reference address. As shown in Fig. 2, WS-PolicyAttachment also leaves open the language to specify the scope, but provides a basic language to specify it based on URIs.

Note that with exogenous attachment: i) it is possible to attach a policy to a set of elements at the same time and ii) changing the policy attachment does not require modifying the definition of an element, *i.e.* it is *non–intrusive* mechanism.

Table 1. Endogeous vs Exogeous Attachment

WS-Policy 1		WS-Policy 2	
<wsdl:definitions name='StockQuote' xmlns:wsp='...' ...>	Element attached	<wsp:PolicyAttachment>	
<wsp:Policy wsu:Id='RmPolicy' >	\	<wsp:AppliesTo>	\
<rmp:RMAssertion>	|	<wsa:EndpointReference>	| Policy
<wsp:Policy/>	|	<wsa:Address>.../MortageRisk.wsdl</...>	| Scope
</rmp:RMAssertion>	|	<wsa:Address>.../StockQuote.wsld</...>	| Scope
</wsp:Policy>	|	</wsa:EndpointReference>	| def.
<wsp:Policy wsu:Id='X509Policy'	| Policy	</wsp:AppliesTo>	/
<sp:AsymmetricBinding>	| assertions	<wsp:Policy wsu:Id='X509Policy'	\
...	|	<sp:AsymmetricBinding>	|
</sp:AsymmetricBinding>	|	...	| Policy
</wsp:Policy>	/	</sp:AsymmetricBinding>	|assertions
...		</wsp:Policy>	/
</wsdl:definitions>		</wsp:PolicyAttachment>	

WS-Policy defines a mechanism to test the compatibility of two policies, called *policy intersection*. According to the WS-Policy specification [9] "policy intersection is optional but a useful tool when two or more parties express policies and want to limit the policy alternatives to those that are mutually compatible". The intersection consists of two parts: a domain-independent policy intersection and domain-specific processing. The former takes into account the assertion type equality, *i.e.* the XML element type equality and its nested elements, but not its parameters. The latter is not defined in WS-Policy, thus assertions authors have to define specific mechanisms for incorporating the intended semantics of the assertions, and the specification does not provide any mechanism to integrate or define it. For instance, the first policy specified in the left column and the policy in the right column of table 1 are incompatible, since their element types *<rmp:RMAssertion>* and *<sp:AsymmetricBinding>* are different. The notion of policy intersection is thus basically syntactical and structural, and it is not valid for complex domain-specific processing or semantic reasoning about policies, as shown in [11] and [12].

When using WS-Policy to specify the policies the previously described documents we encountered the following limitations:

Lack of Context and meta-data (LCD): Governance policies need a rich context to ensure their validity, specifying who enacts the policies and providing additional meta-data, in order to ensure authorization for policy enactment and the integrity of the policies as enacted, thus avoiding tampering. This is the role of seals and signatures of the real documents shown in figure 1. In using WS-Policy, there is not a single point where we can insert the required information that assures the validity of the policy, such as official seals, a declaration of validity by the enacting authority or a preamble. We call this problem Lack of Context Data (LCD).

Scope Definition Limitations (SDL): Defining a policy P1 as simple as "*All services provide an Availability greater than 99%*" may become a nightmare since WS-Policy has not been designed keeping in mind that the scope of a policy could be defined by intension; *i.e.* we need specify the properties of services belonging the scope, not enumerate them. If an exogenous attachment with the proposal described in the WS-Policy specification is used, then all the services references will be inserted in the *AppliesTo* section of the policy. Thus, if there was a change in the policy scope, like *All services except S23, S45, . . .*) this would entail a re-working of all services as well as modifying the content of the scope section. This is not an adequate solution when dealing with a SOA comprising of hundreds of applications and services. Summarizing, governance policies' scope should be defined by intension through scope predicates in most cases.

The expression of these scope predicates require a rich predicate DSL and the specification of the elements and data sources needed to feed the predicate, in order to effectively evaluate policy scope. This problem is particularly acute for governance policies, since governance relevant information is stored in disparate sources, such as UDDI registries, LDAP directories, *ad hoc* databases, etc. For instance, in our sample GDs (fig. 1) there are properties such as "*critical services*" and "*local/external apps & services*" whose values must be obtained from different information sources.

However, WS-Policy extension mechanisms allow the definition of DSL for specifying policy scope (see fig. 3), thus we propose such a DSL as part of our proposal.

Inadequate consistency checking: In our use case, the most valuable property was consistency checking. The only analysis operation WS-Policy envisioned for this was the intersection of policies. The open and flexible nature of WS-Policy makes it difficult to provide homogeneous semantics to policies, since each domain specific DSL would have its own semantics. This problem motivates the merely structural-syntactical nature policy intersection operation as defined in WS-Policy, avoiding its usage in diverse scenarios [11,12]. For example, two identical-meaning policies may prove inconsistent by using the intersection operation (see [11]). We name this drawback as **Syntax/Structure Driven Semantics (SDS)**. This limitation can be addressed by defining a domain specific intersection processor, and consequently, our proposal for providing semantic consistency checking can be integrated in such a way. However, authors consider that defining an entirely new operation called consistency and provide an explicit semantic for the DSLs defined is a better approach. Otherwise, the intersection operation, could process some assertions based on their structure, while others will be

treated semantically, leading to to incoherent results (as shown in [11]). Thus, we leave intersection as an essentially sintactically/structural operation and define a new operation named consistency that supports semantic reasoning based on CSPs.

4 From WS-Policy to WS-Governance

WS-Governance Documents (GDs) address limitations described previosly by incorporating an extensible context to the contained policies (addresing LCD), and defining a global document structure that contains a set of policies under an umbrella governance scope (where the data sources that feed predicates for defining policies scope can be included, addressing SDL). They also describe relevant governance properties of services, applications and organizational structures, and provide mechanisms for incorporating disparate governance-relevant information sources. Moreover, LCD and SDS motivate the creation of two general purpose XML-based DSLs for specifying governance policy assertions and SOA modelling, that are described in detail below. Based on those DSLs and the authors‘ experience providing formal semantics for SLAs specified in WS-Agreement by using CSPs [13,14,15], a CSP based semantics for WS-Governance documents using those DSLs is proposed in Sec. 5 addressing SDS. Based on those semantics a consistency property for policies and governance documents is defined. The structure of a GD in WS-Governance comprises of:

- **Governance Document Context:** It currently defines the governing organization but its grammar is left open in order to support the expressions of authorizations to enact policies on this GD, and the data needed to ensure GD authenticity and integrity.
- **Governance Scope:** It provides information about the SOA where policies are established. Different information sources could be used in this section, from UDDI registries to ad hoc databases, since any SOA element could be a governance policy subject; such as projects, developers, organizations, messages, XML-schemas or applications servers.
- **Governance Properties:** It defines all properties that are relevant for governance policies. Following the philosophy of WS-Policy, its grammar is left open, allowing the use of XML-based DSLs for specifying those properties.
- **Governance Policies:** It defines the policies that conform the governance. Those polices are fully WS-Policy compliant, where the exogenous policy attachment mechanism is mandatory. Assertion and scope definition grammar is left open, allowing the use of XML-based DSLs.

It is noticeable, that neither the definition of the new concept of Governance Document, nor the use of the DSLs for Governance Policies, break WS-Policy compatibility. Although authors consider that the document-oriented treatment of policies is more natural in governance contexts (as shown in the example of fig. 1) and better supports the life-cycle of policy creation [16], from authoring by humans to deployment into servers; any governance document can be transformed into a unique policy fully compliant with WS-Policy[2]. Moreover, the DSLs described below ares used in WS-Policy extension

[2] In `http://labs.isa.us.es/gda/WS-Policy-transformation.xslt` is available an XSLT tranformation that allows to perform this conversion automatically.

mechanisms and variation points as described in fig. 3 for providing suitable languages for governance policy definition. The UML class diagram shown in Fig. 3 represents our proposal of metamodel for WS-Governance documents.

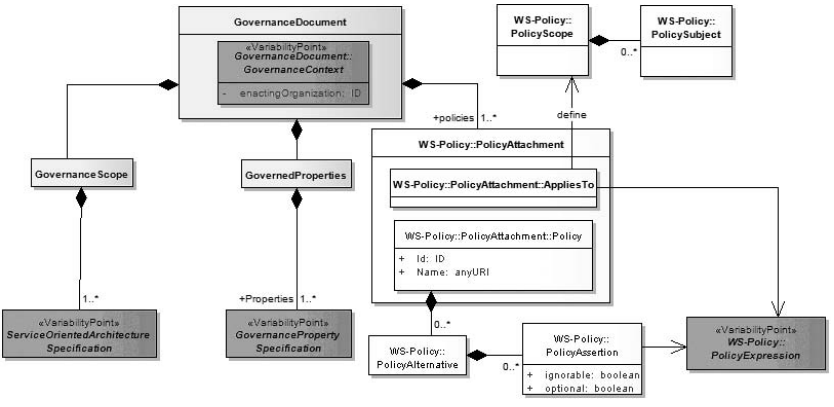


Fig. 3. UML Metamodel of WS-Governance

Some elements of WS-Governance are intentionally left open for extension in order to allow a high degree of flexibility. This flexibility is based on the use of XML-based DSLs in some variability points, allowing the creation of a whole family of governance languages. In Fig. 3 variability points are decorated with a *VariabilityPoint* UML stereotype. A brief description of these variability points is provided as follows:

- **Context DSL:** The GD metadata in the context element can be extended with any information needed by means of the nesting of new XML elements and attributes.
- **SOA Specification:** The architecture and elements to govern must be described in order to define unambiguous policies.
- **Governance Property Specification:** A description of the properties of the governed elements of the SOA is needed in order to define expressive policies. Those properties must be expressed using a XML-Based DSL.
- **Policy Expression Specification:** Policy scope and assertions can be expressed using any predicate-oriented DSL.

In order to define effective governance documents, those DSLs must be set. In our proposal we provide two DSLs that allow the creation of service-focused governance policies, *i.e.* policies that specify assertions defined on service properties and their directly related elements such as consumers, providers, and governance relevant information such as organizational structure. Those DSLs are *Service oriented Architecture Description Language (SADL)*, addressing the SOA Specificaton variation point, and *Governance Assertion Language (GAL)*, addressing the Governance Property Specification and Policy Expression Specification variation points. The UML Class Diagrams in Figs. 4 and 5 depicts the metamodel of SADL and GAL respectively.

4.1 SOA Modeling with SADL

SADL has been designed to model the SOA state and structure, making our proposal independent of the specific governance information sources available on each SOA, such as UDDI Registries, LDAP directories, *ad hoc* databases, etc. SADL describes both the SOA structure as elements, and its state as the corresponding governance properties values for those elements. In this paper we focus on service-related governance policies, so SADL mainly contains elements related with services; however SADL is extensible, supporting the use of any XML-based construct as sub-elements of its basic structural elements. Specifically, structural elements in SADL are described as follows:

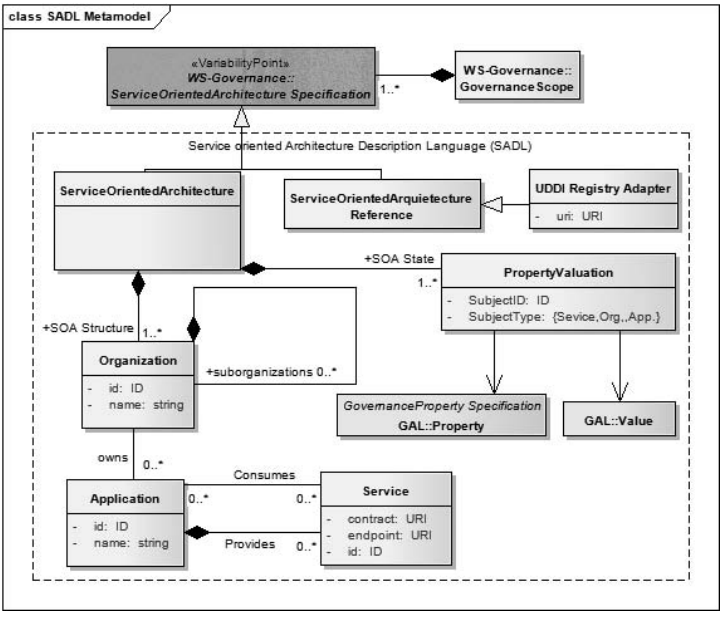


Fig. 4. UML Metamodel of SADL

- Service Oriented Architectures are networks of participants providing and consuming services to fulfill a purpose. In SADL these participants are specified as organizations and applications.
- Organizations are participants with governance relevant identity and properties, tracing an organizational boundary on their owned applications and services. Organizations are arranged hierarchically, where an organization can contain various sub-organizations (*e.g.* departments) and have a unique parent.
- Applications represent business processes, related capabilities and software packages. They allow the arrangement of software artifacts and capabilities independently of the organizational hierarchy in a governance-meaningful way. Applications are owned by a unique organization. Applications have a set of provided and consumed services.
- Services represent capabilities that participants provide and consume.

Regarding SOA state description, SADL allows the specification of property values for all the aforementioned elements based on GAL.

Finally, SADL provides a generic element for the specification of the concrete governance data sources as references; such as UDDI registries, that should be queried to obtain the governance-relevant SOA structure and state in order to check properties and test policies adherence. By creating adapters that query those data sources and create a SADL compliant SOA model, our proposal becomes independent of those specific data-sources, thus semantics of GDs are based on explicit SADL models.

4.2 Specifying Governance Properties, and Policy Assertions with GAL

Governance Assertion Language. GAL is a generic and expressive language designed to declare governance properties and assertions. A governance property is any characteristic attribute of the elements of the SOA wich is relevant for the specification of a policy (either in their scope definition or in their assertions) whose value can be specified or retrieved from a governance data source. Property definitions in GAL have a name and an identifier as attributes, comprising of: (i) type definition, where basic XML-Schema [17] types are supported, (ii) an optional domain definition that restricts the space of valid values of the property; where it could be described as a GAL assertion (by intension) or as a set of values (by extension); and (iii) an optional SADL governance subject declaration, that defines the type of SOA element that can present the property (service, organization, policy, all, etc.). Through GAL constraints we provide a suitable language to specify policy assertions on governance properties. Assertions can be composed using WS-Policy composition operators: All, ExactlyOne and PolicyAlternative.

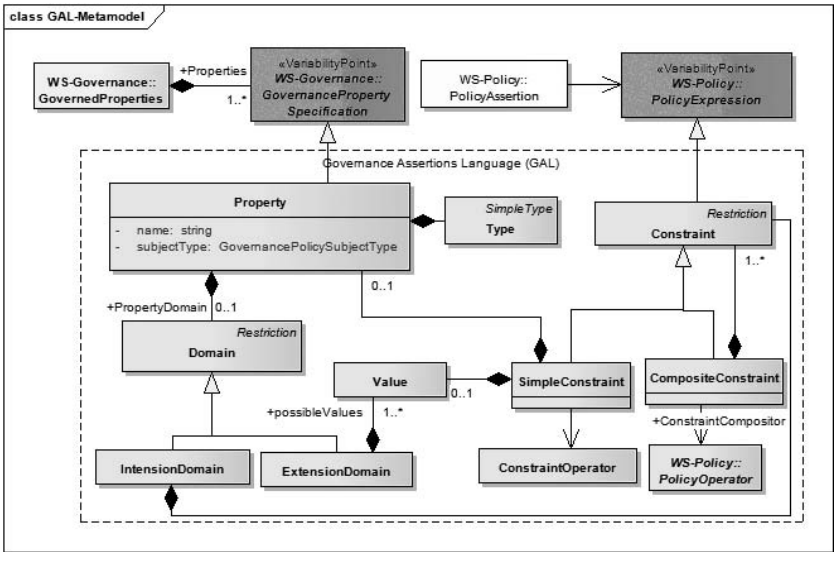


Fig. 5. UML Metamodel of GAL

In order to allow consistency checking, in this paper we use a subset of WS-Governance a bit less expressive, called WS-Governance*. A WS-Governance* document must use SADL to describe the SOA to govern and GAL to define governance properties, policy scopes and policy assertions. A WS-Governance* document (ρ) comprises of:

- Governance Scope defines the set of organizations O, applications A and services S to govern, and their relationships, namely: consumption of services by applications and organizations, provision of services by applications and organizations, ownership of applications by organizations and hierarchy of organizations. Only those elements and relationship functions are used to define policies.
- Governance Vocabulary must define the set of all properties V used in the guarantee terms.
- For each governance policy both scope (s) and assertion (a) must be defined as GAL assertions on the properties defined in the governance vocabulary section, and only to those applied to the sets and relationship functions defined in Governance Scope.
- Policy assertions can be composed using the compositors defined in WS-Policy.

XML-Schemas that model WS-Governance* documents conforming the previously described syntax are available at `http://www.isa.us.es/gda/schemas`. Although readable for humans, XML is not as understandable as plain text. In [18] the structure of a WS-Governance* document is described in a plain text language, named WS-Gov4People, that is equivalent to the WS-Governance*. The mapping of XML elements onto its corresponding WS-Gov4People sentences is also shown in table 2 of [18]. A WS-Gov4People document describing the policies specified in figure 1 and a simple SOA structure is shown in the first column of table 4 of [18].

5 Automatic Consistency Checking through CSPs

As shown in [19], the definition of the semantics of a language can be accomplished through the definition of a mapping between the language itself and another language with well-defined semantics such as Abstract State Machines, Petri Nets, rewriting logic or CSPs. These semantic mappings between semantic domains are very useful not only to provide precise semantics to DSLs, but also to be able to simulate, analyze or reason about them. In this section we define the mappings that tranform GDs* onto *Constraint Satisfaction Problems* (CSPs) that provide their precise semantics, allowing the usage of CSP solvers to reason about policies and complete GDs*. A CSP $\psi = (V, D, C)$ is defined as a set of variables V, a set of domains D (one for each variable), and a set of constraints C specifying which combinations of variables and values are acceptable. A solution σ to a CSP ψ consists of an assignment in which each variable gets a value from its corresponding domain, as long as it satisfies each constraint. The solution space of a CSP ψ, denoted as $sol(\psi)$, is composed of all its possible solutions, if the CSP has at least one solution it is satisfiable; *i.e.* $sat(\psi) \Leftrightarrow sol(\psi) \neq \oslash$.

Mapping a GD* into CSPs. The mapping ($\mu^{GD} : \rho \rightarrow \psi$) of a WS-Governance* GD (ρ) to a CSP (ψ) is performed in two steps as follows:

First, for each policy $p_i = (s, a)$ in the GD*, p_i is mapped for the concrete governance scope (SOA model in terms of services, applications, organizations and its relationships) into a constraint p_i^C that contains only variables and literals composed using logical and algebraic operators. This constraint is constructed so that it tells exactly when the policy holds in the given governance scope. This transformation is performed by the explicit enumeration of the sets and relationships (ownership, provision, consumption, and organizational hierarchy) on the governance scope for each quantifier, combining the resulting constraints using logical AND ($\wedge$) operators for universal quantifiers and logical OR ($\vee$) operators for existential quantifiers.

Next, the set of constraints $p^C = \{p_i^C\}_{i=1}^n$ and original GD* ρ are mapped into a CSP $\psi = \{V, D, C\}$ by creating:

- a variable $v^x_{s_i|o_i|a_i}$ in V for each property x and corresponding element in the SOA (service, organization and application).
- variables $v^{supOrg}_{o_i}$,$v^{prov}_{s_i}$,$v^{cons}_{s_i}$ and $v^{own}_{a_i}$ in V for the relation functions $supOrg$ (hierarchical relationship among organizations), $provider$ and $consumer$ (relationship among services and applications) and $owner$. Additionally, their domain of organizations, applications, and services are created.
- constraints $\{v^{supOrg}_{o_i} = o_j\}$, $\{v^{prov}_{s_i} = a_k\}$, $\{v^{cons}_{s_i} = a_k\}$, and $\{v^{own}_{a_i} = o_l\}$ in C for each variable created in the previous step specifying the values of the relationships$supOrg$, $provider$,$consumer$ and $owner$. Those constraints and variables express the SADL SOA structural model of the governance scope.
- a constraint $\{v^x_{s_i|o_i|a_i} = \{\textit{Value Expr?}\}\}$ in C for each property valuation specified in the state section of the governance scope in ρ.

Finally, for each constraint in p^C property invocation functions $X(s_i|o_j|a_j)$ are exchanged by their corresponding variables $v^x_{s_i|o_i|a_i}$, and the resulting constraint is added to C. The mapping of different elements of a GD* is shown in Table 2.

Readers interested in an example of the use of GD*, can find the expression of the policies contained in the governance documents found in the case study (Figure 1) in table 4 of [18]. In the right side of that table the CSP transformation of the GD* is also presented.

5.1 Checking for Consistency

Checking a GD ρ written in WS-Governance* for consistency lets us know whether it has internal contradictions or not. The root of the inconsistencies can be: (i) that a policy in ρ is intrinsically inconsistent; (ii) that the set of policies in ρ are inconsistent; or (iii) even when the set of policies in ρ, $P^\rho = \{\rho_1, \ldots, \rho_n\}$, are initially consistent, then ρ can be inconsistent due to the additional information added by the SADL SOA state and structure specified. For instance, the GD* shown in the first column of table 4 of [18] is inconsistent. The cause of the inconsistency is that policies ρ_2='*Critical services availability should be "'24x7'*' and ρ_3='*Service* s_1 *availability is "window" if it is consumed by external applications*' are inconsistent, since s_1 is both consumed by application a_2 'Zeus' of department 2 and critical. This information, service consumption and criticality of services is specified by the SOA Structure and State section of the GDs*,

Table 2. Mapping of WS-Governance* elements onto CSPs

WS-Governance* Element	CSP Mapping
Governance Document *- Name (Id)*	GD Name, Id and Governor Org.
Governor: *Org. Name?(Org. Id?)*	are not mapped to CSP
SOA Governance Model:	For each organization o_i a variable $v_{o_i}^{suporg}$ is created
STRUCTURE:	denoting the parent org. of o_i and a domain
{**Organization:** *Org. Name?* (*Org. Id?*)	$d_i^{suporg} = \{o_1, \ldots \emptyset_n\}$ is added to D
SubOrganizations: *Org. Id1?,. . . ,Org. IdN?*	a constraint $\{C_{o_i}^{prov} = o_j\}$ is created encoding the orgs. hierarchy
Applications:	For each application a_j a variable $v_{a_j}^{own}$ is created
Provides:	denoting the owner org. of a_j, a domain
{Service: *Serv. Name?* (*Serv. Id?*)}*	$d_j^{own} = \{o_1, \ldots \emptyset_n\}$ is added to D
Consumes:	and a constraint $\{C_{a_i}^{own} = o_j\}$ is created encoding the app. ownership
{Service: *Serv. Name?* (*Serv. Id?*)}*	For each service s_k a variable $v_{s_k}^{prov}$ is created
}*	, a domain $d_k^{prov} = \{a_1, \ldots, a_m\}$ is added to D
}+	and a constraint $\{v_{s_k}^{prov} = a_l\}$ is created encoding the provisioning
STATE:	
{*Property val. Expr?*}*	Add Constraint: $v_{s_i\|o_i\|a_i}^{x}$ [*Proverty val. Expr.?*]
Vocabulary:	Add variables & domains:
Property: *X* **for** *Services*	for each property x we create a variable
for $\{Servs\|Orgs\|Apps\}$	$v_{s_i\|o_i\|a_i}^{x}$ for each element in
Type: *boolean*	its corresponding set $S\|O\|A$
Domain: *Domain Expr.?*	Add Constraint: $v_{s_i\|o_i\|a_i}^{x}$ [*Domain Expr.?*]
Policies:	Add contraint:
Policy *P1*	
{**forall** *y*	$\bigwedge_{v_{s_i\|o_i\|a_i}}^{S\|O\|A}$ (*Scope expr*)$[y/v_{s_i\|o_i\|a_i}^{y}]$
in $(Servs\|Orgs\|Apps)$}+	$\Rightarrow$ (*Assertion expr*)$[y/v_{s_i\|o_i\|a_i}^{y}]$
{**exists** *y*	$\bigvee_{v_{s_i\|o_i\|a_i}}^{S\|O\|A}$ ((*Scope expr*)$[y/v_{s_i\|o_i\|a_i}^{y}]$
in $(Servs\|Orgs\|Apps)$}+	$\Rightarrow$ (*Assertion expr*)$[y/v_{s_i\|o_i\|a_i}^{y}]$
Scope: *Scope expr?* ,	For each constraint $c \in C$ do:
Assertion: *Assertion expr* —	$(c)[v_{s_i\|o_i\|a_i}^{y}/v_{s_i\|o_i\|a_i}^{x}]$
where E[x/y] means: ' the expression E, but with occurrences of x replaced by y'	

thus the corresponding constraint $Availability(s_1) =' 24x7' \wedge Availability(s_1) =' window'$ is obviously unsatisfiable due to the SOA state and structure, not because of an incosistency of policies *per se*, and consequently ρ_2 and ρ_3 are inconsistent in ρ.

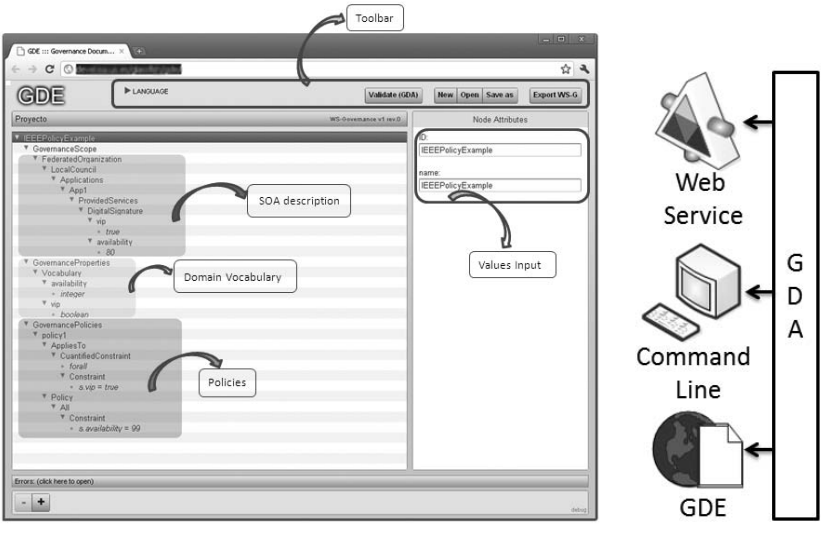
Internal Consistency: A GD* ρ is said to be consistent iff its corresponding equivalent CSP is satisfiable; *i.e.* $consistent(\rho) \Leftrightarrow sat(\psi^{\rho})$.

Consistency: A non empty set of GDs in WS-Governance* $P = \{\rho_1, \ldots, \rho_n\}$ is said to be consistent iff its corresponding equivalent CSPs are simultaneously satisfiable; *i.e.* $consistent(P) \Leftrightarrow sat(\bigwedge_{i=1}^{n} \psi^{\rho}{}_i)$.

As an example, our approach found an additional inconsistence: the mapping of the GD* that models our case study (table 4 of [18]), the corresponding CSP ψ contains among others the following constraints: $\{v_{s1}^{crit} = true \Rightarrow v_{s_1}^{provO} = v_{s_1}^{consO}\}$, $\{v_{s2}^{crit} = true\}$, $\{v_{s_1}^{consO} = o_2\}$, and $\{v_{s1}^{provO} = o1\}$. This set of constraints in unsatisfiable, since $v_{s_1}^{prov} = o_2$ and $v_{s_2}^{prov} = o_1$ are unsatisfiable, and consequently the GD* is inconsistent.

5.2 WS-Governance Tooling: GDA and GDE

Governance Document Analyzer (GDA) is an automatic analyzer for GDs. It performs consistency analysis in three levels: i) individual policy consistency, ii) consistency of the set of policies in the document, and iii) consistency of the whole GD. This analysis starts with the transformation of GD statements to a CSP, that is solved with a CSP solver [20]. The tool is available as a web service so it can be easily integrated into other tools and interoperate. Furthermore, a command line client is provided for direct use. A set of sample GDs has been created, in order to ensure that the analyzer works properly. Both the analyzer and the samples are available at `http://www.isa.us.es/gda`.



(a) GDE User Interface (b) GDA Depl. Opt.

Fig. 6. GDE User Interface and GDA Deployment Options

Governance Document Editor (GDE) is a web application to edit government documents in an assisted way. The editor window is divided in three areas as in Figure 1 (a). On the top section, there are different options to open, save and analyze GDs. On the left side of the main area, every section in the GD is organized in a tree view. Every node on the tree is attributed on the right-hand side. The main tree has separate sections for each subsection in the GD. WS-Governance elements are automatically generated from the tree, so edition focuses on the relevant government contents avoiding errors. It has been integrated with GDA, showing the consistency analysis reports graphically. The editor is available on-line at `http://labs.isa.us.es/apps/gde/`.

6 Related Work

Concerning policy definition, Ponder is the pioneer and probably most widely used language. Ponder [21] is a declarative, object oriented language for the specification

of management polices in distributed object systems. Additionally, Ponder provides structuring techniques for policy administration in large scenarios and systems. WS-Governance incorporates similar concepts by the explicit declaration of governor and document context, and uses SADL and GAL assertions to model scope. The usage of WS-Policy as the base policy expression construct, and explicit declaration of governance relevant information sources, makes WS-Governance better suited for SOA governance policies declaration. However, an interesting capability supported by Ponder that we plan to add to WS-Governance in the future is the declaration of policy types as templates. Rei and KAoS [22], both based on semantic web concepts are proposals oriented to the definition of policies for expressing web services capabilities policies. However, is WS-Policy the proposal that has more successful in industrial scenarios, thus we have chosen it for extension in order to define SOA governance policies.

Our proposal provides a richer consistency notion allowing the detection of semantical inconsistences in policies with complex interaction as shown in this paper. General policy conflict analysis is not a novel problem, but its application in the context of SOA governance policies in this paper is original. Several approaches have been proposed for policy expression and conflict analysis in the context of network management [23], and security [24], but they are based on Binary Decision Diagrams (BDD), forcing the reasoning with less expressive policies than our CSP based proposal.

7 Conclusions and Future Work

The results obtained during the development of this work provide three important conclusions: i) there exists the need of a language for governance policies specification that integrates governance data sources; enables effective governance through a formal semantics, and supports automated consistency checking; ii) this need is not fulfilled by WS-Policy, that has limitations in this context; and iii) WS-Governance allows the specification of expressive Governance Documents independently of the underlying infrastructure. It is based on the WS-Policy framework, and overcomes its previously identified limitations to achieve an effective governance. The GD semantics and consistency operation paves the way for the creation of a new generation of governance tools. In this scenario governance policies are created in a distributed, independent yet collaborative way, maintaining global consistency. This collaborative process helps governance boards on the creation of the essential normative framework needed for the achievement of the SOA promise.

In this context we identify the following ideas as promising future work to be addressed: i) enacting organizations need extensions of the GD Context to effectively ensure authentication and authorization, in order to avoid tampering when defining policy (currently this extension is supported by the language through the variability extension point but those mechanisms are not specified); (ii) extend WS-Governance to support the definition of policy templates improving reuse and usability; iii) carry out a performance analysis of our implementation in order to study the influences of the SOA model, governance properties and number and complexity of governance policies; and iv) exploit the formal definition of policies as CSPs and the capabilities of solvers for providing inconsistency explaining and support policy enforcement.

References

1. Marks, E.A.: Service-Oriented Architecture Governance for the Services Driven Enterprise. John Wiley & Sons (2008)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Computer 40(11), 38–45 (2007)
3. Schepers, T.G.J., Iacob, M.E., Van Eck, P.A.T.: A lifecycle approach to soa governance. In: SAC 2008: ACM Symposium on Applied Computing, pp. 1055–1061 (2008)
4. Kenney, L.F., Plummer, D.C.: Magic quadrant for integrated soa governance technology sets. Technical report, Gartner (2009), http://mediaproducts.gartner.com/reprints/oracle/article65/article65.html
5. Kontogiannis, K., Lewis, G.A., Smith, D.B.: A research agenda for service-oriented architecture. In: SDSOA 2008: 2nd Int. Workshop on Sys. Devel. in SOA Env., pp. 1–6 (2008)
6. Bernhardt, J., Seese, D.: A Conceptual Framework for the Governance of Service-Oriented Architectures. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 327–338. Springer, Heidelberg (2009)
7. Derler, P., Weinreich, R.: Models and Tools for SOA Governance. In: Draheim, D., Weber, G. (eds.) TEAA 2006. LNCS, vol. 4473, pp. 112–126. Springer, Heidelberg (2007)
8. Parejo, J.A., Fernández, P., Ruiz-Cortés, A.: Towards automated sla-based governance policy enforcement. In: Int. Joint Conference on Service Oriented Computing (ICSOC) (2009)
9. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Ümit Yalçinalp: Web services policy 1.5 framework. W3C Recommendation (2007)
10. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Ümit Yalçinalp: Web services policy 1.5 - attachment. W3C Recommendation (2007)
11. Hollunder, B.: Domain-specific processing of policies or: Ws-policy intersection revisited. In: ICWS, pp. 246–253 (2009)
12. Anderson, A.H.: Domain-independent, composable web services policy assertions. In: POLICY, pp. 149–152 (2006)
13. Ruiz-Cortés, A., Martín-Díaz, O., Durán, A., Toro, M.: Improving the automatic procurement of web services using constraint programming. International Journal of Cooperative Information Systems 14(4), 439–467 (2005)
14. Müller, C., Ruiz-Cortés, A., Resinas, M.: An Initial Approach to Explaining SLA Inconsistencies. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 394–406. Springer, Heidelberg (2008)
15. Müller, C., Resinas, M., Ruiz-Cortés, A.: Explaining the Non-compliance between Templates and Agreement Offers in WS-Agreement. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 237–252. Springer, Heidelberg (2009)
16. Zhang, Y., Liu, X., Wang, W.: Policy lifecycle model for systems management. IT Professional 7, 50–54 (2005)
17. Peterson, D., Gao, S.S., Malhotra, A., Sperberg-McQuee, C.M., Thompson, H.S.: W3c xml schema definition language (xsd) 1.1 part 2: Datatypes. W3C Working Draft (2009)
18. Parejo, J.A., Fernandez, P., Ruiz-Cortés, A.: Ws-governance: A language for soa governance policies definition. Technical report, Applied Software Engineering Research Group (Grupo ISA), University of Seville (2010), http://www.isa.us.es/publications
19. Vallecillo, A.: A journey through the secret life of models, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2008)
20. Laburthe, F., Jussien, N., Rochart, G., Cambazard, H., Prud'homme, C., Malapert, A., Menana, J.: Choco, java library for constraint satisfaction problems (csp), Open Source http://www.emn.fr/z-info/choco-solver/

21. 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)
22. Uszok, A., Bradshaw, J., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S.: Kaos policy management for semantic web services. IEEE Intelligent Systems 19(4), 32–41 (2004)
23. Samak, T., Al-Shaer, E., Li, H.: Qos policy modeling and conflict analysis. In: POLICY, pp. 19–26 (2008)
24. Hamed, H.H., Al-Shaer, E.S., Marrero, W.: Modeling and verification of ipsec and vpn security policies. In: ICNP, pp. 259–278 (2005)

QoS-Based Task Scheduling in Crowdsourcing Environments*

Roman Khazankin, Harald Psaier, Daniel Schall, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology,
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at

Abstract. Crowdsourcing has emerged as an important paradigm in human-problem solving techniques on the Web. One application of crowdsourcing is to outsource certain tasks to the crowd that are difficult to implement as solutions based on software services only. Another benefit of crowdsourcing is the on-demand allocation of a flexible workforce. Businesses may outsource certain tasks to the crowd based on workload variations. The paper addresses the monitoring of crowd members' characteristics and the effective use of monitored data to improve the quality of work. Here we propose the extensions of standards such as Web Service Level Agreement (WSLA) to settle quality guarantees between crowd consumers and the crowdsourcing platform. Based on negotiated agreements, we provide a skill-based crowd scheduling algorithm. We evaluate our approach through simulations.

Keywords: crowdsourcing, skill monitoring, scheduling, QoS agreements.

1 Introduction

Recently, business processes need to be adapted or extended more frequently to the changing market. Companies often lack the new capabilities or knowledge required. To tackle those changes, either new personal needs to be hired, or rather, the new process steps are outsourced. The work in this paper is based around a recent and attractive type of outsourcing called *crowdsourcing* [11]. The term crowdsourcing describes a new web-based business model that harnesses the creative solutions of a distributed network of individuals [4], [21]. This network of humans is typically an open Internet-based platform that follows the *open world* assumption and tries to attract members with different knowledge and interests. Large IT companies such as Amazon, LiveOps, or Yahoo! ([3], [17], [22]) have recognized the opportunities behind such *mass collaboration systems* [8] for both

* This work was supported by the European Union FP7 projects COIN (No. 216256) and SCube (No. 215483) and the Vienna Science and Technology Fund (WWTF), project ICT08-032.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 297–311, 2011.

improving their own services and as a business case. The most prominent platform they currently offer is the *Amazon Mechanical Turk* (AMT) [3]. Requesters are invited to issue *human-intelligence tasks* (HITs) requiring a certain qualification to the AMT. The registered customers post mostly tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [12]).

In this paper we extend this simple model and focus on crowdsourcing platforms that deal with task groups consisting of manifold similar jobs provided by consumers. Most current public crowdsourcing platforms with market-like operation chain announce received tasks at their portal as a first step. Next, the worker chooses among the assorted mass of task those s/he likes to process. The selection is motivated by her/his personal preferences. Thus, the following assignment is initiated by the worker, and as a consequence, it hardly allows the system to have an influence upon assignments and to leverage the skill heterogeneity of involved workers.

As each worker is individual, it is natural that the skills of the workers are manifold. The tasks submitted to the platform are also diverse in their requirements. Hence, efficient crowdsourcing must consider the suitability of a worker for a task. One can assume that the more the worker is suitable for a task, the better the expected outcome quality is. Therefore, given the suitability information and a control mechanism for a worker assignment, it is possible to improve the overall result's quality by assigning tasks to best suitable workers for the current situation. Moreover, from a BPM perspective, this mechanism also provides control over task completion times which can be used to maintain objectives, such as deadline fulfillment. To sum up, the task assignment control would enable a crowdsourcing platform provider to develop QoS policies for offered crowdsourcing services, so these services can be integrated into QoS-sensitive business processes. The assignment control demands for a scheduling problem to be solved. The problem is to maximize overall quality while satisfying agreed objectives. Such a scheduling problem is hindered by a number of crowd-specific features, such as lack of full control of the workers and their membership, and their limited predictable availability.

In this paper we tackle these issues by proposing crowdsourcing platform enhancements. Hence, our key contributions are:

- A crowdsourcing platform model which allows for agreement-aware task processing.
- Algorithms for task scheduling and workers' skill profile updates.
- Proof-of-concept implementation and evaluation of the approach in a simulated environment.

The paper is organized as follows. In Sect. 2 we list work related to crowdsourcing and scheduling. Section 3 outlines our platform model that provides agreement-based task assignments to a crowd and also gives insights in the application of current crowdsourcing platforms. Section 4 details the proposed crowdsourcing model. A prototype of the platform is evaluated in Sect. 5. Section 6 concludes the work.

2 Related Work

In this work we position crowdsourcing in a service-oriented business setting by providing automation. In crowdsourcing environments, people offer their skills and capabilities in a service-oriented manner. Major industry players have been working towards standardized protocols and languages for interfacing with people in SOA. Specifications such as WS-HumanTask [10] and BPEL4People [2] have been defined to address the lack of human interactions in service-oriented businesses [16]. These standards, however, have been designed to model interactions in closed enterprise environments where people have predefined, mostly static, roles and responsibilities. Here we address the service-oriented integration of human capabilities situated in a much more dynamic environment where the availability of people is under constant flux and change [6]. The recent trend towards *collective intelligence* and crowdsourcing can be observed by looking at the success of various Web-based platforms that have attracted a huge number of users. Well known representatives of crowdsourcing platforms are the aforementioned form Yahoo!, LiveOps, and Amazon. The difference between these platforms lies in how the labor of the crowd is used.

Crowdsourcing. AMT, for example, offers access to a large number of crowd workers. With their notion of HITs that can be created using a Web service-based interface they are closely related to our aim of mediating the capabilities of crowds to service-oriented business environments. Despite the fact that AMT offers HITs on various topics [12], the major challenges are to find on request skilled workers that are able to provide high quality results for a particular topic (e.g., see [1]), to avoid spamming, and to recognize low-performers. To the best of our knowledge, these problems are still not faced by AMT. In this work we focus on those issues.

Another shortcoming of most existing real platforms is the lack of different and comprehensive *skill information*. Most platforms have a simple measure to prevent workers (in AMT, a threshold of task success rate can be defined) from claiming tasks. In [19], the automated calculation of expertise profiles and skills based on interactions in collaborative networks was discussed.

In [13], a quality management approach for crowdsourcing environments is presented. Unlike our profile management, this work doesn't support multiple skills, but concentrates on a single correctness dimension. On the other hand, if there is a specific need for such a quality management technique, the profile management can thus be replaced with it by correllating the correctness and suitability, as this module is decoupled from the rest of the platform as mentioned in Sec. 3.

Scheduling is a well-known subject in computer science. The novel contribution in this work is to consider multidimensional assignment and allocation of tasks. A thorough analysis and investigation in the area of multidimensional optimal auctions and the design of optimal scoring rules has been done by [7]. In [18] iterative multi-attribute procurement auctions are introduced while focusing on

mechanism design issues and on solving the multi-attribute allocation problem. Focusing on task-based adaptation, [20] near-optimal resource allocations and reallocations of human tasks were presented. Staff scheduling related to closed systems was discussed in [5,9]. However, unlike in closed enterprise systems, crucial scheduling information, i.e., the current user load or precise working hours are usually not directly provided by the crowd. Instead, the scheduling relevant information must be gathered by monitoring. The work in [15] details the challenges for collaborative workforce in crowdsourcing where activities are coordinated, workforce contributions are not wasted, and results are guaranteed.

3 Crowdsourcing Platform Model

In this section a model of a crowdsourcing platform (see Fig. 1) which supports agreement-based task assignment is outlined. To begin with, there is a negotiation that states the objectives between the consumer and the platform regarding the coming task assignments. The life-cycle of a task begins at the consumer. S/He submits a collection of tasks to the crowd. We assume that the tasks have distinct skill requirements, thus, need to be assigned accordingly. After processing, the result is returned to the consumer which is invited to provide a quality feedback on the result.

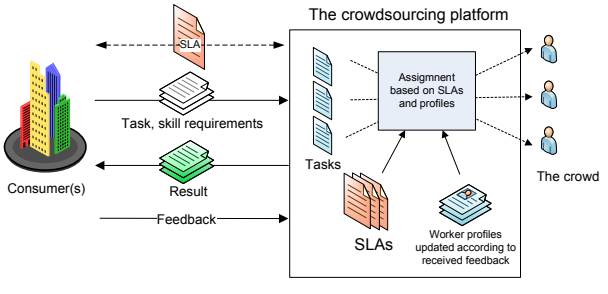


Fig. 1. Platform model

Once a consumer submits a task, s/he provides skill requirements along with the task. In the assignment phase the platform estimates the matching between tasks and workers. The more a worker's skills match the requirements the more this worker is suitable for the task. Also, in service-oriented environments a *Service Level Agreement (SLA)* usually is negotiated which has influence on the assignment strategy. An SLA includes temporal and quality requirements, and preferences. The challenge for the platform is to negotiate the *Service Level Objectives (SLOs)* of an SLA related to possible assignments. These SLOs must base on the observed and predicted behavior of the crowd and the current situation. In particular, already distributed tasks and the resulting task load must be considered.

Therefore, active tasks are assigned not only according to suitability of workers, but also, in line with the SLOs of the agreement. The agreement aims to

avoid or minimize the losses and to maximize the overall result's quality. In other words, the objective is to maximize the quality while enforcing the SLA. Also, the availability of workers is taken into consideration by requesting short-term information regarding their ability to perform tasks.

The suitability is calculated as a match between required skills for the task and the skills of a worker. The skills of crowd workers are maintained in their profiles by the platform management. Initially, skill information is provided by the workers themselves.

However, this information must not be considered complete and reliable. Some workers might not know about their real skill-levels or overestimate their capabilities. Hence, the platform management must be allowed to monitor the activities in the crowd and update the created profiles according to the observations. The resulting real quality is reported by the consumer's feedback. The difference between expected quality and required skills for the tasks hint the real skills of the workers. Also, if an assignment is refused by a worker, despite his claim for availability, various penalty sanctions can be imposed to this worker.

The rules for calculating the suitability are independent of the scheduling logic. Thus, the suitability is represented by a single real value in $[0, 1]$ (0 - not suitable at all, 1 - perfectly suitable) which summarizes the expectations regarding the quality of the result. This enables the technique, which is used to calculate the suitability, to be completely decoupled from scheduling.

3.1 Integration of Service Level Agreements (SLAs)

Assignment control enables the platform to estimate the crowd occupancy and to give certain guarantees to consumers. Such guarantees can be given in form of SLAs, and allow to integrate crowdsourcing activities in service-oriented environments. As SLAs are crucial for business process management, such a model can substantially sustain the use of crowdsourced services in business processes.

Next, we discuss an outline of a WSLA[1] document that could be exchanged and agreed between consumer and crowd platform.

After the contract parties' details, *SchedulingPlatform* and *Consumer* listed in lines 4 to 11, Listing 1.1 states the contract items from line 12 to 20. These are a collection of *ServiceObjectType* items including scheduling, operation description, and configuration, and also, the *SLAParameter*s. Here, as an example the parameter (`TaskSkills`) uses a crowd particular metric `CompareSkills` to compare skill profiles of the workers to the skill required in the document. Important to note, we extend the WSLA definition with our own namespace (task scheduling platform `tsp`) defining xpath methods for expressions and type definitions to comply with all requirements of the platform.

Listing 1.2 shows the agreement's terms as *Obligations* of the contract including some SLOs. An SLO consists of an obliged party, a validity period, and an *Expression*s that can be combined with a logic expression (e.g., *Or*). The *Value* tag in the predicates of WSLA is restricted to double. Hence, another extension

[1] http://www.research.ibm.com/wsla/

```
1  <wsla:SLA
2  xmlns:wsla="http://www.ibm.com/wsla" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xmlns:tsp="http://www.infosys.tuwien.ac.at/tsp/" name="SLA4711">
4   <wsla:Parties>
5    <wsla:ServiceProvider name="SchedulingPlatform">
6     <!-- details -->
7    </wsla:ServiceProvider>
8    <wsla:ServiceConsumer name="Consumer">
9     <!-- details -->
10   </wsla:ServiceConsumer>
11  </wsla:Parties>
12  <wsla:ServiceDefinition name="CrowdService">
13   <wsla:Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType" name="ScheduleTask">
14    <!-- schedule period -->
15    <wsla:SLAParameter name="TaskSkills" type="tsp:SkillList" unit="double">
16     <wsla:Metric>CompareSkills</wsla:Metric>
17    </wsla:SLAParameter>
18    <!-- SLAParameter metrics for NrOfTasks, TaskQuality, Fee. Details: name, wsdl-location, binding -->
19   </wsla:Operation>
20  </wsla:ServiceDefinition>
21  <!-- wsla Obligations -->
22 </wsla:SLA>
```

Listing 1.1. Involved parties and body

with xpath methods allows us to provide more complex expressions for the values (e.g., see Line 8). Generally, in an SLO an evaluation event defines the trigger for the evaluation of the metric function. The content of the expressions connects the pool of *SLAParameters* of the items to a predicate (e.g, `GreaterEqual`) and threshold value (*Value*). In the example we define three objectives. The first, `sloSkills` defines that the match between the task skill requirement `skills required` and the selected potential worker's skills must be greater or equal. The second, `sloQuality` is a composed objective by an *Or* expression. The agreement states, that either a defined number of tasks (`100`) is delivered at the end of an agreed interval (e.g., *wsdl:Months*) or the focus of processing is on the task's quality, and hence, a result quality of at least 80% is expected. The final objective `sloFee` obliges the consumer to confirm the quality of the result and pay the related fee. In the example, similar to Line 8, the xpath method `getInput` parses a document `TaskDesc` and returns the task's fee. As the fee depends also on the result, the result report `TaskRes` contains the reported quality. Multiplied they give the final fee due.

3.2 Discussion

Generally, in our model, workers are more restricted than in market-oriented crowdsourcing platforms. Still, such an approach provides adequate flexibility for workers by allowing them to choose the time periods in which they are willing to work. Thus, assuming a strong competition among workers, the model will be feasible. Constantly increasing interest in crowdsourcing platforms [8] indicates that such competition is quite realistic. Also, SLA-enabled services are higher valued, therefore, the monetary compensation, and, thus, the competition can be higher. The feedback provision from the consumer is of his/her own interest,

```
<wsla:Obligations>
 <wsla:ServiceLevelObjective name="sloSkills" serviceObject="ScheduleTask">
  <wsla:Obliged>SchedulingPlatform</wsla:Obliged>
  <!-- Validity -->
   <wsla:Expression> <!-- skill requirements -->
    <wsla:Predicate xsi:type="wsla:GreaterEqual">
     <wsla:SLAParameter>TaskSkills</wsla:SLAParameter>
     <tsp:Value>tsp:getInput("TaskDesc")//TaskDefinition/skills_required</tsp:Value>
    </wsla:Predicate>
   </wsla:Expression>
  <wsla:EvaluationEvent>TaskAssignment</wsla:EvaluationEvent>
 </wsla:ServiceLevelObjective>
 <wsla:ServiceLevelObjective name="sloQuality" serviceObject="ScheduleTask">
  <wsla:Obliged>SchedulingPlatform</wsla:Obliged>
  <!-- Validity -->
  <wsla:Or>
  <wsla:Expression>
   <wsla:Predicate xsi:type="wsla:Equal">
    <wsla:SLAParameter>NrOfTasks</wsla:SLAParameter>
    <wsla:Value>100.0</wsla:Value>
   </wsla:Predicate>
  </wsla:Expression>
  <wsla:Expression>
   <wsla:Predicate xsi:type="wsla:GreaterEqual">
    <wsla:SLAParameter>TaskQuality</wsla:SLAParameter>
    <wsla:Value>0.8</wsla:Value> <!-- expected qty 80%-->
   </wsla:Predicate>
  </wsla:Expression>
  </wsla:Or>
  <wsla:EvaluationEvent>TaskResult</wsla:EvaluationEvent>
 </wsla:ServiceLevelObjective>
 <wsla:ServiceLevelObjective name="sloFee" serviceObject="ScheduleTask">
  <wsla:Expression>
  <wsla:Obliged>Customer</wsla:Obliged>
   <wsla:Predicate xsi:type="wsla:Equal">
    <wsla:SLAParameter>Fee</wsla:SLAParameter>
    <tsp:Value>
     <![CDATA[tsp:getInput("TaskDesc")//Fee * tsp:getInput("TaskRes")//Quality]]>
        </tsp:Value>
   </wsla:Predicate>
  </wsla:Expression>
  <wsla:EvaluationEvent>TaskResult</wsla:EvaluationEvent>
 </wsla:ServiceLevelObjective>
       <!-- agreed qualified actions on slo violations -->
</wsla:Obligations>
```

Listing 1.2. Obligations and SLOs

as it positively affects the result quality. It can be provided not for all results but selectively.

In this paper we don't discuss the cost of the work and payments in detail. Although it is an important factor, in our vision, it can be seamlessy integrated into the platform. One design solution could be that the workers specify the minimal cost for their work and the consumers would pay as much as they want as in a traditional crowdsourcing platform. Thus, the more the customer is willing to pay, the more workers would be considered for assignment, and, as it is sensibly to assume, more suitable workers could be found. However, such a design will not change the basics and the algorithms of our platform substantially. Thus, for the sake of simplicity, we assume that all the jobs cost correspondingly to their specified duration.

4 Quality and Skill-Aware Crowdsourcing

This section explains in detail the quality and skill-aware crowdsourcing platform architecture, which is a proof-of-concept implementation of the model described in Sect. 3.

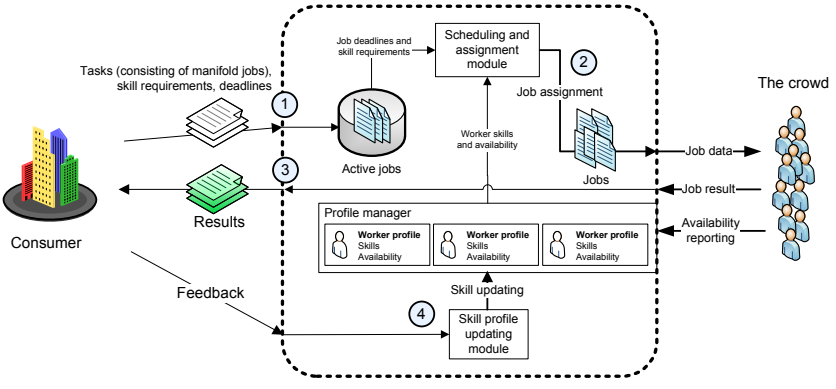


Fig. 2. Platform architecture

The platform behavior can be summarized by the following steps (in the line with the ones in Fig. 2):

1. A consumer submits a task that consists of manifold similar jobs. The consumer also specifies required skills and the deadline for the task, so all the jobs should be processed until this deadline. The task is added to active task pool.
2. The scheduling module periodically assigns active jobs to workers according to deadlines and suitability of workers.
3. When a job is done, the result is sent to the consumer.
4. The consumer can provide a feedback about the result quality. It is reported to the skill profile updating module, which matches it with the expected quality and corrects the worker skill profile if necessary.

Of course, the platform is intended for usage by multiple consumers, this fact is not depicted for simplicity. A deterministic time model is used in the platform, so the time is discreet and is represented by sequential equally long time periods. A time period can represent any time duration like a minute, an hour, or a day.

4.1 Skills and Suitability

The system distinguishes a fixed set of skills. Each worker has a skill profile, where each skill is described by a real number $s \in [0, 1]$ which defines it quantitatively (0 - lack of skill, 1 - perfect).

Each submitted task has the required skills specified. Each required skill is also represented as a real number $r \in [0, 1]$. If $r = 0$ then the quality does not depend on this skill. If $r > 0$ then the best outcome quality is expected in case if the corresponding worker's skill s is $s >= r$. If $s < r$ then the expected quality is affected in the inverse proportion to $r - s$. The quality is again represented as a real number $q \in [0, 1]$. The suitability of a worker for a task is equal to the expected outcome quality. The exact matching formula is shown below.

Let WS_i - worker skills, RS_i - required skills of a task, $i = \overline{1, N}, N-$ number of skills. Then the suitability of the worker to the task is calculated as

$$S = 1 - \sum_{i \in M} \frac{Max((RS_i - WS_i)/RS_i, 0)}{|M|} \qquad M : k \in M \Leftrightarrow k \in N, RS_k > 0$$

Thus, the more worker's skills are proportionally closer to the required skills of a task, the more the worker is suitable to the task. If the worker's skill is equal or greater than the corresponding required skill, then this skill suits perfectly.

4.2 Worker and Consumer Communication

As the user interface is not the focus of our work, the communication with consumers and workers is performed by means of web services. The full implementation of the platform can employ, e.g., a web interface developed on top of these web services.

4.3 Scheduling

Scheduling is performed based on deadlines and skill requirements also derived from an SLA. The objective of the scheduling module is to maximize the overall quality, while fulfilling deadlines. The assumption is that missing a deadline can not be justified by any quality gain, thus, meeting a deadline is the first-priority objective, and the quality maximization is the second-priority objective.

Algorithm 1 describes a scheduling algorithm which is used in our platform. The idea behind the algorithm is that the best quality is achieved when a task is assigned to most suitable workers. The quality is higher when a task is performed by a smaller number of best workers, but this number should not be too small, so the task can be finished until the deadline. This number is calculated in $toTake$ for each active task. The tasks with earlier deadlines are assigned in the first place. In an attempt to improve the algorithm's efficiency, we tried a number of heuristic extensions, such as:

- Based on reported short-time worker availability, assign less jobs at a given time to wait for more suitable workers to become available (while avoiding possible crowd "overloads")
- Assign more jobs at a given time if the suitability of additional workers is almost as good as the suitability of best workers.
- Having $toTake$ numbers calculated, optimize the worker-task assignments for each time period using an optimization framework.

Algorithm 1. Greedy scheduling algorithm.

Require: $currentTime$ current time
Require: $tasks$ active tasks
1: **for** $task \in tasks$ in the order of ascending $task.deadline$ **do**
2: $stepsToDeadline = (task.deadline - currentTime+1)$ / $task.duration$ - 1
3: **if** $stepsToDeadline > 0$ **then**
4: **if** $(task.deadline - currentTime + 1)$ % $task.duration) > 0$ **then**
5: $toTake = 0$
6: **else**
7: $toTake = Trunc(task.numberOfJobsToDo/stepsToDeadline)$
8: **end if**
9: **else**
10: $toTake = task.numberOfJobsToDo$
11: **end if**
12: **while** $toTake > 0$ AND some workers are still available **do**
13: Assign a job of $task$ to most suitable available worker for $task$
14: $toTake = toTake - 1$
15: **end while**
16: **end for**

However, as shown in Sect. 5, such extensions do not give a substantial improvement. We believe that the reason of such a weak improvement is the size of the crowd: if a worker cannot be assigned to a due task, in most of the cases a good enough replacement for the worker can be found. Thus, we conclude that a greedy algorithm is generally sufficient for scheduling in a crowdsourcing environment. The refinement of the algorithm can be done according to the particular crowd characteristics that can be estimated only when the system is used by real users in the commercial operation.

4.4 Profile Management

The crowd workers' profile management is of major importance in our assumptions. As mentioned before, the success of the scheduling algorithm partially depends on the accuracy of the profile monitoring. At the beginning of her/his membership at the crowdsourcing platform a user registers with a profile representing the skills. Usually this information is not very accurate because users tend to over-/underestimate their skills. Hence at runtime, a monitoring module must run on-line and manage the profiles by updating the provided information. It is necessary to avoid conflicts with the promised quality agreements (c.f., Sect. 3). This is a major challenge. The task processing results and the expected quality outcome must be used as a reference for the real skills of a worker. The quality expectations on the tasks result are often detailed in the task description. At the AMT, for example, the result feedback contains usually only a task accept or reject. At our platform, with an agreement requiring the customer to give a feedback on the quality, the feedback contains crucial information for the Algorithm 2 that can be used to update the skills of the reported worker profiles.

As the scheduler requires skill knowledge the profile update is twofold. If the worker only provided a low quality the update depends on the difference (Lines

Algorithm 2. Profile monitoring.

Require: QF quality feedback of the provider, QE quality expected by the provider
Require: $worker$ processing worker and $taskSkills$ required task skills
1: $workerSkills \leftarrow worker.getSkills()$
2: **if** $QE > \vartheta_q$ /* high quality result */ **then**
3: /* compare with latest history entry, update and continue on better QF */
4: $entry \leftarrow setHistory(QF, taskSkills)$
5: **for** skill $s \in workerSkills$ **do**
6: $reqSkill \leftarrow getTaskSkills(s)$
7: $diff \leftarrow |s - reqSkill| \times \alpha_w$
8: **if** $s > reqSkill$ **then**
9: $workerSkills.set(s + diff)$
10: **else**
11: $workerSkills.set(s - diff)$
12: **end if**
13: **end for**
14: **return**
15: **end if**
16: /* low quality result */
17: $wprofile \leftarrow setOfProfiles.get(worker)$ /* set of registered profiles */
18: $diff \leftarrow QF/QE$ /* difference between the qualities */
19: **for** skill $s \in workerSkills$ **do**
20: /* skill == 1 perfect knowledge */
21: **if** $skill \times diff <= 1$ **then**
22: $workerSkills.set(s \times diff)$
23: **end if**
24: **end for**

17-24). If the quality is above a certain threshold ϑ_q and is better than a previous then we consider the required skills close to the workers own (Line 3-14). Hence, the difference between the required and the worker's own skills (weighed by the factor α_w) influence the worker's skill update.

5 Experiments

To evaluate our platform, we set up a simulated environment that comprises a crowd which perform tasks and consumers who submit tasks and provide the feedback. We assigned a real skill set for each simulated worker to calculate the real quality outcome (which consumers report) using the suitability formula (see Sect. 4.1). The platform management had no access to the real skills, but was only able to estimate workers' skills based on the feedback provided by consumers. We evaluated the average job quality in different setups, varying the workload of the crowd, the availability of workers, the scheduling algorithms, and the skill awareness.

Simulation of a real crowdsourcing environment is challenging due to the lack of comprehensive statistical data in this area. Although we don't rely on any real data in our simulations, we tried our best to prognosticate the meaningful simulation parameters based on our knowledge and experience.

5.1 Experiment Setup

In our experiments we use a set of 10 skills for describing worker skills and task skill requirements.

Customers. The customers submit tasks to the platform and provide the feedback on completed jobs. Tasks are submitted randomly while ensuring the average crowd workload and avoiding overloads.

Each task comprises skill requirements, number of jobs, and deadline. During each time period of the simulation, if the *Task Limit* has not been reached yet, a new task is submitted to the system with *Task Concentration* probability. The job duration is calculated as $Min(1 + abs(\phi/2 * \sigma), \sigma + 1)$, where ϕ is a normally distributed random value with mean 0 and standard deviation 1. The deadline is assigned randomly according to *Steps To Deadline* parameter. The number of jobs is calculated so that the crowd workload is near equally distributed among the tasks, and the average workload remains close to *Intended Schedule Density*. The parameters and their values are described in Table 1.

Table 1. Task generation parameters

Name	Description	Value(s)
Tasks Limit	The total number of submitted tasks	200
Job Duration Sigma (σ)	Describes the deviation and the maximum for job durations	20
Steps To Deadline	Average maximum number of jobs of a task that a single worker can finish until the deadline.	50
Task Concentration	The probability of new task submission for each time period.	0.35
Intended Schedule Density	Target assignment ratio for each time period.	0.2 - 0.7 (step 0.1)

Skill requirements are generated so that each skill with approximately equal probability either equals 0 which means that this skill is not required for the task, or is in $(0, 1]$ range. The random values for the $(0, 1]$ range are normally distributed (mean = 0.4, variance = 0.3).

The feedback that a consumer provides for a job is generated using the real skills of the worker which were assigned for this job. In contrast to the estimated skills, these real skills are unknown to the platform and are only used to simulate the real outcome quality (by calculating the suitability with these skills). This quality is thus reported as the feedback.

Crowd workers. The workers are assigned for jobs and return the result of job processing. Each worker has the claimed skills that s/he initially reports to the platform, and the real skills. The real skills are generated randomly with normal distribution with 0 mean and variance of 0.3. Then, the reported skills are initiated as real skills with injected error (normally distributed with mean value equal to the real skill and variance of 0.2). The crowd size in experiments was 1000 workers. This size is big enough to enclose the diversity of workers, but still allows for fast simulation. We tried to use 10000 instead, but the results did not change substantially. Workers can be unavailable at certain periods.

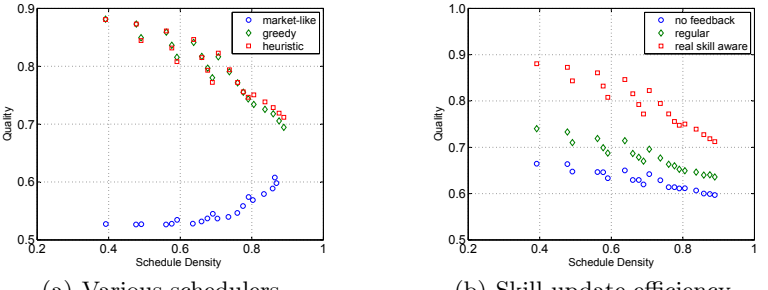
In our experiments we use a *Workers Unavailability* parameter which indicates the mean ratio of unavailable workers for each period of time (values used: $0.2 - 0.6$, step 0.1). The busy periods are generated randomly, but have a continuous form which reproduces human behavior. The amount of time that takes a worker to finish the job is the *Job Duration* with injected variations. In our experiments we used a value of 30%, which means that a job can be executed for $0.7 - 1.3$ of job duration. This reflects the random nature of the real world.

5.2 Experiment Types and Results

The aim of the experiments is to show the advantages of the platform's architecture. The first experiment type demonstrates the convenience of skill-based scheduling to the ordinary (random) task assignment. The second experiment type gives evidence of the skill update mechanism efficiency.

All the plots show the average job outcome quality for the resulting assignment density which is calculated as total number of periods for all workers while they were either unavailable or busy, divided by the difference between the first task submission time and the latest deadline. Apart from where explicitly specified, performed experiments contained no task deadline violations.

Various schedulers. To demonstrate the advantage of skill-based assignment, a scheduler which mimics a market-like platform was compared with the greedy scheduler and the heuristically-enhanced greedy scheduler (See Sect. 4). In market-like scheduling, the assignment followed the logic that randomly chosen workers were picking the most suitable for them active tasks. The results are shown in Fig. 3(a). In tests with high schedule density (about 0.8 or more), market-like assignment performed better than in tests with low density, because workers had more tasks to choose from. However, about 15% of task deadlines were violated in these tests, because workers aimed to fulfill their own preferences rather than the goals of the system. For the rest of the tests, the average quality was 1.5 times better for skill-based scheduling in the large. This clearly shows the benefit of skill-based scheduling. The heuristics did not improve the greedy algorithm substantially, and for some tests even impaired it.



(a) Various schedulers. (b) Skill update efficiency.

Fig. 3. Experimental results

Skill update efficiency. To demonstrate the efficiency of skill update mechanism, we compared the regular simulation which implements the logic described in Sect. 4 ("regular" series) to upper and lower bounds. The series named "no feedback" represents the lower bound and only the initial information on the profiles is used for scheduling. An upper bound to the algorithm is shown by the series of "real skill-aware". In this case the exact skills of a worker are known to the system. The improvement of the skill update mechanism over the lower bound is evident and keeps performing better at any scheduling density. In the experiments of Fig. 3(b) the improvement over no feedback remains between 10-15%. As Sect. 4.4 explains, the reason why it is never reaching real skill awareness is twofold. First, the scheduling strategy need some input right from start when only few feedback is available. Second, the feedback is a single value that describes the performance depending on ten different skills. Also, a skill value greater than required calculates the quality with the lowest value required. Even if there was enough data an accurate calculation would not be feasible in all cases. Thus, we decided to stick to a simpler quicker update algorithm that provides almost constant quality improvement and, after all, supports quality negotiation with a considerable and steady lower bound to make agreements.

The performance of scheduling and skill updating in a high workload test (10000 workers and 1000 tasks) was good enough for a period size of one minute. Thus, the performance is not a concern, since the real period size is likely to be bigger (e.g., 10 - 60 minutes).

6 Conclusion and Future Work

In this paper we proposed a skill-aware crowdsourcing platform model which allows to provide crowdsourcing services with SLAs and to control the task performance quality. In contrast to existing crowdsourcing platforms such as AMT, which follow a task market-oriented approach, our platform model is based on services computing concepts. Such a model is typically applied in enterprise workflow systems using, for example, the WS-HumanTask specification to design human interactions in service-oriented systems. However, WS-HumanTask and related specifications lack the notion of human SLAs and task quality. In our approach, negotiated SLAs and monitoring help to assign task requests to suitable workers. Thus, our platform ensures quality guarantees by selecting skilled workers. We introduced the proof-of-concept implementation with particular algorithms for task scheduling and worker profile management. The applicability of the platform design was proved in a simulated environment. The experimental results shows the clear advantage of skill-based scheduling in crowdsourcing, as the average quality is 50% better in the large comparing to the case when the workers choose tasks by themselves. The skill monitoring and updating mechanism improves the overall quality by 10-15%.

In our future work we will focus on workload and workforce availability predictions, SLA negotiation, and pricing in such scheduled crowdsourcing platforms. Also, we plan to extend the boundaries of our previous work [14] to consider

the scenarios where the SLAs between the crowdsoucing platform and a business process engine can be negotiated beforehand in an autonomic and adaptive fashion.

References

1. Agichtein, E., Castillo, C., Donato, D., Gionis, A., Mishne, G.: Finding high-quality content in social media. In: WSDM 2008, pp. 183–194. ACM (2008)
2. Agrawal, A., et al.: WS-BPEL Extension for People (BPEL4People) (2007)
3. Amazon Mechnical Turk (May 2011), `http://www.mturk.com`
4. Brabham, D.: Crowdsourcing as a model for problem solving: An introduction and cases. Convergence 14(1), 75 (2008)
5. Caprara, A., Monaci, M., Toth, P.: Models and algorithms for a staff scheduling problem. Math. Program. 98(1-3), 445–476 (2003)
6. Castellano, C., Fortunato, S., Loreto, V.: Statistical physics of social dynamics. Reviews of Modern Physics 81(2), 591–646 (2009)
7. Che, Y.: Design competition through multidimensional auctions. The RAND Journal of Economics 24(4), 668–680 (1993)
8. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54, 86–96 (2011)
9. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1), 3–27 (2004), timetabling and Rostering
10. Ford, M., et al.: Web Services Human Task (WS-HumanTask), Version 1.0. (2007)
11. Howe, J.: The rise of crowdsourcing (June 2006), `http://www.wired.com/`
12. Ipeirotis, P.G.: Analyzing the Amazon Mechanical Turk Marketplace. SSRN eLibrary 17(2), 16–21 (2010)
13. Kern, R., Thies, H., Satzger, G.: Statistical Quality Control for Human-Based Electronic Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 243–257. Springer, Heidelberg (2010)
14. Khazankin, R., Schall, D., Dustdar, S.: Adaptive request prioritization in dynamic service-oriented systems. In: Proceedings of the 8th International Conference on Services Computing (to appear, 2011)
15. La Vecchia, G., Cisternino, A.: Collaborative Workforce, Business Process Crowdsourcing as an Alternative of BPO. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 425–430. Springer, Heidelberg (2010)
16. Leymann, F.: Workflow-Based Coordination and Cooperation in a Service World. In: Meersman, R., Tari, Z., et al. (eds.) OTM 2006. LNCS, vol. 4275, pp. 2–16. Springer, Heidelberg (2006)
17. LiveOps (May 2011), `https://www.livework.com/`
18. Parkes, D., Kalagnanam, J.: Models for iterative multiattribute procurement auctions. Management Science 51(3), 435–451 (2005)
19. Schall, D., Dustdar, S.: Dynamic Context-Sensitive Pagerank for Expertise Mining. In: Bolc, L., Makowski, M., Wierzbicki, A. (eds.) SocInfo 2010. LNCS, vol. 6430, pp. 160–175. Springer, Heidelberg (2010)
20. Sousa, J., Poladian, V., Garlan, D., Schmerl, B., Shaw, M.: Task-based adaptation for ubiquitous computing. IEEE Transactions on Systems, Man, and Cybernetics 36(3), 328–340 (2006)
21. Vukovic, M.: Crowdsourcing for Enterprises. In: Proceedings of the 2009 Congress on Services, pp. 686–692. IEEE (2009)
22. Yahoo! Answers (May 2011), `http://answers.yahoo.com/`

Model Driven Security Analysis of IDaaS Protocols

Apurva Kumar

IBM Research – India, 4-Block C, Institutional Area, Vasant Kunj, New Delhi, India 110070
kapurva@in.ibm.com

Abstract. Offloading user management functions like authentication and authorization to identity providers is a key enabler for cloud computing based services. Protocols used to provide identity as a service (IDaaS) are the foundation of security for many business transactions on the web and need to be thoroughly analyzed. While analysis of cryptographic protocols has been an active research area over the past three decades, the techniques have not been adapted to analyze security for complex web interactions. In this paper, we identify gaps in the area and propose means to address them. We extend an important belief logic (the so-called BAN logic) used for analyzing security in authentication protocols to support new concepts that are specific to browser based IDaaS protocols. We also address the problem of automating belief based security analysis through a UML based model driven approach which can be easily integrated with existing software engineering tools. We demonstrate benefits of the extended logic and model driven approach by analyzing two of the most commonly used IDaaS protocols.

Keywords: Security Protocol Analysis, Identity Management, Model Driven Security, Identity as a Service.

1 Introduction

Service providers on the Web are rapidly moving towards a model of focusing on their core business and relying on partners for functions that are needed to support the business. A major enabler for this trend is the existence of protocols that provide user management services over the cloud. We refer to these as IDaaS protocols. While initially limited to providing authentication (single sign-on), user management services are now being used for identity federation, authorization, sharing of user attributes and content. Transactions on the web involving multiple providers often depend on IDaaS protocols for their security needs and thus it becomes important that these protocols be thoroughly examined for the security guarantees they provide.

Important protocols in this space are Security Assertion Markup Language (SAML) [2], OpenID [3] and more recently OAuth [4], but use of proprietary solutions to address specific business to business collaboration requirements is not uncommon. Currently, there is no common set of tools and techniques that can be used to analyze security in these protocols and it requires a lot of skill to perform the analysis. It is often when a protocol starts getting commonly used that it receives enough attention for a thorough analysis, usually once an incident has been reported.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 312–327, 2011.

We advocate need for analysis tools for standards based and custom identity services that can be incorporated in the toolkit of a common solution designer so that the implications of using a security mechanism in a solution are clearly understood.

Analysis of cryptographic protocols has been an active area of research in the past three decades. We identify challenges in extending these approaches for web based IDaaS protocols. A major challenge is the need to support users without identifying keys (public or shared) which is a common situation in web transactions. Another challenge is the need to support transport layer security (e.g. SSL/TLS), which forms part of most web transactions as a primary construct (much like public and shared key encryption in existing techniques). Finally, unlike typical cryptographic protocols, user interactions play an important role in these protocols. Users perform actions like submitting a request, signing in, accepting terms, clicking a link etc. When identities are not global, establishing that a user has previously performed an action is often more important than knowing its identity. Need for representing user actions in security protocols was motivated in [17].

An important set of approaches for security protocol analysis are based on the Burrows, Abadi, Needham (BAN) [1] logic which is used to express and reason about beliefs for secure communication. It defines a logic based on statements about keys, messages, principals and has inference rules which can be used to generate beliefs at participants based on messages exchanged. We extend the belief logic in fundamental ways to support concepts like users communicating over secure channel, authenticating using passwords and performing actions.

We also propose a Unified Modeling Language (UML) based, model driven approach for automating analysis of protocols that are analyzed using belief logics. The approach allows us to analyze security in a wide range of protocols and transactions using domain specific models as input. We show how representation of protocol concepts and properties in the model can be used to simplify modeling and analysis. Using UML as the basis for automating security protocol analysis ensures that our approach can be easily integrated with the software development process. We demonstrate the use of extended belief logic and model driven analysis by analyzing two of the most well-known IDaaS protocols: SAML browser single sign-on protocol and the OAuth [4] protocol.

The rest of the paper is organized as follows: Section 2 covers comparison with related work. Section 3 provides an overview of BAN logic. Section 4 describes the extended logic proposed in this paper along with an analysis of SAML using the logic. Section 5 describes the model driven analysis approach. In Section 6 we describe analysis of the OAuth protocol. In Section 7, we discuss important conclusions from our work.

2 Related Work

Approaches for security protocol analysis can be broadly classified under two categories. *Inference construction* approaches attempt to use inference in specialized logics to establish required beliefs at the protocol participants. The logic of authentication described in [1] was one of the first successful attempts at representing and reasoning about security properties of protocols. In [6], minor improvements to the logic's syntax and inference rules are suggested to remove some ambiguity. Authors

of [7] introduced the concept of 'recognizability'. Logic in [5] introduces the concept of possession along with belief and uses it to support constructs like 'not originated here'. In [8] authors attempt to consolidate good features from earlier belief logic approaches. These logics have the advantage of being usually decidable and efficiently computable. There have been efforts to automate verification for these logics. In [9], a transformation of BAN logic and inference rules to first order formula is performed and theorem prover SETHEO is used for finding proofs. In [10], the authors attempt to embed BAN logic in EVES theorem prover.

Attack construction approaches on the other hand do not try to establish beliefs at the participants but use model-checking techniques to construct attacks. The states and transitions used for modeling the protocol include modeling the structure of the message passing over the channel and a model of the intruder. The intruder is usually based on a Dolev-Yao model [11], and is allowed to perform any sequence of operations such as data interception, concatenation, deconcatenation, encryption, decryption etc. These complexities result in such approaches suffering from state-space explosion problem. Few works that are representative of this class of approaches are mentioned below. The first such approach was introduced in [11] but the class of protocols studied in this work was very limited. In [12] the author has modeled an extension of Dolev-Yao model in a specialized PROLOG based model-checker. Other approaches in this area include the use of FDR model checker for CSP [13] and use of SAT based model-checking techniques to solve a simplified version of the protocol insecurity problem [14]. We note that protocol modeling is generally quite complex in these approaches and even the analysis stage often depends upon user inputs at various stages of the state exploration process.

In our work we prefer to use the inference construction based approach for several reasons. Firstly, attack construction approaches do not actually determine what the protocol achieves but only whether a particular bad state (e.g. a secret being discovered by intruder) is reachable. While this may at times be sufficient for authentication problems, it is hard to specify goals of generic web transactions in this manner. Secondly, the higher abstraction level of inference based approaches allows us to focus on the protocol flow rather than insecurities at the message level. For web transactions, message level security is increasingly being addressed through transport level security. Finally, browser-based protocols are harder to analyze in attack construction based approaches that explicitly model the intruder, since it requires extending the intruder model to incorporate browser-based attacks.

Another relevant work is the analysis of SAML in [15] but the author's do not propose a generic framework for analysis of similar protocols.

3 Overview of Logic of Authentication

BAN is a logic of belief of honest principals participating in a security protocol. Since our approach is an extension of the belief logic described in [1], we provide a brief overview of BAN logic in this section.

BAN statements. A formula in BAN logic is constructed using operators from Table 1. P and Q range over principals. The two statements about keys represent atomic statements. In all other statements X represents a BAN formula constructed using one or more BAN operators.

Table 1. Operators used in BAN logic. X represents a statement in BAN logic.

Notation	Meaning	Notation	Meaning
$P \mid\equiv X$	P believes X	$P \overset{K}{\leftrightarrow} Q$	Shared key K
$P \triangleleft X$	P sees X	$\mapsto_K Q$	Q has public key K
$P \mid\sim X$	P said X	$\{X\}_K$	X encrypted by K
$P \mid\Rightarrow X$	P controls X	$\sharp X$	X is fresh

The expression $\sharp X$ means that the message X is fresh and has not been used before the current run of the protocol. This is especially true for a *nonce*, a sequence number or timestamp generated with this specific purpose. Nonces are used in protocols to defeat replay attacks from previous executions of the protocol.

Inference Rules. There is a set of inference rules for deriving new beliefs from old ones. E.g. the *message-origin* inference rule below states that if P knows that K is a secret key between itself and Q and it sees a message X encrypted by K, then P is entitled to believe that Q said X. A similar inference rule about public keys is also provided where K^{-1} represents the private key corresponding to public key K.

$$\frac{P \mid\equiv Q \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \mid\equiv Q \mid\sim X} \qquad \frac{P \mid\equiv \mapsto_K Q, P \triangleleft \{X\}_{K^{-1}}}{P \mid\equiv Q \mid\sim X} \tag{R1}$$

A *nonce-verification* rule (R2) states that, in addition if the message is known to be fresh, then P believes that Q must still believe X. Further, the *jurisdiction rule* (R3) states that, if in addition, P also believes that Q is an authority on the subject of X (i.e. Q controls X), then P is entitled to believe X himself.

$$\frac{P \mid\equiv Q \mid\sim X, P \mid\equiv \sharp X}{P \mid\equiv Q \mid\equiv X} \tag{R2} \qquad \frac{P \mid\equiv Q \mid\equiv X, P \mid\equiv Q \mid\Rightarrow X}{P \mid\equiv X} \tag{R3}$$

Idealization. Each message exchanged in the protocol is idealized into a BAN formula representing meaning of the message including any facts that the sending of the message implies. Consider for example, the second message in the Needham Schroeder protocol [1] in which a server S sends a response to an initiator A containing a session key K_{ab}, along with a message for another principal B encrypted using B's key containing the same session key and A's identity. In typical Alice-Bob notation used in literature this can be expressed as:

$$S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$$

where N_a is a nonce value. K_{as} and K_{bs} represent keys shared between A and S, B and S respectively. The message is idealized as follows:

$$S \rightarrow A : \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \sharp(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

The idealization makes explicit that the server says that K_{ab} is a shared key for communication between *A* and *B* and also that it is fresh (due to the presence of the nonce).

Analysis. Protocol analysis in inference construction approaches involves two main tasks: (*i*) identification of an initial set of beliefs i.e. assumptions at each principal. (*ii*) message-by-message manual reasoning based on combining formula (idealized messages) that a principle sees and what it knows using inference rules of the logic.

4 Extending Belief Logic

Our goal is to extend BAN for analysis of generic browser based web transactions. This requires us to extend the logic of authentication in the following ways.

Supporting Principals without Identifying Keys. Existing techniques for security protocol analysis require principals to possess identifying keys. However, it is common for users to authenticate to websites using passwords over secure connections. A *secure channel* in this paper refers to a transport layer security mechanism e.g. SSL, TLS that provides server authentication, confidentiality and integrity in message exchanges. We introduce a new sort (type) in the many sorted BAN logic called *user* which represents the client side of a secure connection.

Support for Passing Domain Specific Information. There is often need for exchanging protocol specific information. The information could represent a role of a participant, a set of authorizations required from another site etc. We allow this by providing protocol specific parameters to be defined.

Support for User Actions and Secrets. While principals make statements signed using identifying keys, users interact with principals (usually servers) over secure channels. We define the concept of an action and allow it to be associated with a user. We allow secrets to be associated with actions in order to identify a user that performed an action (possibly at a different place or time). Actions are defined as *Aname* (t_1,t_2..t_n) where t_i represents a sort (type) in the logic or a parameter name. A specific action is *SignIn* (*Principal*) which represents user signing in as a principal.

The new concepts are represented using the notation described in Table 2.

Table 2. Additional operators used in our extended belief logic. *Aname* ranges over action names, while *Pname* ranges over parameter names. Secure channel associated with a user is identified by a suffix. New operators are allowed in formula *X* in inference rules R1-R3.

Notation	Meaning	Notation	Meaning
$P \overset{\Delta}{\longleftrightarrow} U_C$	*C is a secure channel between U_c and P.*	$X \rightsquigarrow Aname$	*Secret X is associated with action Aname.*
$[\![X]\!]_C$	*Formula X exchanged over secure channel C.*	$U_C \ni X$	*U_c possesses secret X.*
$U_C \triangleright Aname((t_1..t_n))$	*U_C performs action Aname*	*Pname = val*	*Parameter Pname has value 'val'.*

4.1 Reasoning about Users, Actions and Secrets

The following new inference rules have been defined to reason about users, their actions and associated secrets.

Table 3. Inference rules added to the logic. The description assumes that P is connected to U_c over a secure channel C.

Inference Rule (R4, R5, R6)	Description
$\dfrac{P \models (P \overset{\Delta}{\longleftrightarrow} U_C), P \lhd [\![Aname]\!]_C}{P \models (U_C \rhd Aname)}$	*If P sees an action on the channel C it believes U_c performed the action.*
$\dfrac{P \models (S \rightsquigarrow Aname), P \models (P \overset{\Delta}{\longleftrightarrow} U_C), P \lhd [\![S]\!]_C}{P \models (U_C \rhd Aname)}$	*If P sees a secret associated with an action on channel C, it believes U_c performed the action.*
$\dfrac{P \models (P \overset{\Delta}{\longleftrightarrow} U_c), P \models U_c \rhd SignIn(Q), P \lhd [\![Aname]\!]_C}{P \models (Q \rhd Aname)}$	*If P believes that U_c has already signed in as Q, then any further actions on C are associated with Q.*

4.2 Example: Analysis of SAML Web Single Sign-On

We assume that all communication between client and servers is protected by a secure channel. We introduce new parameters identifying roles of identity provider, *Prov* and identity consumer, *Cons* and service, *Service*. We introduce the action *Request* (*Service*) and use the action *SignIn* (*Principal*) defined in Section 4. We include only those messages in the idealized version with target as *SP* (assertion consumer, *C*) or IdP (assertion provider, P) since we are interested in beliefs only at these participants:

$$\begin{aligned}
&\text{Message 1} \quad U_{C1} \rightarrow C: \quad [\![Request(S)]\!]_{C1} \\
&\text{Message 3} \quad U_{C2} \rightarrow P: \quad [\![\{N_{cp}, prov = P, cons = C\}_{K_C^{-1}}]\!]_{C2} \\
&\text{Message 5} \quad U_{C2} \rightarrow P: \quad [\![N_{cp}, SignIn(Q)]\!]_{C2} \\
&\text{Message 7} \quad U_{C3} \rightarrow C: \quad [\![\{N_{cp}, T, T \rightsquigarrow SignIn(Q), \\
&\qquad\qquad prov = P, cons = C\}_{K_P^{-1}}]\!]_{C3}
\end{aligned}$$

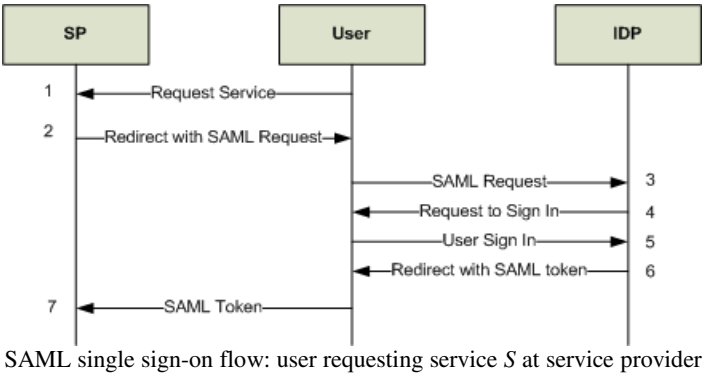


Fig. 1. The SAML single sign-on flow: user requesting service *S* at service provider (SP) site is redirected to Identity Provider (IdP) with SAML request. After authenticating the user, IdP redirects user back to SP site with a signed SAML token having the asserted identity (*Q*).

The SAML request contains a request identifier and a timestamp which we combine as a single nonce value N_{cp} in Message 3. Message 7 is the SAML response with token T, association of the token with the sign-in action performed by the user. We now perform a BAN style analysis of the protocol. We start with the assumptions:

$$C \models C \overset{\triangle}{\leftrightarrow} U_{C1} \qquad P \models P \overset{\triangle}{\leftrightarrow} U_{C2}$$
$$C \models C \overset{\triangle}{\leftrightarrow} U_{C3}$$
$$C \models \sharp(N_{cp}) \qquad P \models \sharp(N_{cp})$$
$$C \models \mapsto_{K_P} P \qquad P \models \mapsto_{K_C} C$$
$$C \models (P \models\!\Rightarrow SignIn(X)) \qquad P \models (C \models\!\Rightarrow prov)$$
$$C \models (P \models\!\Rightarrow cons)$$

The first three assumptions are about secure channels. The next say that both C and P believe the nonce N_{cp} to be fresh. The next two statements are about C and P having knowledge of the others public key. The last assumption is about C trusting P on the subject of the *SignIn* action. The analysis is described in Table 4. The inference in the last row says that C now believes that U_{c3} has signed in as Q at P.

Table 4. SAML analysis for idealized messages. Receiver of message indicated in parentheses

Msg.	Reasoning	Rules	Inference
1 (C)	*Combining message with assumption about channel C1.*	R4	$C \models (U_{c1} \triangleright Request(S))$
3 (P)	*Combining message with assumption about K_c and N_{cp}.*	R1, R2	$P \models C \models (prov = P, cons = C)$
3 (P)	*Using assumption about C having control over prov.*	R3	$P \models (prov = P)$
5 (P)	*Combining message with assumption about channel C2.*	R4	$P \models (U_{c2} \triangleright SiginIn(Q))$
7 (C)	*Combining message with assumption about K_p and N_{cp}*	R1, R2	$P \models T \rightsquigarrow SiginIn(Q)$, $P \models (prov = P), P \models (cons = C)$
7 (C)	*Using assumption about P's control over SignIn(X).*	R3	$C \models (T \rightsquigarrow SiginIn(Q))$
7 (C)	*Using the fact that T is observed on secure channel C3.*	R5	$C \models (U_{C3} \triangleright SiginIn(Q))$

5 Model Driven Security Protocol Analysis

5.1 Automated Analysis of Belief Logics

Analysis of security protocols using a belief logic like BAN is often simple enough to be carried out manually and message-by-message in a way similar to the SAML example. Despite this there have been efforts to automate verification for these logics [9, 10]. We consider automation to be even more important for our logic which is not restricted to authentication problem and is a bit more complex than BAN. Existing

approaches like [9] perform mapping of BAN formula and inference rules to first-order logic. The approach essentially transforms all the assumptions, messages and inference rules into a single first order formula and feeds it to the theorem prover SETHEO for proving implication of a required theorem. There are two problems with such approaches. Firstly, combining formulae representing all the messages as described above is unsafe because it results in loss of relative timing and it becomes possible to use an inference rule to resolve a message based on information that became available only later in the protocol run. Secondly, existing theorem provers are not user-friendly enough to be incorporated in the software development process. For these reasons, we propose a model driven approach for automated analysis of protocols modeled using belief logics like BAN and its extensions.

5.2 Overview

Reasoning performed in belief logics like BAN is not very complex. Given a message, the applicable inference rules and the sequence in which they apply is often deterministic (e.g. R1 results in expressions that can be used with R2 which in turn leads to expressions usable in R3). We thus follow a forward chaining approach to discover all possible beliefs. We store beliefs at a principal using a model representing sorts (types) in the logic and relationships between them. The algorithm we use for combining messages with existing beliefs is aware of this model and makes queries only to retrieve relevant facts e.g. if it sees a message encrypted with K, it queries to find owners of K.

The entities and relationships in the model are not fixed and can be specific to a domain. This makes our approach model driven as the same analysis program works with different protocol specific models without any change. The analysis program is only impacted when new inference rules are added. Entities in the model correspond to all entities that appear in a protocol description while relationships correspond to beliefs established through a protocol execution. We call this the *domain model* of the protocol. An instance of this model is used at each participant to represent its current set of beliefs. This is termed as the *belief graph* of the participant.

The analysis program is given an idealized representation of the protocol to be examined, a domain model of the protocol and a set of assumptions. The program uses unification algorithms that we discuss in Section 5.5, which process an idealized formula by combining it with known facts available from the belief graph of the message recipient on the basis of inference rules of the logic, to generate beliefs. The beliefs are added to the belief graph of the participant. Once all the protocol messages have been processed by the program, the belief graph at each protocol participant represents its final set of beliefs.

5.3 Benefits of Model Driven Analysis

Ease of Use. The model driven analysis approach is based on UML which makes it easy to integrate with existing CASE tools. Viewing belief graphs generated by the

analysis program provides tremendous insight into the design of the protocol and often provides hints on resolving security issues.

Simplifying Idealization. A protocol parameter can be mapped to a high level protocol specific concept in the model. This reduces complexity in each idealized expression where the concept is referred.

Representing Protocol Properties. Some properties of the protocol are important for proving goals even though they are never communicated to the participants through protocol messages. Use of idealization to represent such properties is both arbitrary and error prone. We solve this problem by representing such properties in the model itself by specifying constraints, thus removing the need for them to be artificially communicated through idealized messages.

5.4 Modeling of Extended Belief Logic

Message handling algorithms that perform reasoning have to interact with the belief graph to query existing beliefs and to add new ones. This requires statements representing belief in the logic to be mapped to the model. Table 5 shows how atomic sentences of the logic are modeled in the graphical representation. Each important sort (type) of the logic is mapped to a UML class (stereotyped entity) and statements are mapped to associations. The important classes are `Principal`, `SharedKey`, `AsymKey`, `Token`, `User`, `Request`, `Action`. `Token` is used to represent both secrets and nonces, `AsymKey` to represent public/private keys, while `Request` is used to represent protocol sessions. `Action` is an abstract class and is the super class for specific actions like `SignIn`.

Table 5. Mapping of statements of the logic to graphical UML syntax. Multiplicity at association ends is used to limit ownership of entities like keys. Trust types in parentheses.

Statement	Graphical Representation	Statement	Graphical Representation
$P \xleftrightarrow{K} Q$ *(SkeyTrust)*	«entity» **Principal** — +subjects (2 — 0..*) — «entity» **SharedKey**	$\mapsto_K Q$ *(PkeyTrust)*	«entity» **Principal** — +subject (1 — 0..1) — «entity» **AsymKey** +type
$\sharp X$ *(NonceTrust)*	«entity» **Request** — +freshvals (1 — 0..*) — «entity» **Token** +val	$U \ni X$ *(HolderTrust)*	«entity» **User** +channel — +holder (0..* — 0..*) — «entity» **Token** +val
$U \triangleright Action$, $X \rightsquigarrow Action$ *(Action-Trust)*	«entity» **Action** — +performedBy (* — 1) — «entity» **User** +channel; +secret (0..*) — «entity» **Token** +val	$U \triangleright SignIn$ $X \rightsquigarrow SignIn$ *(SigninTrust)*	«entity» **SignIn** — +performedBy (* — 1) — «entity» **User** +channel; +authSub (0..* — *) — «entity» **Principal**; «entity» **Token** +val — +secret (*)
$P \xleftrightarrow{\Delta} U_c$ *(SecChannel Trust)*	«entity» **User** +channel		

Note that statements that include operators $\lhd, \{\ \}_K, [\![\ ]\!]_C$ appear only in premises of the inference rules of the logic are not required to be modeled as we are only interested in beliefs established at the participants. Jurisdiction statements are not mapped to the model. Instead each statement in Table 5 is associated with a trust type. An N x N table (where N is the number of participants) is used to represent trust between participants. An element of the table, TRUST [P,Q] is a set of trust types with each type t such that $P \models Q \models\Rightarrow t$. The table is provided as one of the assumptions to the analysis program.

5.5 Unification Algorithms

Reasoning involves unifying (combining) the message and one or more existing beliefs with the inference rules of the logic to obtain new beliefs. We now describe the message handling algorithms which process an idealized message, perform reasoning to obtain beliefs and store the beliefs in a participant's belief graph.

Handling Encrypted Formula. The algorithm HANDLEENCRFORMULA takes an encrypted statement of the logic, and the name of the principal that receives the formula as its input. Any encrypted sub-formulae are removed (routine EXTRACTENCRFORMULA) and processed using a recursive call. The rest of the algorithm corresponds to applying the nonce-verification and message-origin rules. Any tokens which represent fresh values or secrets as per the belief graph are extracted (GETTOKEN). If no nonces are found, then the message is not fresh and no beliefs can be established. Otherwise, the origin of the assertion is identified using the encryption key for the message. In the event the encryption key is not known, the message is rejected. If the key is the public key owned by the principal itself, then any secrets in the message are used to identify the origin. Once the source subject is identified, ADDBELIEF is called for each sub-formula.

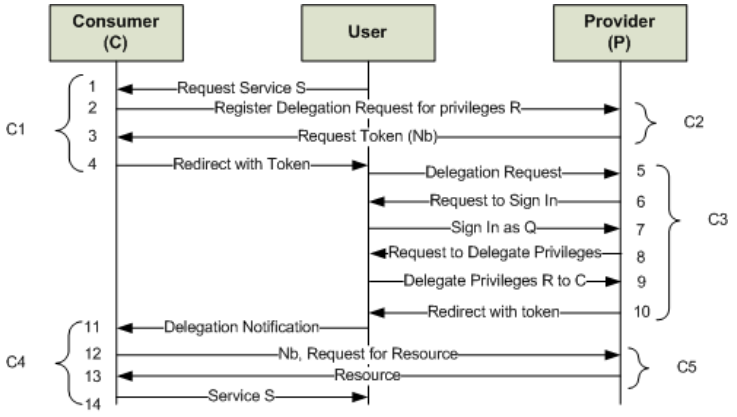
Handling Secure Channel Messages. The algorithm HANDLESECUREMSG takes a statement, a secure channel identifier for the channel on which it is received and the principal where it is received as the input. It is determined whether the channel is known to the principal (routine GETCHANNEL). If the principal is acting as a client, then the other end of the channel is the subject principal and an attempt is made to add each sub-formula using ADDBELIEF. If the principal is acting as a server, the other end represents a user and the formula could either be an action or a secret. If it is an action, the ADDACTION routine is called. In case of secret any associated actions are inferred to be performed by the user and added to the participant's belief graph. The ADDBELIEF routine adds beliefs (including beliefs about other participants beliefs e.g. $A \models\equiv B \models\equiv$) to a participant belief graph applying jurisdiction rule in the process.

```
procedure HANDLEENCRFORMULA(stmt, principal)
    local variables: m, formulae,
    subject ← null, candidates ←{ }, nonces←{ }, secrets←{ }
    while (m ← EXTRACTENCRFORMULA(stmt)) do
        HANDLEENCRFORMULA(m, principal)
    GETTOKENS(stmt, principal, nonces, secrets)
    if (nonces = { }) then return
    key ← KEY[stmt], formulae ← FORM[stmt], type← KEYTYPE [stmt]
    candidates ← GETPRINCIPALS(key)
    if (candidates = { }) return
    if (candidates = {principal} and SIZE[secrets]=1) then
        candidates ← GETPRINCIPALS(secrets)
    if (candidates = {principal, Q}) then subject ← Q
    else if (candidates = {Q}, type = Private) then subject ← Q
    if (subject = null) then return
    ADDBELIEF(formulae, subject, principal) return

procedure HANDLESECUREMSG (stmt, cid, principal)
    local variables: m, formulae, subject, channel ←null, action←null
    while (m ← EXTRACTENCRFORMULA(stmt)) do
        HANDLEENCRFORMULA(m, principal)
    channel ← GETCHANNEL(cid, principal), subject ← CLIENT[channel]
    if (channel = null) then return
    formulae ← FORM[stmt]
    if (subject= principal) then
        ADDBELIEF(formulae, SERVER[channel], principal) return
    for each (m ∈ formulae) do
        if (ACTION?(m)) then aname = ACTIONNAME[m]
            if (aname = SignIn) then
                ADDSIGNINACTION(subject, AUTHNAME[m], principal)
            else ADDACTION(subject, m, principal)
        else if (SECRET?(m)) then action = GETACTION(m)
            if (action = null) then ADDSECRET(subject, m, principal)
            else ADDACTION(subject, m, principal)
    return

procedure ADDBELIEF(formulae, subject, principal)
    for each (f ∈ formulae) do
        ADDSTATEMENT(f, subject, principal)
        if (TRUSTTYPE[f] ∈ TRUST[principal, subject])
            ADDSTATEMENT(f, principal, principal)
    return
```

6 OAuth Protocol Analysis

The OAuth protocol [4] provides a web based workflow that allows a user to temporarily delegate privileges of his account at a provider to a third party without sharing his login credentials. Privileges could mean access to pictures, contacts, blogs etc. OAuth is the primary protocol used by Google, Facebook and Twitter to allow third party access to their users' content. In this paper, we use OAuth to refer to the original version (1.0) of the protocol.

We show the concrete and idealized protocols next to each other in Fig. 2. Message 2 represents registration of delegation request by C at P. The message signed by C,



Message 1	$U \rightarrow C : Access\ Service\ S$	$\rightleftharpoons$	$U_{C1} \rightarrow C : \ [\![Request(S)]\!]_{C1}$
Message 2	$C \rightarrow P : \{R, N_c, url_c\}K_c^{-1}$	$\rightleftharpoons$	$C_{C2} \rightarrow P : \ [\![\{N_c, Privileges = R, Callback = url_c\}K_c^{-1}]\!]_{C2}$
Message 3	$P \rightarrow C : N_b$	$\rightleftharpoons$	$P \rightarrow C_{C2} : \ [\![OAuthTkn = N_p]\!]_{C2}$
Message 4	$C \rightarrow U : N_b, url_p$		
Message 5	$U \rightarrow P : N_b$	$\rightleftharpoons$	$U_{C3} \rightarrow P : [\![N_p]\!]_{C3}$
Message 6	$P \rightarrow U : Login\ request$		
Message 7	$U \rightarrow P : Q,\ password$	$\rightleftharpoons$	$U_{C3} \rightarrow P : [\![SiginIn(Q)]\!]_{C3}$
Message 8	$P \rightarrow U : Request\ for\ delegation$		
Message 9	$U \rightarrow P : Delegate\ R\ to\ C$	$\rightleftharpoons$	$U_{C3} \rightarrow P : [\![Delegate(R, C)]\!]_{C3}$
Message 10	$P \rightarrow U : N_b, url_c$		
Message 11	$U \rightarrow C : N_b$	$\rightleftharpoons$	$U_{C4} \rightarrow C : [\![N_p]\!]_{C4}$
Message 12	$C \rightarrow P : N_b,\ Access\ Content$		
Message 13	$P \rightarrow C : Content$	$\rightleftharpoons$	$P \rightarrow C : \ [\![x \triangleright Delegate(R, C)]\!]_{C5}$
Message 14	$C \rightarrow U : Provide\ Service\ S$		

Fig. 2. OAuth protocol flow. *Steps 1-4*: User requests service S from C which requires access privileges R to the user account at P. C registers delegation request with P and gets returned a OAuth token. C redirects user to P with the token. *Steps 5-10*: User is requested to sign in and delegate access and then redirected back to C with the token. *Steps 11-14*: C uses the token to get user content from P and provides service S to the user. C1-C5 represents secure channels.

contains list of privileges to be delegated, a callback URL (url_c) and a nonce N_c, identifying the request. In response P returns the request token, N_b.

An execution of the protocol can be uniquely identified as $OAuth(C, url_c, S, R, N_c, N_b, Q)$. We name the protocol parameters as *Consumer*, *Callback*, *Service*, *Privileges*, *Nonce_c*, *OAuthTkn, AuthPrincipal*. We only use parameters when there is no statement in the logic which can succinctly represent a message. In the idealized protocol we use only the following three parameters: *Callback*, *Privileges* and *OAuthTkn*. The user performs three actions in the protocol: request for service in message 1, signing in as Q in message 7 and delegating privileges to C in message 9. We define the following action types, *Request*(*Service*), *SignIn*(*Principal*) and *Delegate* (*Privileges*, *Principal*). Instances of these action types are *Request*(*S*), *SignIn*(*Q*) and *Delegate*(*R*, *C*).

Since we are interested only in beliefs established at principals C and P, we ignore messages with destination as user during idealization. The idealization of message 13 is interesting. P's returning requested user content to C implies that some user x has performed the delegation.

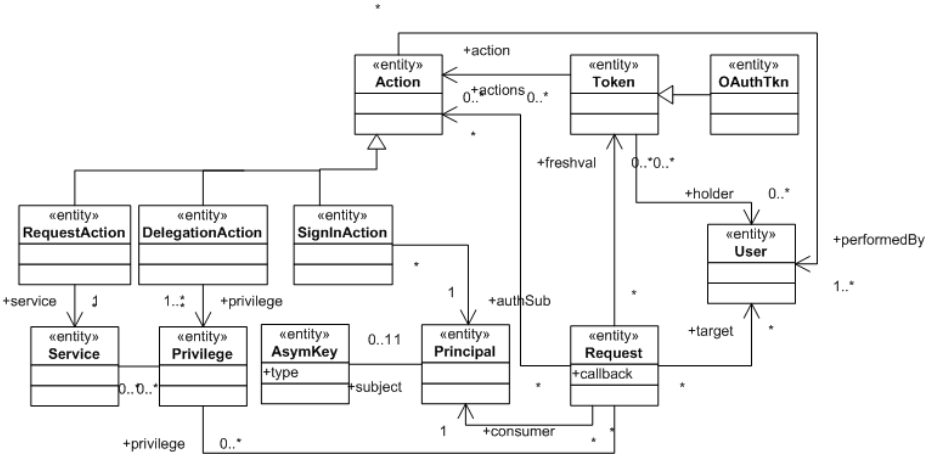


Fig. 3. Domain model for OAuth. Idealization is greatly simplified using parameters and actions that map to the domain model.

The domain model of the protocol is shown as a UML class diagram in Fig. 3. Entities `Principal`, `User`, `Action`, `Token`, `AsymKey` are from the mapping described in Section 5.4. `Service`, `Privileges` and `OAuthTkn` (sub-class of `Token`) are protocol specific entities. Classes representing the three specific user actions are defined and associated with other entities like `Service`, `Privilege`, `Principal`. The `target` relationship is used to identify the `User` which is provided the service in Message 14. The constraints on the model are described in Table 6. Property 1 in Table 6 is a property of the `OAuthTkn`. We can see how mapping parameter *OAuthTkn* to class `OAuthTkn` simplifies idealization by replacing two statements: $\sharp N_b$, $N_b \rightsquigarrow Request(S)$. Property 2 is an example of how protocol properties which are useful for proving goals but do not appear in an idealized message can be represented in the model.

Table 6. Constraints representing properties and goals of the protocol. Object constraint Language (OCL) expressions are used to specify constraints.

Type	OCL Expression	Meaning
Definition	**def** *delegator* : `User` = self.*actions*-> **select**(OclAsType(`DelegationAction`)). *performedBy*	Identify user who performed delegation as *delegator*.
Property 1	**context** `OAuthTkn` *self.isFresh* = *true* **and** self.action-> **select**(OclAsType(`RequestAction`))	An OAuth Token is fresh as well a secret associated with Request action.
Property 2	**context** `Request` *delegator* =self.*freshvals*-> **select**(OCLAsType(`OAuthTkn`)).*holder*	User who delegates privileges also holds OAuth token.
Goal	**context** `Request` self.*target* = *delegator*	Service is given to *delegator.*

Fig. 4 shows the belief graph at *C* after the automated analysis program has processed all idealized message using HANDLESECUREMSG. We can see that the goal constraint in Table 6 is not satisfied since user U_{C4} (which is the target) is not associated with the delegation action. Inspecting Fig. 4, it is not hard to find an assumption, which if true, can satisfy the goal. User *x,* associated with delegation action, is also holder of OAuth token as are users U_{C1} and U_{C4}. If one assumes that U_{C1} which originally received the token does not share the token then all user instances shown in Fig. 4 can be merged and we can immediately see that the target and delegator links point to the same user instance. However, this is clearly an unreasonable assumption to make for an unauthenticated user.

A security issue [16] was reported with the original version of OAuth which was fixed in the next release of the protocol. The attack works as follows: The attacker can perform steps 1-4 (see Fig. 2) of the protocol with the Consumer, send a link with the request token to the victim (who has an account with the provider) which then performs steps 5-10 and then rejoin the protocol from step 11 onwards. This quite clearly exploits the invalid assumption identified above.

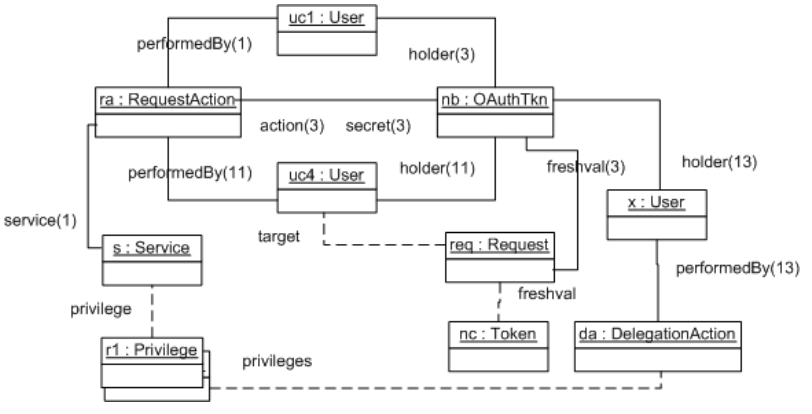


Fig. 4. Belief graph generated by the analysis program for Consumer C. Objects names are derived from idealized expressions. Broken lines represent assumptions. For other links the message number that resulted in the belief is identified in parentheses.

7 Conclusion

Protocols that provide user management services to third parties are central to the cloud computing model and analyzing their security properties is extremely important. We identified some unique aspects of these protocols that do not allow analysis methods for cryptographic protocols to be applied. We find that existing inference construction based approaches are the closest to satisfying the need. We extend an important belief logic that has been used successfully for analyzing numerous cryptographic protocols for analysis of web based identity services. We provide algorithms to automate analysis for belief based analysis approaches. We demonstrate the extended logic and model driven approach through analysis of two of the most well known browser based IDaaS protocols. UML has particularly wide acceptance in the software industry and supported by all major CASE tools. Use of UML for automating security analysis allows third party developers to appreciate implications of using standard or custom identity services in their solutions.

References

1. Burrows, M., Abadi, M., Needham, R.: A Logic of Authentication. ACM Transactions on Computer Systems (TOCS) 8(1), 18–36 (1990)
2. OASIS SAML Specifications. SAML v2.0, Core, `http://saml.xml.org/saml-specifications`
3. OpenID 2.0 Specifications, `http://openid.net/specs/openid-authentication-2_0.html`
4. The OAuth 1.0 Protocol. IETF RFC: 5849, `http://www.rfc-editor.org/rfc/rfc5849.txt`
5. Gong, L., Needham, R., Yahalom, R.: Reasoning about Belief in Cryptographic Protocols. In: Proceedings 1990 IEEE Symposium on Research in Security and Privacy (1990)
6. Abadi, M., Tuttle, M.R.: A semantics for a logic of authentication. In: Proceedings of the ACM Symposium of Principles of Distributed Computing (1991)
7. Kessler, V., Wedel, G.: AUTLOG: An advanced logic of authentication. In: Proceedings of Computer Security Foundation Workshop VII, pp. 90–99 (1994)
8. Syverson, P., van Oorschot, P.: On unifying some cryptographic protocol logics. In: Proceedings of the Symposium on Security and Privacy, Oakland, CA, pp. 14–28 (1994)
9. Schumann, J.: Automatic Verification of Cryptographic Protocols with SETHEO. In: McCune, W. (ed.) CADE 1997. LNCS, vol. 1249, pp. 831–836. Springer, Heidelberg (1997)
10. Craigen, D., Saaltink, M.: Using EVES to analyze authentication protocols. Technical Report TR-96-5508-05, ORA Canada (1996)
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inform. Theory IT-29, 198–208 (1983)
12. Meadows, C.: Applying formal methods to the analysis of a key management protocol. Journal of Computer Security 1, 5–53 (1992)
13. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)

14. Armando, A., et al.: An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. Elec. Notes in Theoret. Comp. Sci. 125(1) (March 2005)
15. Groß, T.: Security analysis of the SAML single sign-on browser/artifact profile. In: Proceedings of 19th ACSAC 2003, pp 298–307. IEEE Computer Society Press (2003)
16. Hammer-Lahav, E.: Explaining the OAuth Session Fixation Attack, http://hueniverse.com/2009/04/explaining-the-oauth-session-fixation-attack/
17. Kumar, A.: Integrated Security Context Management of Web Components and Services in Federated Identity Environments. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 565–571. Springer, Heidelberg (2008)

Credibility-Based Trust Management for Services in Cloud Environments

Talal H. Noor and Quan Z. Sheng

School of Computer Science,
The University of Adelaide, Adelaide SA 5005, Australia
{talal,qsheng}@cs.adelaide.edu.au

Abstract. Trust management is one of the most challenging issues in the emerging cloud computing. Although many approaches have been proposed recently for trust management in cloud environments, not much attention has been given to determining the credibility of trust feedbacks. Moreover, the dynamic nature of cloud environments makes guaranteeing the availability of trust management services a difficult problem due to the unpredictable number of cloud consumers. In this paper, we propose a framework to improve ways on trust management in cloud environments. In particular, we introduce a credibility model that not only distinguishes between credible trust feedbacks, but also has the ability to detect the malicious trust feedbacks from attackers. We also present a replication determination model that dynamically decides the optimal replica number of the trust management service so that the trust management service can be always maintained at a desired availability level. The approaches have been validated by the prototype system and experimental results.

Keywords: Trust Management, Cloud Computing, Credibility Model, Service Availability.

1 Introduction

In recent years, cloud computing has been receiving much attention as a new computing paradigm for providing flexible and on-demand infrastructures, platforms and software as services [2,6]. Both the public and the private sectors can benefit from the adoption of cloud services. For instance, it only took 24 hours, at the cost of merely $240, for the New York Times to archive its 11 million articles (1851-1980) using Amazon Web Services[1].

Given the fact of the accelerated adoption of cloud computing in the industry, there is a significant challenge in managing trust among cloud providers, service providers, and service requesters. Indeed, trust is one of the top obstacles for the adoption and the growth of cloud computing [2,6,14]. Recently, a considerable amount of research works have recognized the significance of trust management

[1] http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 328–343, 2011.

and proposed several solutions to assess and manage trust based on trust feedbacks collected from participants [14,7,26,9]. However, one particular problem has been mostly neglected: to what extent can these trust feedbacks be credible. On the one hand, it is not unusual that a trust management system will experience malicious behaviors from its users. On the other hand, the quality of the trust feedbacks differs from one person to another, depending on how experienced she is. This paper focuses on improving ways on the trust management in cloud environments. In particular, we distinguish the following key issues of the trust management in cloud environments:

- **Trust Robustness.** Determining the credibility of trust feedbacks is a significant challenge due to the overlapping interactions between service requesters, service providers, and cloud providers. This is true because cloud service interactions are dynamic. It is more likely that a cloud consumer has many interactions with the same cloud service, leading to multiple trust feedbacks to the cloud service. In addition, it is difficult to know how experienced a cloud consumer is and from whom malicious trust feedbacks are expected. Indeed, the trust management protection still requires extensive probabilistic computations [29,16] and trust participants' collaboration by manually rating trust feedbacks [19].
- **Availability of the Trust Management Service.** In a cloud environment, guaranteeing the availability of the trust management service is a difficult problem due to the unpredictable number of cloud consumers and the highly dynamic nature of the cloud environment. Consequently, approaches that requires understanding of the trust participants' interests and capabilities through similarity measurements [25] are inappropriate in the cloud environment. Trust management systems should be adaptive and highly scalable.
- **Trust Feedback Assessment and Storage.** The trust assessment of a service in existing techniques is usually centralized, whereas the trust feedbacks come from distributed trust participants. Trust models that follow a centralized architecture are more prone to several problems including scalability, availability, and security (e.g., Denial of Service (DoS) attack) [13]. Given the open and distributed nature of cloud environments, we believe that centralized solutions are not suitable for trust feedback assessment and storage.

In this paper, we overview the design and the implementation of the trust management framework. This framework helps distinguish between credible trust feedbacks and malicious trust feedbacks through a credibility model. It also guarantees high availability of the trust management service. In a nutshell, the salient features of the framework are:

- **A Credibility Model.** We develop a credibility model that not only distinguishes between trust feedbacks from experienced cloud consumers and amateur cloud consumers, but also has the ability to detect the malicious

trust feedbacks from attackers (i.e., who intend to manipulate the trust results by giving multiple trust feedbacks to a certain cloud service in a short period of time).

- **A Replication Determination Model.** High availability is an important requirement to the trust management service. We propose to spread replicas of the trust management service and develop a *replication determination model* that dynamically determines the optimal number of trust management service replicas, which share the trust management workload, thereby always maintaining the trust management service at a desired availability level.
- **Distributed Trust Feedback Assessment and Storage.** To avoid the drawbacks of centralized architectures, our trust management service allows trust feedback assessment and storage to be managed in a distributed way. Each trust management service replica is responsible for trust feedbacks given to a set of cloud services.

The remainder of the paper is organized as follows. Section 2 overviews the related work. Section 3 briefly presents the design of the trust management framework. Section 4 details the trust management service, including the distributed trust feedback collection and assessment, as well as the replication determination model for high availability of the trust management service. Section 5 describes the details of our credibility model. Finally, Section 6 reports the implementation and several experimental evaluations and Section 7 provides some concluding remarks.

2 Related Work

Several research works recognized the significance of trust management [15,28,13]. In particular, trust management is considered as one of the critical issues in cloud computing and is becoming a very active research area in recent years [17,21,14,5].

Several trust management approaches were proposed as policy-based trust management. For instance, Hwang et al. [14] proposed a security aware cloud architecture that uses VPN or SSL for communication security, focusing on both the cloud provider's and the cloud consumer's perspectives. In the cloud provider's perspective, the proposed architecture uses the trust negotiation approach and the data coloring (integration) using fuzzy logic techniques. In the cloud consumer's perspective, the proposed architecture uses the Distributed-Hash-Table (DHT)-based trust-overlay networks among several data centers to deploy a reputation-based trust management technique. Brandic et al. [5] proposed a novel approach for compliance management in cloud environments to establish trust between different parties. The centralized architecture focuses on the cloud consumer's perspective that uses compliant management to help cloud consumers to have proper choices when selecting cloud services. Unlike previous works that use centralized architecture, we present a credibility model supporting distributed trust feedback assessment and storage. This credibility model also distinguishes between trustworthy and malicious trust feedbacks.

Other trust management approaches were proposed as reputation-based trust management. For example, Conner et al. [9] proposed a trust management framework for the service-oriented architecture (SOA) that focuses on the service provider's perspective to protect resources from unauthorized access. This framework has a decentralized architecture that offers multiple trust evaluation metrics, allowing service providers to have customized evaluation of their clients (i.e., service requesters). Malik and Bouguettaya [20] proposed reputation assessment techniques based on the existing quality of service (QoS) parameters. The proposed framework supports different assessment metrics such as majority rating, past rating history, personal experience for credibility evaluation, etc. Unlike previous works that require extensive computations or trust participants' collaboration by rating the trust feedbacks, we present a credibility model that include several metrics namely the *Majority Consensus* and the *Feedback Density* which facilitates the determination of credible trust feedbacks. We were inspired by Xiong and Liu who differentiate between the credibility of a peer and the credibility of a feedback through distinguishing several parameters to measure the credibility of the trust participants feedbacks [30]. However, their approach is not applicable in cloud environments because peers supply and consume services and they are evaluated on that base. In other words trust results are used to distinguish between credible and malicious feedbacks.

3 The Trust Management Framework

We propose a trust management framework based on the service-oriented architecture (SOA). In particular, our framework uses Web services to span several distributed trust management service nodes. Trust participants (i.e., the cloud consumers) can give their trust feedbacks or inquire about a certain cloud service's trust results using Simple Object Access Protocol (SOAP) or REST [24] messages. We design our framework in this way because of the dynamic nature of cloud environments (e.g., new cloud consumers can join while others might leave around the clock). This requires the trust management service to be adaptive and highly scalable in order to collect the trust feedbacks and update the trust results constantly. Figure 1 depicts the main components of the trust management framework, which consists of two different layers, namely the *Service Layer* and the *Service Requester Layer*.

The Service Layer. This layer represents the big umbrella which includes cloud services (i.e., IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service)), e-services (e.g., booking a flight) and the trust management service where a service requester can give trust feedbacks to a particular service. Interactions within this layer are considered as *Cloud Service Interaction* and *Trust Interaction*.

The Service Requester Layer. This layer consists of different service requesters who consume services in the service layer. For example, a user can book a flight

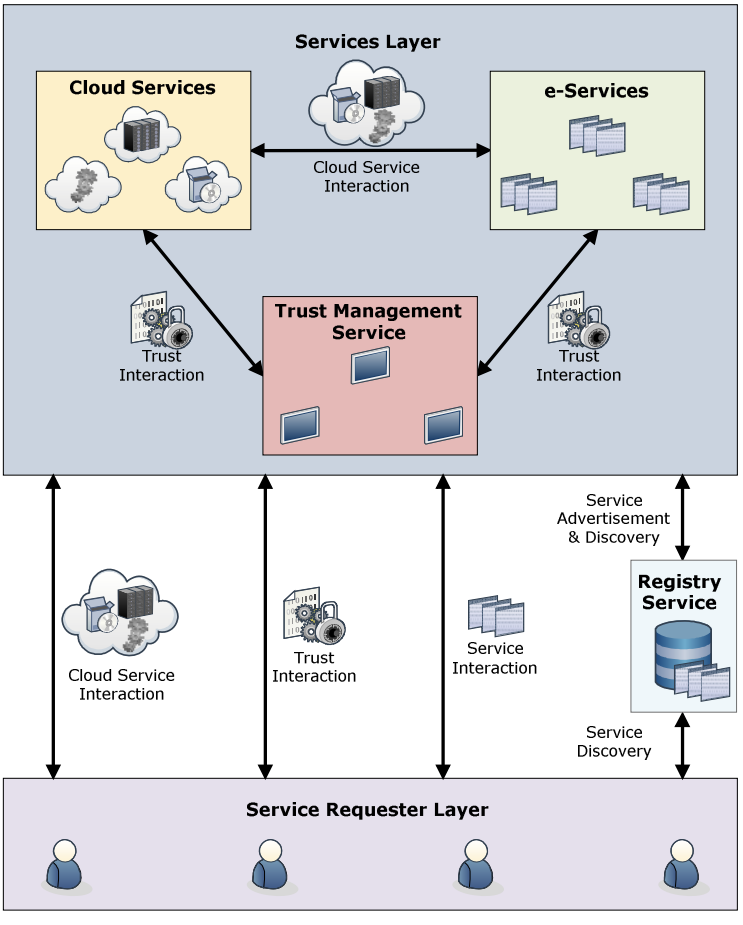


Fig. 1. Architecture of the Trust Management Framework

through an e-service provided by a certain airline company. A new startup that has limited funding can consume cloud services (e.g., hosting their services in Amazon S3). Service requesters can give trust feedbacks of a particular cloud service by invoking the trust management service (see Section 4).

Our framework also contains a *Registry Service* (see Figure 1) that has the following responsibilities:

- *Service Advertisement.* Both cloud providers and service providers are able to advertise their services through the *Service Registry.*
- *Service Discovery.* Service providers, cloud providers, and service requesters are able to access the *Service Registry* to discover services.

3.1 Assumptions and Attack Models

We assume that communications are secure. Attacks that occur in the *communication security level* such as *Man-in-the-Middle* (MITM) attack [3] are beyond the scope of this work. We also assume that cloud consumers have unique identities. Attacks that use the notion of multiple identities (i.e., the *Sybil* attack [12]) or *Whitewashing* attack that occur when the malicious cloud consumers (i.e., attackers) desperately seek new identities to clean their negative history records [18] are also beyond the scope of this work. In this paper, we only consider two types of malicious behaviors including *Self-promoting* attack and *Slandering* attack.

Self-promoting Attack. This attack arises when the malicious cloud consumers attempt to increase their trust results [10] or their allies in order to achieve their common interests. In the proposed framework this type of attack can happen in two cases. The first case (*Individual Collusion*) occurs when a certain malicious cloud consumer gives numerous fake or misleading trust feedbacks to increase the trust results of a certain cloud service. The second case (*Collaborative Collusion*) occurs when several malicious cloud consumers collaborate to give numerous fake or misleading trust feedbacks.

Slandering Attack. This attack is considered as the opposite of the *Self-promoting* attack that happens when the malicious cloud consumers try to decrease the trust results of certain cloud service [4]; this aggressive behavior is taken because of jealousy from competitors. In the proposed framework this type of attack can also happen either through *Individual Collusion* or *Collaborative Collusion*.

Service requesters can give trust feedbacks for a certain cloud service or send a query to the trust management service regarding a certain cloud service. In the following sections, we will focus on introducing our design of the trust management service.

4 Trust Management Service

4.1 Trust Feedback Collection and Assessment

In our framework, the trust behavior of a cloud service is represented by a collection of invocation history records denoted as $\mathcal{H}$. Each cloud consumer c holds her point of view regarding the trustworthiness of a specific cloud service s in the invocation history record which is managed by a trust management service. Each invocation history record is represented in a tuple that consists of the cloud consumer primary identity $\mathcal{C}$, the cloud service identity $\mathcal{S}$, a set of trust feedbacks $\mathcal{F}$ and the aggregated trust feedbacks weighted by the credibility $\mathcal{F}_c$ (i.e., $\mathcal{H} = (\mathcal{C}, \mathcal{S}, \mathcal{F}, \mathcal{F}_c)$. Each feedback in $\mathcal{F}$ is represented in numerical form with the range of [0, 1], where 0, +1, and 0.5 means negative feedback, positive feedback, and neutral respectively.

Whenever a cloud consumer inquires the trust management service regarding the trustworthiness of a certain cloud service s, the trust result, denoted as $\mathcal{T}r(s)$, is calculated as the following:

$$\mathcal{T}r(s) = \frac{\sum_{l=1}^{|\mathcal{V}(s)|} \mathcal{F}_c(l,s)}{|\mathcal{V}(s)|} \tag{1}$$

where $\mathcal{V}(s)$ is all of the feedbacks given to the cloud service s and $|\mathcal{V}(s)|$ represents the length of the $\mathcal{V}(s)$ (i.e., the total number of feedbacks given to the cloud service s). $\mathcal{F}_c(l,s)$ are the trust feedbacks from the l^{th} cloud consumer weighted by the credibility.

The trust management service distinguishes between credible trust feedbacks and malicious trust feedbacks through assigning the credibility aggregated weights $\mathcal{C}r(l,s)$ to the trust feedbacks $\mathcal{F}(l,s)$ as shown in Equation 2, where the result $\mathcal{F}_c(l,s)$ is held in the invocation history record h and updated in the assigned trust management service. The details on how to calculate $\mathcal{C}r(l,s)$ is described in Section 5.

$$\mathcal{F}_c(l,s) = \mathcal{F}(l,s) * \mathcal{C}r(l,s) \tag{2}$$

4.2 Availability of the Trust Management Service

Guaranteeing the availability of the trust management service is a significant challenge due to unpredictable number of invocation requests the service has to handle at a time, as well as the dynamic nature of the cloud environments. An emerging trend for solving the high-availability issue is centered on replication. In our approach, we propose to spread trust management service replicas over various clouds and dynamically direct requests to appropriate clouds (e.g., with lower workload), so that its desired availability level can be always maintained.

However, there is clearly a trade-off between high availability and replication cost. On the one hand, more clouds hosting trust management service means better availability. On the other hand, more replicas residing at various clouds means higher overhead (e.g., cost and resource consumption such as bandwidth and storage space). Thus, it is essential to develop a mechanism that helps to determine the optimal number of the trust management service replicas in order to meet the trust management service's availability requirement.

We propose a replication determination model to allow the trust management service to know how many replicas are required to achieve a certain level of availability. Given the trust management service s_{tms} failure probability denoted p that ranges from 0 to 1, the total number of s_{tms} replicas denoted r, and the availability threshold denoted e_a that also ranges from 0 to 1. The desired goal of the replication is to ensure that at least one replica of the trust management service is available, represented in the following formula:

$$e_a(s_{tms}) < 1 - p^{r(s_{tms})} \tag{3}$$

where $p^{r(s_{tms})}$ represents the probability that all trust management service replicas are failed, and $1 - p^{r(s_{tms})}$ represents the opposite (i.e., the probability of at least one trust management replica is available). As a result, the optimal number of trust management service replicas can be calculated as follows:

$$r(s_{tms}) > log_p(1 - e_a(s_{tms})) \tag{4}$$

For example, if the availability threshold $e_a(s_{tms}) = 0.9999$ and the failure probability of the trust management service $p = 0.2$ (low), $r(s_{tms}) > 5.723$, meaning that at least 6 trust management service replicas are needed. Similarly, if $e_a(s_{tms}) = 0.9999$ and the failure probability of the trust management service $p = 0.8$ (high), $r(s_{tms}) > 41.28$ which means at least 42 replicas are required.

Whenever a cloud consumer needs to send the invocation history record or query the trust result of a certain cloud service, $h(c, s)$ can be sent to a particular trust management service decided by using a consistent hash function (e.g., sha-256) as follows:

$$Tms_{id}(s) = \left(\sum_{i=1}^{|hash(s)|} byte_i\,(hash(s)) \right) mod\ r(s_{tms}) \tag{5}$$

where the first part of the equation represents the sum of each byte of the hashed cloud service identity $hash(s)$. The second part of the equation represents the optimal number of the trust management service replicas $r(s_{tms})$. This insures that the chosen trust management service replica is within the optimal number range.

5 The Credibility Model

Since the trust behavior of a cloud service in our framework is represented by a collection of invocation history records that contain cloud consumers trust feedbacks, there is a considerable possibility of the trust management service receiving *inaccurate* or even *malicious* trust feedbacks from amateur cloud consumers (e.g., who lack experience) or vicious cloud consumers (e.g., who submit lots of negative feedbacks in a short period in order to disadvantage a particular cloud service). To overcome these issues, we propose a *credibility model*, which considers several factors including the *Majority Consensus* and the *Feedback Density*.

5.1 Majority Consensus

It is well-known that the majority of people usually agree with experts' judgments about what is good [8]. Similarly, we believe that the majority of cloud consumers agree with *Expert Cloud Consumers'* judgments. In other words, any cloud consumer whose trust feedback is close to the majority trust feedbacks is considered as an *Expert Cloud Consumer*, *Amateur Cloud Consumers* otherwise.

In order to measure how close the cloud consumer's trust feedbacks to the majority trust feedbacks (i.e., the *Majority Consensus*, $\mathcal{J}(c)$), we use the slandered deviation (i.e., the root-mean-square) which is calculated as follows:

$$\mathcal{J}(c) = 1 - \sqrt{\frac{\sum_{h \in \mathcal{V}c(c)} \left(\sum_{k=1}^{|\mathcal{V}c(c,k)|} \left(\frac{\mathcal{F}(c,k)}{|\mathcal{V}c(c,k)|} - \left(\frac{\sum_{l \neq c, l=1}^{|\mathcal{V}c(l,k)|} \mathcal{F}(l,k)}{|\mathcal{V}(k)| - |\mathcal{V}c(c,k)|} \right) \right) \right)^2}{|\mathcal{V}c(c)|}} \quad (6)$$

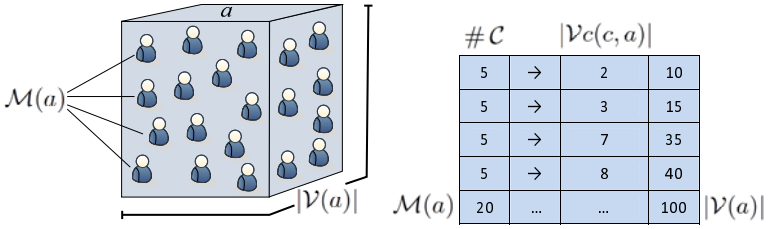
where the first part of the numerator represents the mean of the cloud consumer c's trust feedbacks $\mathcal{F}(c,k)$ for the k^{th} cloud service. The second part of the numerator represents the mean of the majority trust feedbacks given by other cloud consumers denoted $\mathcal{F}(l,k)$ (i.e., the l^{th} cloud consumer trust feedbacks, except the cloud consumer c's trust feedbacks) to the k^{th} cloud service. This procedure is done for all cloud services to which cloud consumer c gives trust feedbacks (i.e., $\mathcal{V}c(c)$).

5.2 Feedback Density

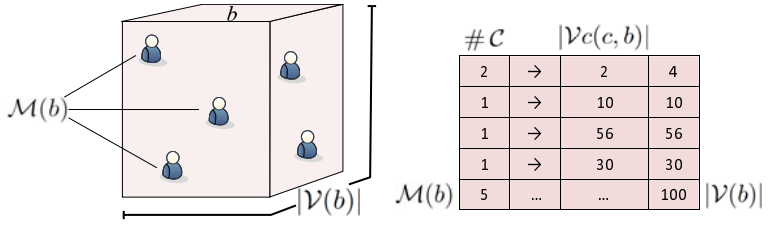
Some malicious cloud consumers may give numerous fake or misleading trust feedbacks to increase or decrease the trust result for a certain cloud service in order to achieve their personal interests (i.e., *Self-promoting* and *Slandering* attacks). Several online reputation-based systems such as auction systems (e.g., eBay [11], and Amazon [1]), have tried to help their consumers to overcome such attacks based on revealing the aggregated trust feedbacks as well as the number of trust feedbacks. The number of trust feedbacks gives the evaluator a hint in determining how credible the trust feedback is, which is supported by the research findings in [30,27].

However, the number of trust feedbacks is not enough in determining how credible the aggregated trust feedbacks are. For instance, suppose there are two different cloud services a and b as shown in Figure 2. The aggregated trust feedbacks of the both cloud services are high (i.e., a has 90% positive feedbacks from 100 feedbacks, b has 93% positive feedbacks from 100 feedbacks). Intuitively, cloud consumers should proceed with the cloud service that has the highest aggregated trust feedbacks (e.g., cloud service b in our case). However, *Self-promoting* attack might has been performed on cloud service b, which clearly should not be selected by cloud consumers.

In order to overcome this problem, we introduce the concept of *Feedback Density*, which facilitates the determination of credible trust feedbacks. Specifically, we consider the total number of cloud consumers who gave trust feedbacks to a particular cloud service as the *Feedback Mass*, the total number of trust feedbacks given to the cloud service as the *Feedback Volume*. The feedback volume is influenced by the *Feedback Volume Collusion* factor which is controlled by a specified volume collusion threshold. This factor regulates the multiple trust feedbacks extent that could collude the overall trust feedback volume. For instance, if the volume collusion threshold is set to 5 feedbacks, any cloud consumer



(a) Cloud Service a



(b) Cloud Service b

Fig. 2. Trust Feedback Density Determination

c who gives more than 5 feedbacks is considered to be suspicious of involving in a feedback volume collusion. The feedback density of a certain cloud service s, $\mathcal{D}(s)$, is calculated as follows:

$$\mathcal{D}(s) = \frac{\mathcal{M}(s)}{|\mathcal{V}(s)| * \left(\left(\frac{\sum_{h \in \mathcal{V}(s)} \left(\sum_{l=1}^{|\mathcal{V}(l,s)|} \left(\sum_{|\mathcal{V}c(l,s)| > e_v(s)} |\mathcal{V}c(l,s)|\right)\right)}{|\mathcal{V}(s)|}\right) + 1\right)} \tag{7}$$

where $\mathcal{M}(s)$ denotes the total number of cloud consumers who gave trust feedbacks to the cloud service s (i.e., the *Feedback Mass*). $|\mathcal{V}(s)|$ represents the total number of trust feedbacks given to the cloud service s (i.e., the *Feedback Volume*). The second part of the denominator represents the *Feedback Volume Collusion* factor. This factor is calculated as the ratio of the number of trust feedbacks given by the cloud consumers who give feedbacks more than the specified volume collusion threshold (i.e., $e_v(s)$) over the total number of feedbacks received by the cloud service (i.e., $|\mathcal{V}(s)|$). The idea behind adding 1 to this ratio is to reduce the value of the multiple trust feedbacks which are given diversely from the same cloud consumer.

Figure 2 depicts the same example mentioned before where the first row in the table on the right side of Figure 2(a) shows that 5 particular cloud consumers gave 2 feedbacks to the cloud service a in which the total number of those trust feedbacks is 10. The last row shows the total number of cloud consumers (i.e., $\mathcal{M}(a) = 20$) and the total number of trust feedbacks given to the cloud service a (i.e., $|\mathcal{V}(a)| = 100$). Both cloud services a and b have the same total number of

trust feedbacks (i.e., $|\mathcal{V}(a)| = 100$ and $|\mathcal{V}(b)| = 100$) and very close aggregated feedbacks (e.g., a has 90% positive feedbacks and b has 93% positive feedbacks).

However, the *Feedback Mass* of the cloud service a is higher than the Feedback Mass of the cloud service b (i.e., $\mathcal{M}(a) = 20$ and $\mathcal{M}(b) = 5$). If the volume collusion threshold e_v is set to 3 feedbacks per cloud consumer, 15 cloud consumers gave more than 3 feedbacks to the cloud service a where the total amount of trust feedbacks' lengths $|\mathcal{V}c(c, a)| = 70$ feedbacks; while 3 cloud consumers gave more than 3 feedbacks to the cloud service b where the total amount of trust feedbacks' lengths $|\mathcal{V}c(c, b)| = 80$ feedbacks. According to Equation 7, the *Feedback Density* of the cloud service a is higher than the cloud service b (i.e., $\mathcal{D}(a) = 0.118$ and $\mathcal{D}(b) = 0.028$). In other words, the higher the *Feedback Density*, the more credible the aggregated feedbacks are. The lower the *Feedback Density*, the higher possibility of collusion in the aggregated feedbacks.

Based on the specified trust feedbacks credibility factors (i.e., majority consensus and feedback density), the trust management service distinguishes between trust feedbacks from experienced cloud consumers and the ones from amateur or even vicious cloud consumers through assigning the credibility aggregated weights $\mathcal{C}r(c, s)$ to each of the cloud consumers trust feedbacks as shown in Equation 2. The credibility aggregated weights $\mathcal{C}r(c, s)$ is calculated as follows:

$$\mathcal{C}r(c, s) = \frac{\mu * \mathcal{J}(c) + \rho * \mathcal{D}(s)}{\lambda} \tag{8}$$

where μ and $\mathcal{J}(c)$ denote the *Majority Consensus* factor's normalized weight (i.e., parameter) and the factor's value respectively. The second part of the equation represents the *Feedback Density* factor where ρ denotes the factor's normalized weight and $\mathcal{D}(s)$ denotes the factor's value. λ represents the number of factors used to calculate $\mathcal{C}r(c, s)$. For example, if we only consider majority consensus, $\lambda = 1$; if we consider both the majority consensus and the feedback density, $\lambda = 2$.

6 Implementation and Experimental Evaluation

In this section, we report the implementation and preliminary experimental results in validating the proposed approach. Our implementation and experiments were developed based on the NetLogo platform [23], which was used to simulate the cloud environments. We particularly focused on validating and studying the performance of the proposed credibility model (see Section 5).

Since it is hard to find some publicly available real-life trust data sets, in our experiments, we used Epinions[2] rating data set which was collected by Massa and Avesani [22]. The reason that we chose Epinions data set is due to its similar data structure (i.e., consumers' opinions and reviews on specific products and services) with our cloud consumer trust feedbacks. In particular, we considered `user_id` in Epinions as the cloud consumer primary identity $\mathcal{C}$, `item_id` as the cloud service identity $\mathcal{S}$, and we normalized the `rating_value` as the cloud

[2] http://www.trustlet.org/wiki/Downloaded_Epinions_dataset

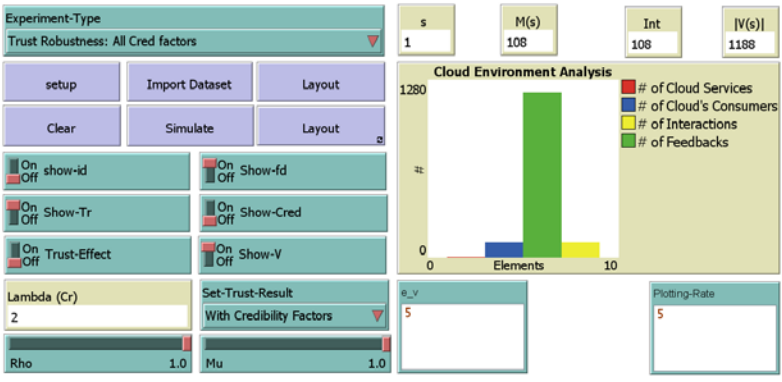


Fig. 3. Netlogo-based Prototype System's GUI

consumers trust feedbacks $\mathcal{F}$ to scale of 0 to 1. The data set has 49,290 users, 139,738 items, and 664,824 trust feedbacks. Figure 3 depicts the Graphical User Interface (GUI) for a cloud service. We imported the Epinions data set to create the cloud environment that we are intending to analyze.

We evaluate the trust robustness of our credibility model using both *analytical analysis* and *empirical analysis*. The analytical analysis focuses on measuring the trust result robustness (i.e., with respect to *Malicious Behavior Rate* of malicious cloud consumers) when using the credibility model and without using the credibility model. The analytical model calculates the trust results without weighting the trust results (i.e., we turn the $\mathcal{C}r(c, s)$ to 1 for all trust feedbacks). The empirical analysis focuses on measuring the trust result robustness for each factor in our credibility model including the *Majority Consensus* and the *Feedback Density*. The parameters setup for each corresponding experiment factor are depicted in Table 1.

Table 1. Experiment Factors and Parameters Setup

Experiment Design	μ	ρ	λ	$\mathcal{C}r(c, s)$
With Credibility factors	1	1	2	
Without Credibility factors				1
Majority Consensus factor	1	0	1	
Feedback Density factor	0	1	1	

Figure 4 depicts the analytical analysis of the trust results for a particular cloud service. From the figure, it can be seen that the higher the malicious behavior rate the lower the trust results are when considering to calculate the trust with all credibility factors. On the other hand, the trust results shows nearly no response to the malicious behavior rate when considering to calculate the trust without credibility factors. This demonstrates that our credibility model is robust and more sensitive in detecting malicious behaviors.

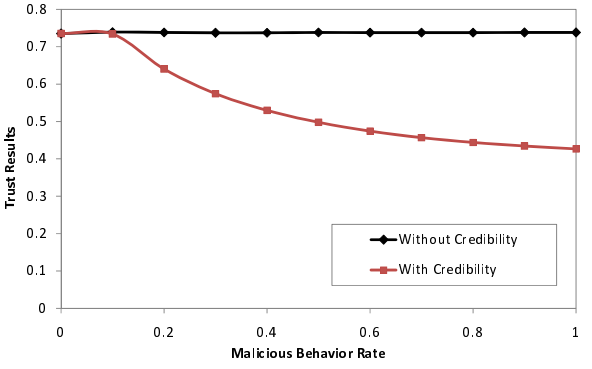


Fig. 4. Trust Robustness: With Credibility Vs. Without Credibility

Figure 5 shows the empirical analysis of the trust results for the same cloud service. It is clear that the trust results obtained by only considering the majority consensus factor are more accurate than the trust results obtained by only considering the feedback density factor when the malicious behavior rate is low (e.g., when the malicious behavior rate = 0.1, $\mathcal{T}r(s) = 0.59$ if we consider the majority consensus factor only while $\mathcal{T}r(s) = 0.73$ if we consider the feedback density factor only). This is true because there is still not many vicious cloud consumers (e.g., who submit lots of positive feedbacks in a short period in order to advantage a particular cloud service) during the trust aggregation. However, the trust results obtained by only considering the feedback density factor significantly response more when the malicious behavior rate become higher (e.g., when the malicious behavior rate = 0.9, $\mathcal{T}r(s) = 0.61$ for the majority consensus factor, $\mathcal{T}r(s) = 0.13$ for the feedback density factor). As a result, we can consider the majority consensus factor as a trust accuracy factor while the feedback density factor as a trust robustness factor (i.e., the feedback density factor is responsible for the robustness and the sensitiveness of our credibility model).

7 Conclusions and Future Work

Given the fact of the accelerated adoption of cloud computing in the recent years, there is a significant challenge in managing trust among cloud providers, service providers, and service requesters. In this paper, we present a trust management framework to manage trust in cloud environments. We introduce a credibility model that assesses cloud services' trustworthiness by distinguishing between credible trust feedbacks and amateur or malicious trust feedbacks. Also, the credibility model has the ability to detect the malicious trust feedbacks from attackers (i.e., who intend to manipulate the trust results by giving multiple trust feedbacks to a certain cloud service in a short period of time). We particularly introduce two trust parameters including the *Majority Consensus* factor and the *Feedback Density* factor in calculating the trust value of a cloud service.

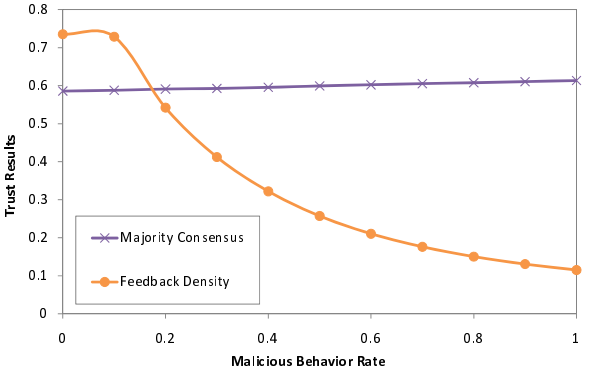


Fig. 5. Trust Robustness: Credibility Factors

In addition, our trust management service allows trust feedback assessment and storage to be managed in a distributed way.

In the future, we plan to deal with more challenging problems such as the *Sybil* attack and the *Whitewashing* attack. Performance optimization of the trust management service is another focus of our future research work.

References

1. Amazon: Amazon.com: Online shopping for electronics, apparel, computers, books, dvds & more (2011),`http://www.amazon.com/` (accessed March 01, 2011)
2. Armbrust, M., et al.: A View of Cloud Computing. Communiaction of the ACM 53(4), 50–58 (2010)
3. Aziz, B., Hamilton, G.: Detecting Man-in-the-Middle Attacks by Precise Timing. In: Proc. of the 3rd Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE 2009). Athens, Glyfada, Greece (June 2009)
4. Ba, S., Pavlou, P.: Evidence of the Effect of Trust Building Technology in Electronic Markets: Price Premiums and Buyer Behavior. MIS Quarterly 26(3), 243–268 (2002)
5. Brandic, I., Dustdar, S., Anstett, T., Schumm, D., Leymann, F., Konrad, R.: Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds. In: Proc. of IEEE 3rd Int. Conf. on Cloud Computing (CLOUD 2010), Miami, Florida, USA (July 2010)
6. Buyya, R., Yeo, C., Venugopal, S.: Market-oriented Cloud Computing: Vision, Hype, and Reality for Delivering it Services as Computing Utilities. In: Proc. of IEEE 10th Int. Conf. on High Performance Computing and Communications (HPCC 2008), Dalian, China (September 2008)
7. Chen, K., Hwang, K., Chen, G.: Heuristic Discovery of Role-Based Trust Chains in Peer-to-Peer Networks. IEEE Transactions on Parallel and Distributed Systems 20(1), 83–96 (2008)
8. Child, I.: The Psychological Meaning of Aesthetic Judgments. Visual Arts Research 9(2(18)), 51–59 (1983)

9. Conner, W., Iyengar, A., Mikalsen, T., Rouvellou, I., Nahrstedt, K.: A Trust Management Framework for Service-Oriented Environments. In: Proc. of the 18th Int. Conf. on World Wide Web (WWW 2009), Madrid, Spain (April 2009)
10. Douceur, J.R.: The Sybil Attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
11. eBay: ebay - new & used electronics, cars, apparel, collectibles, sporting goods & more at low prices (2011), http://www.ebay.com/ (accessed March 01, 2011)
12. Friedman, E., Resnick, P., Sami, R.: Manipulation-Resistant Reputation Systems. In: Algorithmic Game Theory, pp. 677–697. Cambridge University Press, New York (2007)
13. Hoffman, K., Zage, D., Nita-Rotaru, C.: A Survey of Attack and Defense Techniques for Reputation Systems. ACM Computing Surveys (CSUR) 42(1), 1–31 (2009)
14. Hwang, K., Li, D.: Trusted Cloud Computing with Secure Resources and Data Coloring. IEEE Internet Computing 14(5), 14–22 (2010)
15. Jøsang, A., Ismail, R., Boyd, C.: A Survey of Trust and Reputation Systems for Online Service Provision. Decision Support Systems 43(2), 618–644 (2007)
16. Jøsang, A., Quattrociocchi, W.: Advanced Features in Bayesian Reputation Systems. In: Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G. (eds.) TrustBus 2009. LNCS, vol. 5695, pp. 105–114. Springer, Heidelberg (2009)
17. Krautheim, F.J., Phatak, D.S., Sherman, A.T.: Introducing the Trusted Virtual Environment Module: A New Mechanism for Rooting Trust in Cloud Computing. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 211–227. Springer, Heidelberg (2010)
18. Lai, K., Feldman, M., Stoica, I., Chuang, J.: Incentives for Cooperation in Peer-to-Peer Networks. In: Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA (June 2003)
19. Malik, Z., Bouguettaya, A.: Rater Credibility Assessment in Web Services Interactions. World Wide Web 12(1), 3–25 (2009)
20. Malik, Z., Bouguettaya, A.: RATEWeb: Reputation Assessment for Trust Establishment Among Web services. The VLDB Journal 18(4), 885–911 (2009)
21. Manuel, P., Thamarai Selvi, S., Barr, M.E.: Trust Management System for Grid and Cloud Resources. In: Proc. of the 1st Int. Conf. on Advanced Computing (ICAC 2009), Chennai, India (December 2009)
22. Massa, P., Avesani, P.: Trust Metrics in Recommender Systems. In: Computing with Social Trust. Human-Computer Interaction Series, pp. 259–285. Springer, London (2009)
23. NetLogo: Netlogo home page (2011), http://ccl.northwestern.edu/netlogo/ (accessed March 1, 2011)
24. Sheth, A.P., Gomadam, K., Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Computing 11(6), 84–87 (2007)
25. Skopik, F., Schall, D., Dustdar, S.: Start Trusting Strangers? Bootstrapping and Prediction of Trust. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 275–289. Springer, Heidelberg (2009)
26. Skopik, F., Schall, D., Dustdar, S.: Trustworthy Interaction Balancing in Mixed Service-Oriented Systems. In: Proc. of ACM 25th Symp. on Applied Computing (SAC 2010), Sierre, Switzerland (March 2010)
27. Srivatsa, M., Liu, L.: Securing Decentralized Reputation Management Using TrustGuard. Journal of Parallel and Distributed Computing 66(9), 1217–1232 (2006)

28. Wang, Y., Vassileva, J.: Toward Trust and Reputation Based Web Service Selection: A Survey. International Transactions on Systems Science and Applications 3(2), 118–132 (2007)
29. Weng, J., Miao, C.Y., Goh, A.: Protecting Online Rating Systems from Unfair Ratings. In: Katsikas, S.K., López, J., Pernul, G. (eds.) TrustBus 2005. LNCS, vol. 3592, pp. 50–59. Springer, Heidelberg (2005)
30. Xiong, L., Liu, L.: Peertrust: Supporting Reputation-based Trust for Peer-to-Peer Electronic Communities. IEEE Transactions on Knowledge and Data Engineering 16(7), 843–857 (2004)

Monere: Monitoring of Service Compositions for Failure Diagnosis

Bruno Wassermann* and Wolfgang Emmerich

University College London,
Gower Street, London, WC1E 6BT, UK
{b.wassermann,w.emmerich}@cs.ucl.ac.uk

Abstract. Service-oriented computing has enabled developers to build large, cross-domain service compositions in a more routine manner. These systems inhabit complex, multi-tier operating environments that pose many challenges to their reliable operation. Unanticipated failures at runtime can be time-consuming to diagnose and may propagate across administrative boundaries. It has been argued that measuring readily available data about system operation can significantly increase the failure management capabilities of such systems. We have built an online monitoring system for cross-domain Web service compositions called Monere, which we use in a controlled experiment involving human operators in order to determine the effects of such an approach on diagnosis times for system-level failures. This paper gives an overview of how Monere is able to instrument relevant components across all layers of a service composition and to exploit the structure of BPEL workflows to obtain structural cross-domain dependency graphs. Our experiments reveal a reduction in diagnosis time of more than 20%. However, further analysis reveals this benefit to be dependent on certain conditions, which leads to insights about promising directions for effective support of failure diagnosis in large Web service compositions.

1 Introduction

Service-oriented technologies have simplified the development of larger, more complex software systems that now routinely span administrative and organisational boundaries. These large-scale distributed systems inhabit a complex operating environment that presents numerous threats to their dependability. They are often asynchronous and rely on best-effort networks with variable performance in order to integrate and share resources across domain boundaries. Communication across the Internet and heavy loads increase the potential for failures. Some of the components that are critical to the correct operation of an application may be in different administrative domains. These systems often rely on various layers of middleware components. Developers have been enabled to build such applications as compositions of services in a more routine manner

* This work is partially supported by the EC's 7^{th} Framework Programme under grant agreement n 215605 (RESERVOIR) and a BT EPSRC Case studentship.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 344–358, 2011.

through standards and middleware that attempt to hide much of the underlying complexity.

This transparency, which is so useful to developing large service compositions, becomes an obstacle to rapid diagnosis of failures at runtime. Given the complexity of the operating environment and the high demands placed upon it, there may be many unforeseen failures at runtime. Data about failures are spread across the various layers of the system and often across administrative domains. It has been argued [23,27,18,28,9,12] that the availability of large-scale distributed software systems could benefit from improved monitoring capabilities. The definition of availability, *Availability = Mean-time-to-failure / (Mean-time-to-failure + Mean-time-to-repair)*, further supports this intuition. Mean-time-to-repair (MTTR) itself is defined as $MTTR = MTT_{detect} + MTT_{diagnose} + MTT_{repair}$. Approaches to monitoring that improve understanding of system behaviour and thereby enable operators to reduce the time to detect and diagnose failure causes, should result in a significant increase in system availability. However, in practice the benefits of monitoring for the diagnosis of operational failures have only been demonstrated by argument or limited anecdotal evidence.

The contributions of this paper are two-fold. First, it provides a brief architectural and functional overview of the Monere monitoring framework for cross-domain Web service compositions. The idea behind Monere is to support operators with the early identification of system-level failures by measuring readily available attributes of system operation from all relevant components and integrating this data. Monere monitors application-level components, such as BPEL processes and Web services, middleware components including application servers, database servers, BPEL runtimes and Grid computing middleware as well as aspects of the operating system, such as the file system, network interfaces and running processes. We describe how Monere exploits the structure of BPEL workflows and knowledge about other components to create structural cross-domain dependency graphs. Second, this paper presents the results of a controlled experiment, in which we compare the failure diagnosis times achieved by 22 human operators when using Monere to that achieved using a standard UNIX toolset. The results illustrate under what circumstances such an approach is beneficial and point to further work in this area.

We briefly characterise the systems and types of failures we consider based on an example. Then, we provide an overview of the architecture of Monere and its key features, paying particular attention to its mechanism for dependency discovery. This is followed by a performance analysis of Monere, a detailed description of the controlled experiment and a discussion of its results.

2 The Polymorph Search Workflow

Global service compositions exhibit a number of interesting characteristics. They are asynchronous distributed systems communicating over the Internet. As network quality tends to vary and standard communication protocols are used, long delays can become indistinguishable from failed endpoints and lost messages.

A service composition relies on correct service from several layers of often interdependent middleware components. At the service-level, these applications also form complex dependencies across organisational boundaries, where any service may in turn be a high-level application that is composed of more basic functionality.

The Polymorph Search Workflow [14] is our case study. It is a large Web service composition expressed in the Business Process Execution Language (BPEL) [16] and shares many of the above characteristics. It is used by Theoretical Chemists for the computational prediction of organic crystal structures, which is a compute- and data-intensive application. Its middleware consists of several layers. A BPEL runtime is responsible for managing the execution of a set of BPEL processes comprising an application. It is intimately dependent on the services of a SOAP [10] runtime that manages the interactions between Web services. It is also the place where many of the decentralised middleware services, such as transactional mechanisms, are implemented. The SOAP runtime relies on an application server, which is executed by a JVM, and all components use the services of an operating system.

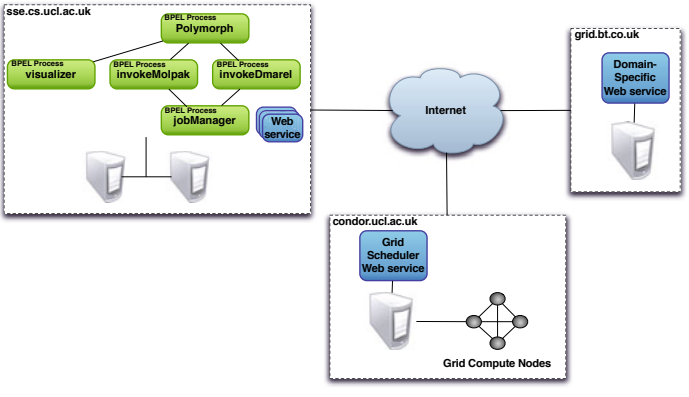


Fig. 1. The Polymorph Search Workflow and its deployment on our testbed

The deployment of the overall workflow on our testbed is shown in Figure 1. The workflow itself is hierarchically composed from several BPEL processes. The processes along with some utility Web services and their backend implementations are deployed across two hosts in the UCL domain. Some Grid computing middleware is used in addition to the standard Web services stack. Access to a set of compute nodes is provided through a Web service and underlying Condor [20] job scheduler instance on a third host. GridSAM [19] manages the interaction between the BPEL processes and Condor. Finally, the workflow also interacts with a Web service deployed in another administrative domain.

The failures we want to enable operators to diagnose more efficiently are operational failures that occur at runtime. System-level failures originate beneath the application-level in the middleware components, operating system

and the network. Failure is brought about by sustained high demands on some or all parts of the system, which over time leads to overload and eventually to resource exhaustion. They often depend on the operating environment of the service composition. Examples are the exhaustion of shared resources such as memory, threads or file descriptors by some components, network connectivity issues that prevent timely progress and slowdown of services or components due to being overloaded. Other common causes stem from the cross-domain nature of these applications, where resources and services fail, become unavailable or where varying network conditions make them appear so. Failure effects are almost never isolated or tolerated within a single component or even layer and cascade across layers and even across administrative domain boundaries. These failures often result in abnormal termination or bring the application to a virtual standstill.

3 Monere

Monere implements a number of key features. It automatically discovers all dependencies a BPEL process forms on other application components across domains and lower-level system components within a domain. It continuously collects metrics to aid in understanding of system behaviour and provides historical records of these measurements. The collected lower-level measurements can then be correlated with specific application activity. Finally, Monere integrates all these measurements from the various services, components and hosts in a single user interface.

3.1 Metrics

A great deal of data about system operation can be obtained without the need for substantial changes to the monitored systems. Monere tracks the availability of all discovered components and maintains statistics about their past availability. It listens for error and warning messages from all components that provide log files in the OS, middleware and application. Monere also captures information about the activities performed by the monitored application, such as actions performed by a BPEL process. This affords correlation of application activity with measurements taken at the system level via timestamps.

Performance indicators can be useful in diagnosing the cause of lower than expected performance, reveal network issues or show that a remote service has become overloaded. Examples include request latencies experienced by clients, execution times and throughput of invoked Web services, packet drop rates at the network interface and latencies between hosts. Object-relational mapping tools provide information and various statistics about transactions.

Resource usage metrics facilitate identification of components that have exceeded certain limits, but also deficiencies in the configuration of components, where a demanding workload may result in resource exhaustion. Monere measures OS resources, such as file descriptors, threads, memory and CPU utilisation

by load type. It measures the available swap space and the rate at which the OS makes use of the swap. Internal resource usage of Java-based components is obtained using JMX [22] and, as a last resort, Monere integrates metrics that can be obtained from command-line tools.

Measurement collection is driven by collection intervals defined for each metric and current intervals range from five to 60 seconds. Monitoring agents collect measurements through a variety of techniques. For example, Monere subscribes to the BPEL runtime to receive notifications of state changes of monitored processes. Request interception is another common pattern, in which a probe is placed between a client and a server. A third approach is polling, where a variety of means are used to periodically obtain measurements. For example, Monere queries relevant MBeans in JMX servers. The availability of some components can be checked by issuing HTTP HEAD requests or by querying the OS process table.

3.2 Overview

Monere is based on the open-source RHQ enterprise management system [3] and its JBoss extension Jopr [24]. RHQ provides a set of core services for systems management, such as an abstract inventory model and the discovery of hardware and software resources running on hosts based on a simple containment hierarchy. Monere modifies and extends RHQ in such a way as to make it suitable for monitoring cross-domain Web service compositions.

The three main components of Monere are shown in Figure 2. Each domain hosts a central Monere server, which communicates with the deployed agents in its domain. The server obtains resource discovery results and measurements from its agents, persists this in a database and then makes this data available to end users via a user interface. The agents, which are deployed on the hosts they monitor, register with their Monere server and then periodically examine their environment to discover resources and obtain measurements. A Monere

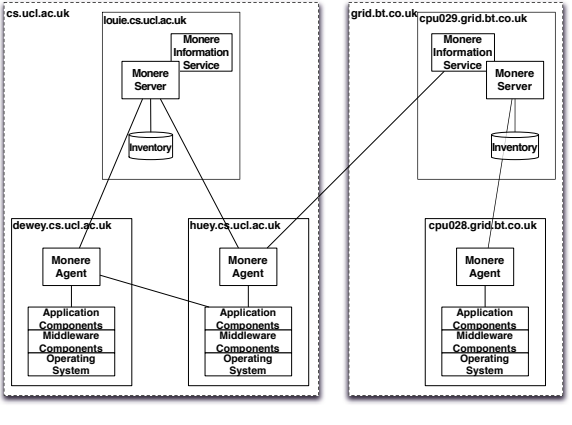


Fig. 2. The key components of Monere in a typical deployment

agent is essentially a runtime system for a set of plugins and manages communication with the Monere server. Plugins represent particular resource types and encapsulate the functionality for their discovery and the measurement of corresponding metrics. Plugins consist of a descriptive part expressed in XML and a Java implementation for the process of discovery and measurement collection. Examples of resources represented by plugins include a Tomcat server, an Axis SOAP runtime, the Web services deployed within it and the various components of a Linux OS.

The Monere Information Service (MIS) addresses the requirement of providing visibility of dependencies on resources in remote administrative domains and sharing information about their state. Service providers can make certain metrics about their published services available to clients. These metrics currently include the throughput and execution times of Web services. An agent that has discovered a dependency from one of its local resources onto a Web service in another administrative domain, can subscribe to an RSS feed at the corresponding MIS. The MIS restricts visibility to the level of published Web services and provides no further insight about the underlying infrastructure. It thus safeguards the sensitivity of implementation details, while providing data that can help to determine problems with remote services.

The Monere user interface has been built using Adobe Flex [1]. The UI is composed of a number of panels, each of which provides a particular view onto the available monitoring data. The Resource Tree view (Figure 3) provides an hierarchical view of all discovered components and the Dependency view displays the dependents and dependencies of a selected component along with basic information, such as its current availability. There are panels to display the activities performed by BPEL processes and report on any severe log messages from any part of the monitored system. The UI provides more detailed information on request. The Selected Resource Metrics view displays the metrics of any resource dragged onto it. The values are updated in real-time and data is aggregated. Charts of historical records help to identify trends and anomalies.

Fig. 3. Panels of the Monere UI showing dependencies of a selected resource across hosts and administrative domains

3.3 Dependency Discovery

Our dependency graph represents all relevant dependencies from the viewpoint of an application. This includes dependencies among application-level components, such as BPEL processes and Web services, across host and domain boundaries and dependencies on and among the lower-level components down to the operating system. Conceptually, Monere's dependency discovery process can be described in three phases. The first phase is concerned with the discovery of local components by each agent. An agent iterates over its deployed plugins and invokes their discovery interface implementations. Each plugin knows how to discover a corresponding component on the local host and returns a representation of it along with relevant attributes. We use a number of different mechanisms for discovery. Hyperic's SIGAR API [2], implements native libraries for a number of operating systems and can discover OS components, such as running processes, file systems and network interfaces. Another mechanism is to query JMX servers for specific MBeans that represent components of interest or parse deployment descriptors in the case of application servers. Some components offer administrative interfaces, which allow for programmatic queries on their properties and deployed components.

For each type of component, the RHQ plugin XML definition specifies what other types of components it is hosted by. For example, a BPEL process 'runs-in' a BPEL runtime. This information enables an agent to establish simple containment relationships between components. Another type of knowledge about dependencies is encapsulated within the implementation of some of the discovery code where, upon discovery of a component, an agent is instructed to parse specific configuration files to find dependencies on other middleware components. If two components are on the same host, the agent immediately establishes the dependency. Otherwise, the agent will insert a placeholder component into its inventory. This placeholder contains sufficient information about the remote component for the monitoring server to resolve it to the actual component discovered by another agent.

In the second phase, agents carry out a static analysis of the discovered application components on their hosts. BPEL processes explicitly identify the partner Web service interfaces with which they interact. Furthermore, WSDL documents import other WSDL service interfaces that they depend on. An agent begins by parsing the discovered BPEL processes to obtain information about the Web service interfaces used as partner services. The corresponding WSDL binding definitions are then parsed to link this abstract information about interfaces to the URL of the corresponding Web service instance. The process continues by parsing the documents of the identified interfaces and resolves further dependencies recursively. When an agent discovers a dependency on a Web service on another host or in a different administrative domain, it can insert a placeholder as the agent on the remote host is responsible for the corresponding part of the dependency graph.

In the final phase of the process, the agents submit their local inventories to the monitoring server. Upon submission by an agent, the server processes the

simple containment hierarchy and high-level dependencies among application-level components. It replaces any placeholder components with dependencies on the corresponding components from other hosts already committed to its inventory. This approach captures structural dependencies as opposed to functional ones obtained dynamically. In practice, the discovered dependencies represent functional ones well. However, a number of relationships are omitted, such as links to the backend implementations of Web services or the file system.

4 Performance Analysis

In order to determine the **impact on application performance**, we have executed 30 runs on the Polymorph service composition with and without monitoring enabled on real input data sets. We find a relative slowdown of about 8%, or a mean runtime of 199.23 seconds compared to 181.63 seconds. The slowdown is not high given the number of different components and metrics, but it is not negligible either. Our measurements of the **per-host overhead** imposed by monitoring agents over a 60 minute period are summarised in table 1. The agents monitor about 340 metrics each and, given the current set of collection intervals, each take about 500 to 600 measurements per minute. The number of components and metrics in this case is relatively large as is the variety of the measurement techniques used. As can be seen in table 1 the requirements on the CPU[1] and memory are low.

Table 1. Performance overhead of two monitoring agents

Ag	Components	Metrics	CPU	Heap Usage
1	66	341	3.2%	11.3MB range:5.7
2	90	349	4.2%	18.1MB range:6.9

Analysing **local communication overhead**, we can assume that there will always be new measurements to report on within the 30-second report interval, given that many of the metrics are continuous. Approximately, the size of measurement report (MR) data within an administrative domain is given by

$$a \times \prod_{i=0}^{n} \frac{30}{c_i} \times |metrics_i| \times k, \tag{1}$$

where a is the number of agents in an administrative domain, c_i is a particular collection interval, $metrics_i$ is the number of metrics at this collection interval and k is some constant for the size of a measurement. The smallest collection interval Monere supports is one second, which imposes an upper bound on the fraction of 30. As the collection intervals increase, we see that the cumulative size of exchanged MRs is given by the product of the number of agents and the number of metrics, with the latter one likely to be the dominant factor.

[1] A single core Pentium 4 3.2 GHz.

Measurements performed with *tcpdump* over a period of 30 minutes reveal a bit rate within an administrative domain of 3.06Kbits/s and 0.48Kbits/s for the exchange between administrative domains.

5 Experiment

The controlled experiment examines the performance of 22 human operators tasked with the diagnosis of a number of typical system-level failures in order to quantify the effects of Monere on diagnosis times.

5.1 Experiment Setup

We have replicated the Polymorph workflow and its infrastructure on a testbed with two separate domains, as shown in Figures 1 and 2. Each participant was asked to diagnose six failures injected into the system. The failures were injected in random order and participants were given 10 minutes per failure. The stated goal was to determine the root component(s) of the failure and identify what about its behaviour is likely to have caused the observed problem. Participants were randomly assigned to use either the Monere prototype or a standard set of tools. Both groups were given a brief written overview of the Polymorph service composition and the toolset to be used. The control group's tools consisted of root shells onto the hosts in the local domain, JConsoles connected to the local Tomcat JVMs, pointers to log files and a brief explanation of Condor shell commands. Control group participants furthermore had access to a session of the ActiveBPEL monitoring console [4], which provides an overview of the state of executing BPEL processes. Monere participants had only access to Monere's web-based UI.

Each group consisted of 11 participants. The participants were volunteers from among the post-docs and faculty of several CS departments and also included a small number of professional software engineers. The level of experience of each participant was determined through a questionnaire. The levels of experience between the groups are well-balanced, but we omit a closer analysis of this aspect for space reasons.

The failures were selected among a larger set of failures as they were observed with the Polymorph composition running in production. Injection was through a script-driven framework we have developed to reliably reproduce the conditions that lead to the six failures. These failures are:

1. unavailability of a remote service without known cause
2. slowdown of a remote service caused by overload
3. application server failure caused by thread exhaustion
4. application server failure caused by heap exhaustion
5. unavailability of the grid scheduler due to disk space exhaustion
6. failure to schedule compute jobs in a timely manner due to overload

5.2 Results

Success Rate. We examine the proportion of successfully diagnosed failure causes and find that Monere participants achieve a success rate of 95% on average, while the Control group only diagnosed 72% of the presented failures correctly. For the Control group, the two most frequently misdiagnosed failures are the ones affecting the Web service hosted in the remote administrative domain. As such, this is not a surprising result. However, it does serve to demonstrate the usefulness of cross-domain dependency graphs, which enable operators to more quickly identify when an issue lies outside the local infrastructure. The most frequently misdiagnosed failure for the Monere group was failure 5. This highlights a weakness of structural dependency graphs, where some of the existing functional dependencies (i.e. between Condor and the file system) may not be represented explicitly.

Diagnosis Time. Next, we examine whether there is a statistically significant difference in the observed diagnosis times. We refer to the mean times to diagnose the six failures as MTT_{diag}.

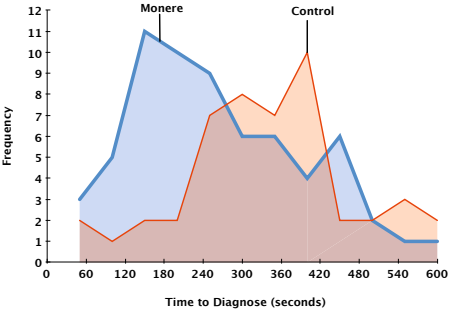


Fig. 4. Histogram of the diagnosis times for the Monere and the Control group

Table 2. Summary of testing for a significant reduction in mean diagnosis times

	MTT_{diag}	StdDev	n	t_0	$t_{0.05,109}$
Monere	241.55s	132.34	63	-2.80	1.659
Control	311.96s	129.71	48		

Figure 4 is a histogram of the diagnosis times of each group and table 2 summarizes our observations. The histogram suggests that operators using Monere achieve shorter diagnosis times, but there is also a cluster of observations that reveals a higher number of almost seven-minute diagnosis times. The MTT_{diag} for the Monere group is 241.55 seconds. The participants in the Control group diagnosed the same failures with a MTT_{diag} of 311.96 seconds. We perform a one-sided t-test at the 95% confidence level in order to determine whether the diagnosis times achieved by Monere are indeed shorter with statistical significance.

For this, we need to confirm $t_0 \leq t_{0.05,109}$. The degrees of freedom are obtained from the number of observations $(63 + 48 - 2 = 109)$. As the resulting values show, $-2.80 < 1.659$. This allows us to reject the null hypothesis in favour of the alternative hypothesis, which states that participants using Monere do indeed achieve shorter diagnosis times. We find an average reduction in MTT_{diag} with Monere of 22.5%.

Failures. We were are able to identify some failures with noticeably shorter and longer median diagnosis times than the rest. For the Control group, the failure with the highest median diagnosis time is failure 1 (remote service unavailable) and the lowest times are achieved for failure 5 (Condor unavailable). As table 3 illustrates, these failures are exactly reversed for the Monere group.

Table 3. Failures with longest/shortest median diagnosis times

	Control	Monere
Longest	Failure 1 (419s)	Failure 5 (430s)
Shortest	Failure 5 (264s)	Failure 1 (76s)

The relatively long diagnosis times for failure 1 are due to lack of tool support that provides adequate visibility of critical components in separate administrative domains. This forces Control group participants to examine many components in the local infrastructure before focussing their attention on the remote dependencies. Control group participants identified failure 5 relatively quickly as upon issuing one of the available Condor commands, they would immediately learn that the Condor process was down, which prompted them to inspect its log file and based on that examine the available disk space. Monere participants had much more difficulty identifying this problem as the available dependency graph did only reveal a dependency between Condor and the OS. This left participants with a larger problem space to explore. Conversely, for failure 1 the dependency graph enabled participants to quickly identify that they were dealing with an issue outside their local domain.

5.3 A Short Cost-Benefit Analysis

So when is the investment in online monitoring actually justified? To answer this question, a service provider would need to determine two things. First, one needs to have an accurate idea of the downtime cost per unit of time for a particular service offering. For cross-domain systems, this information should be available at the time of writing an SLA in the form of financial penalties associated with the violation of promised service level agreements [25]. Second, it is necessary to estimate the typical proportion of diagnosis time on overall downtime for a service implementation and its system-level failures. This information could be obtained either through an organisational baseline or, once available, from detailed measurement-based failure studies on large service compositions.

In general, we can conclude that an approach such as Monere will only be worth it when applied to systems whose overall failure repair times are clearly dominated by their diagnosis times and where the financial penalties or reputational loss for performing below agreed service levels are substantial.

5.4 Validity

Ideally, we would have liked to install Monere on an application in production and compare the performance of system operators to an organisational baseline. As this is not feasible outside an industrial setting, we rely on a single service composition and limit the number of failures to six. This limits the generalizability of our results to some extent. However, the Polymorph composition is a real-world application of moderate size and complexity and has been executed under realistic workloads. It encompasses most of the challenging characteristics we are interested in. Furthermore, the selected failures are not trivial to diagnose. The internal validity is strong as each participant uses only one of the tool sets and is presented with each failure only once and in random order. The measures used (i.e. success rate, diagnosis time) are objective ones. The number of data points on diagnosis times are sufficient, even though they could be improved on. The failure injection method we employ does not merely simulate the symptoms of a failure, but reproduces the actual conditions in the system that lead to it.

6 Related Work

6.1 Dependency Discovery

Indirect approaches to dependency discovery trace some aspect of system operation and perform statistical analysis in order to determine likely dependencies. In [15] readily available performance data, such as request counts and times, are measured in order to to determine containment of activity periods of transactions between pairs of nodes. The frequency of such containments between the same pairs of nodes gives an indication of the strength of their relationship. The rate of false dependencies increases with the rate of concurrency in the system. The addition of a statistical model to estimate false positives, [17] improves the accuracy of dynamic dependency graphs. Their approach needs to intercept messages that pass network devices. In Pinpoint [13] middleware services within a J2EE server are instrumented to trace the traversal of requests and learn likely call paths.

The main advantages of indirect approaches are that they discover functional dependencies and rely primarily on instrumentation at the middleware and network layer. Monere discovers structural dependencies. The resulting dependency graph may contain relationships that become active only rarely. Nevertheless, these structural dependencies seem to correspond well to functional ones and do not require a large number of observations under varying workloads. Even though some manual effort is required for the expression of model information,

this knowledge is expressed modularly (i.e. per component type) and can be reused among many deployments. Finally, Monere does not rely on any instrumentation beyond what is already made available by the OS and discovered components.

6.2 Monitoring

Monere differs from much work on SOC monitoring ([6], [21]) in that it does not primarily focus on the composition- and service-level and associated higher-level events. Instead, it regards these as one of several types of components and events to be integrated from all relevant layers of a middlware-based distributed system. Magpie [7] obtains control paths and resource usage for the threads involved in servicing requests by instrumenting the Windows NT kernel. It records events at points where control flow transfers among components and uses request schemas to associate events with observed requests. vPath [26] is another approach, which does not require propagation of request identifiers. It instruments virtual machine monitors to capture thread activity and TCP system calls. Assuming synchronous request-reply invocations and a dispatch-worker thread model, it can link recorded activity and requests.

These approaches are quite elegant in that they exploit observable events that occur in response to end-user requests to infer resulting system activity. Monere provides an evidence-based determination of the impact of online monitoring at all layers of a service composition on diagnosis times and shows under what circumstances this is actually justified.

Several approaches focus on tracing events at the kernel-level. Chopstix [8] instruments the operating system kernel to collect measurements about process scheduling, I/O activity, system calls and socket communication. Two other approaches that avoid instrumentation above the level of the operating systems are DTrace [11] and SysProf [5]. DTrace enables instrumentation of user-level and kernel-level events through user-specified probes. SysProf instruments the kernel to record resource consumption by capturing context switches and system calls.

Observing events at the kernel-level provides visibility of everything that happens in a system, requires only a small number of measurement techniques and affords instrumentation of legacy applications. A downside is that operators are left to figure out how kernel-level events relate to particular application-level activities. Monere demands considerable development effort given the variety of required measurement techniques. Its advantages are that it produces more familiar metrics, makes it easier to understand how parts of the system react to particular application activity and should be easier for a broader range of developers to extend.

7 Conclusions

We have presented the results and insights from an empirical evaluation of the effect of monitoring readily available data about system operation on mean

diagnosis times for system-level failures in large distributed systems. We have developed a prototype of such an approach called Monere and given an overview of its architecture and measurement techniques. We have also shown how it is possible to obtain dependency graphs through modular models and by exploiting the structure inherent to BPEL workflows. A brief analysis reveals that such an approach will outweigh its performance cost in cases where diagnosis time is the dominant factor of overall system downtime and where the financial penalties associated with the violation of SLAs are substantial. Our results also provide some insights about the utility of dependency graphs and how missing or inaccurate relationships can be so misleading during diagnosis as to reduce an operator's performance to below that of someone not having access to such a graph.

One key lesson from our investigation is that the collection of a wide array of measurements about system operation is not in itself sufficient to drastically reduce diagnosis times in most scenarios. Measuring aspects of system operation is important in order to keep applications operational. However, availability of rich repositories of data, rather than being the solution are a prerequisite for providing higher-value added functionality. In our view, a promising direction to improve the handling of unanticipated system-level failures in large Web service compositions is to investigate automated mechanisms that can determine anomalies and other areas of interest within a large volume of data and thereby effectively reduce the problem space on behalf of system operators.

References

1. Adobe Flex, `http://bit.ly/2DbkE9`
2. Hyperic SIGAR API, `http://bit.ly/96BIG3`
3. RHQ, `http://bit.ly/apijCR`
4. ActiveBPEL (2010), `http://bit.ly/be87LF`
5. Agarwala, S., Schwan, K.: Sysprof: Online distributed behavior diagnosis through fine-grain system monitoring. In: ICDCS (July 2006)
6. Baresi, L., Guinea, S.: Self-supervising bpel processes. IEEE TSE 37, 247–263 (2011)
7. Barham, P., Donnelly, A., Isaacs, R., Mortier, R.: Using magpie for request extraction and workload modelling. In: OSDI. USENIX, Berkeley (2004)
8. Bhatia, S., Kumar, A., Fiuczynski, M.E., Peterson, L.: Lightweight, high-resolution monitoring for troubleshooting production systems. In: OSDI. USENIX, Berkeley (2008)
9. Birman, K., van Renesse, R., Vogels, W.: Adding high availability and autonomic behavior to web services. In: ICSE. IEEE CS, Washington, DC, USA (2004)
10. Box, D., et al.: Simple Object Access Protocol (SOAP 1.1) (May 2000)
11. Cantrill, B.M., Shapiro, M.W., Leventhal, A.H.: Dynamic instrumentation of production systems. In: ATC. USENIX, Berkeley (2004)
12. Chandra, A., Prinja, R., Jain, S., Zhang, Z.: Co-designing the failure analysis and monitoring of large-scale systems. SIGMETRICS Perform. Eval. Rev. 36 (August 2008)
13. Chen, M.Y., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.: Pinpoint: problem determination in large, dynamic internet services. In: DSN. IEEE (2002)

14. Emmerich, W., Butchart, B., Chen, L., Wassermann, B., Price, S.L.: Grid Service Orchestration using the Business Process Execution Language (BPEL). JOGC 3(3-4), 283–304 (2005)
15. Gupta, M., Neogi, A., Agarwal, M.K., Kar, G.: Discovering Dynamic Dependencies in Enterprise Environments for Problem Determination. In: Brunner, M., Keller, A. (eds.) DSOM 2003. LNCS, vol. 2867, pp. 125–166. Springer, Heidelberg (2003)
16. Jordan, D., et al.: Web Services Business Process Execution Language 2.0 WS-BPEL (August 2006)
17. Kashima, H., Tsumura, T., Ide, T., Nogayama, T., Hirade, R., Etoh, H., Fukuda, T.: Network-based problem detection for distributed systems. In: ICDE. IEEE CS, Washington, DC, USA (2005)
18. Katchabaw, M., Howard, S., Lutfiyya, H., Marshall, A., Bauer, M.: Making distributed applications manageable through instrumentation. In: Proc., 2nd Intl. Workshop on SEPDS 1997 (May 1997)
19. Lee, W., McGough, S., Newhouse, S., Darlington, J.: A standard based approach to job submission through web services. In: Cox, S. (ed.) Proc. of the UK e-Science All Hands Meeting, Nottingham, pp. 901–905. UK EPSRC (2004) ISBN 1-904425-21-6
20. Litzkow, M., Livny, M., Mutka, M.: Condor - A Hunter of Idle Workstations. In: ICDCS (June 1988)
21. Moser, O., Rosenberg, F., Dustdar, S.: Event Driven Monitoring for Service Composition Infrastructures. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 38–51. Springer, Heidelberg (2010)
22. Perry, J.S.: Java Management Extensions, 1st edn. O'Reilly & Associates, Inc., Sebastopol (2002)
23. Plattner, B.: Real-time execution monitoring. IEEE TSE 10(6), 756–764 (1984)
24. Red Hat, Inc.: Jopr. `http://www.jboss.org/jopr`
25. Skene, J., Raimondi, F., Emmerich, W.: Service-level agreements for electronic services. IEEE TSE 36(2), 288–304 (2010)
26. Tak, B.C., Tang, C., Zhang, C., Govindan, S., Urgaonkar, B., Chang, R.N.: vpath: precise discovery of request processing paths from black-box observations of thread and network activities. In: ATC. USENIX, Berkeley (2009)
27. Vogels, W.: World wide failures. In: Proc. of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications. ACM, New York (1996)
28. Vogels, W., Re, C.: Ws-membership - failure management in a web-services world. In: WWW (2003)

Multi-layered Monitoring and Adaptation*

Sam Guinea[1], Gabor Kecskemeti[2], Annapaola Marconi[3], and Branimir Wetzstein[4]

[1] Politecnico di Milano
Deep-SE Group - Dipartimento di Elettronica e Informazione
Piazza L. da Vinci, 32 - 20133 Milano, Italy
guinea@elet.polimi.it
[2] MTA-SZTAKI
Laboratory of Parallel and Distributed Systems
Kende u. 13-17, 1111 Budapest, Hungary
kecskemeti@sztaki.hu
[3] Fondazione Bruno Kessler
via Sommarive 18, 38123 Trento, Italy
marconi@fbk.eu
[4] University of Stuttgart
Institute of Architecture of Application Systems
Universitaetsstr. 38, 70569 Stuttgart, Germany
wetzstein@iaas.uni-stuttgart.de

Abstract. Service-based applications have become more and more multi-layered in nature, as we tend to build software as a service on top of infrastructure as a service. Most existing SOA monitoring and adaptation techniques address layer-specific issues. These techniques, if used in isolation, cannot deal with real-world domains, where changes in one layer often affect other layers, and information from multiple layers is essential in truly understanding problems and in developing comprehensive solutions.

In this paper we propose a framework that integrates layer specific monitoring and adaptation techniques, and enables multi-layered control loops in service-based systems. The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

1 Introduction

Service-based systems are built under an open-world assumption. Their functionality and quality of service depend on the services they interact with, yet these services can evolve in many ways, for better or for worse. To be sure these evolutions do not lead to systems that behave inadequately or fail, service-based

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (Network of Excellence S-Cube) and grant agreement 216556 (SLA@SOI).

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 359–373, 2011.

systems must be able to re-arrange themselves to cope with change. A typical way of dealing with these issues is to introduce some variant of the well-known monitor-analyze-plan-execute (MAPE) loop into the system [6], effectively making the system self-adaptive.

The service abstraction has become so pervasive that we are now building systems that are multi-layered in nature. Cloud-computing allows us to build software as a service on top of a dynamic infrastructure that is also provided as a service (IaaS). This complicates the development of self-adaptive systems because the layers are intrinsically dependent one of the other. Most existing SOA monitoring and adaptation techniques address one specific functional layer at a time. This makes them inadequate in real-world domains, where changes in one layer will often affect others. If we do not consider the system as a whole we can run into different kinds of misjudgments. For example, if we witness an unexpected behavior at the software layer we may be inclined to adapt at that same layer, even though a more cost-effective solution might be found either at the infrastructure layer, or by combining adaptations at both layers. Even worse, a purely software adaptation might turn out to be useless due to infrastructural constraints we fail to consider. Similar considerations are made in case of the unexpected behavior at the infrastructure layer, or at both.

In this paper we propose a framework that integrates software and infrastructure specific monitoring and adaptation techniques, enabling multi-layered control loops in service-based systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a coordinated fashion. In our prototype we have focused on the monitoring and adaptation of BPEL processes that are deployed onto a dynamic infrastructure.

Building upon our past experiences we have integrated process and infrastructure level monitoring [2,8] with a correlation technique that makes use of complex event processing [1]. The correlated data, combined with machine-learning techniques, allow us to pinpoint where the problems lie in the multi-layered system, and where it would be more convenient to adapt [7,12]. We then build a complex adaptation strategy that may involve the software and/or the infrastructure layer [13], and enact it through appropriate effectors.

The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

The rest of this paper is organized as follows. Section 2 gives a high-level overview of the integrated monitoring and adaptation framework used to enable the multi-layered control loops. Section 3 details the software and infrastructure monitoring tools and how they are correlated using complex event processing. Section 4 explains how decision trees are used to identify which parts in the system are responsible for the anomalous behaviors and what adaptations are needed. Section 5 explains how we coordinate single-layer adaptation capabilities to define a multi-layered adaptation strategy, while Section 6 presents the tools used to actually enact the adaptations. Section 7 evaluates the integrated

approach on a medical imaging procedure. Section 8 presents related work, and Section 9 concludes the paper.

2 The Integrated Monitoring and Adaptation Framework

We propose an integrated framework that allows for the installation of multi-layered control loops in service-based systems. We will start with a conceptual overview, and then provide more details on the single techniques we have integrated in our prototype.

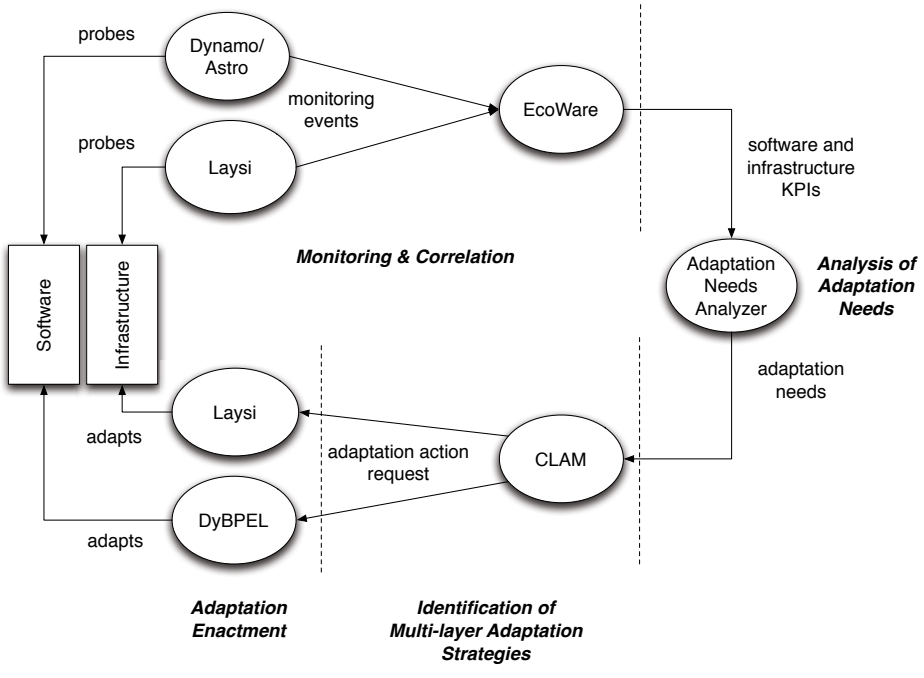


Fig. 1. The Monitoring and Adaptation Framework

Figure 1 gives a high-level view of our integrated monitoring and adaptation framework, as used in a multi-layered software and infrastructure system. To establish self-adaptation, the framework applies a slight variation of the well-known MAPE control loop. Dashed vertical lines separate the four main steps in the loop, while oval shapes represent the concrete techniques that we have integrated – detailed later in Sections 3–6.

In the *Monitoring and Correlation* step, sensors deployed throughout the system capture run-time data about its software and infrastructural elements. The collected data are then aggregated and manipulated to produce higher-level correlated data under the form of general and domain-specific metrics. The main

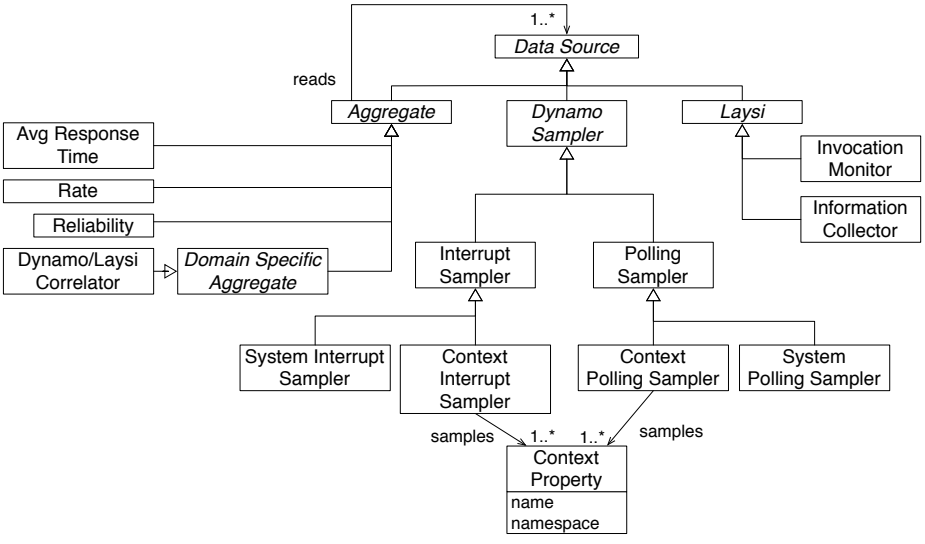


Fig. 2. The Monitoring and Correlation Model

goal is to reveal correlations between what is being observed at the software and at the infrastructure layer to enable global system reasoning.

In the *Analysis of Adaptation Needs* step, the framework uses the correlated data to identify anomalous situations, and to pinpoint and formalize where it needs to adapt. It may be sufficient to adapt at the software or at the infrastructure layer, or we may have to adapt at both.

In the *Identification of Multi-layer Adaptation Strategies* step, the framework is aware of the adaptation capabilities that exist within the system. It uses this knowledge to define a multi-layer adaptation strategy as a set of software and/or infrastructure adaptation actions to enact. A strategy determines both the order of these actions and the data they need to exchange to accomplish their goals.

In the *Adaptation Enactment* step, different adaptation engines, both at the software and the infrastructure layer, enact their corresponding parts of the multi-layer strategy. Each engine typically contains a number of specific modules targeting different atomic adaptation capabilities.

3 Monitoring and Correlation

Monitoring consists in collecting data from a running application so that they can be analyzed to discover runtime anomalies; event correlation is used to aggregate runtime data coming from different sources to produce information at a higher level of abstraction. In our integrated framework we can obtain low-level data/events from the process or from the context of execution using Dynamo [2], or from the infrastructure using Laysi [8]. We can then manipulate the data to

obtain higher-level information using the event correlation capabilities provided by EcoWare [1]. Figure 2 gives an overview of the kind of data sources available through Dynamo, Laysi, and EcoWare.

Dynamo provides means for gathering events regarding either (*i*) a process' internal state, or (*ii*) context data[1]. `Interrupt Samplers` interrupt a process at a specific point in its execution to gather the information, while `Polling Samplers` do not block the process but gather their data through polling.

The `Invocation Monitor` is responsible for producing low-level infrastructure events through the observation of the various IaaS systems managed by Laysi. These events signal a service invocation's failure or success, where failures are due to infrastructure errors. The infrastructure, however, can also be queried through the `Information Collector` to better understand how services are assigned to hosts. The differences between the utilized infrastructures and the represented information are hidden by the information collector component of the MetaBroker service in Laysi.

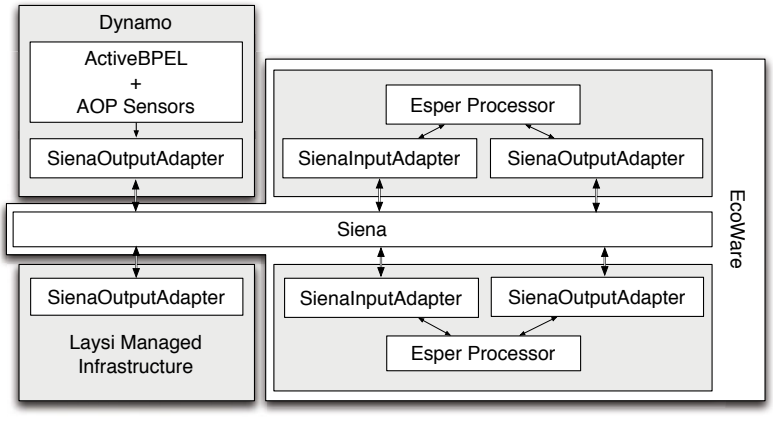


Fig. 3. The Dynamo and EcoWare Architecture

The events collected through Dynamo and Laysi can be further aggregated or manipulated by EcoWare. We can use a predefined aggregate metric such as `Reliability`, `Average Response Time`, or `Rate`, or we can use a domain-specific aggregate whose semantics is expressed using the Esper event processing language. Aggregates process events coming from one or more data sources and produce new ones that can be even further manipulated in a pipe-and-filter style.

For our integrated approach we developed a domain-specific aggregate called the `Dynamo/Laysi Correlator` to correlate events produced at the software and the infrastructure layers. This component exploits a correlation data set that is artificially introduced by Dynamo in every service call it makes to the Laysi infrastructure. The correlation data contains the name of the process making

[1] We intend as context any data source, external to the system, that offers a service interface.

the call to Laysi, the invocation descriptor in the form of a unique JSDL (Job Submission Description Language) document, and a unique ID for the process instance that is actually making the request. These data are also placed within the events that are generated by the Invocation Monitor, allowing EcoWare to easily understand which software- and infrastructure-level events are related. Figure 3 gives an overview of the technical integration of Dynamo, Laysi, and EcoWare, which is achieved using a Siena publish and subscribe event bus. Input and output adapters are used to align Dynamo, Laysi, and the event processors with a normalized message format.

4 Analysis of Adaptation Needs

Monitoring and correlation produce simple and complex metrics that need to be evaluated. A Key Performance Indicator consists of one of these metrics (e.g., overall process duration) and a target value function which maps values of that metric to two or more categories on a nominal scale (e.g., "process duration < 3 days is *good*, otherwise *bad*" defines two KPI categories). These KPI categories allow us to interpret whether, and how, KPI metric values conform to business goals. If monitoring shows that many process instances have bad KPI performance, we need to (*i*) analyze the influential factors that lead to these bad KPI values, and (*ii*) find adaptation actions that can improve those factors and thus the KPI. Figure 4 shows an overview of the KPI-based Adaptation Needs Analyzer Framework [7,12] and its relation to the overall approach. It consists of two main components: an `Influential Factor Analysis` component and an `Adaptation Needs Analysis` component.

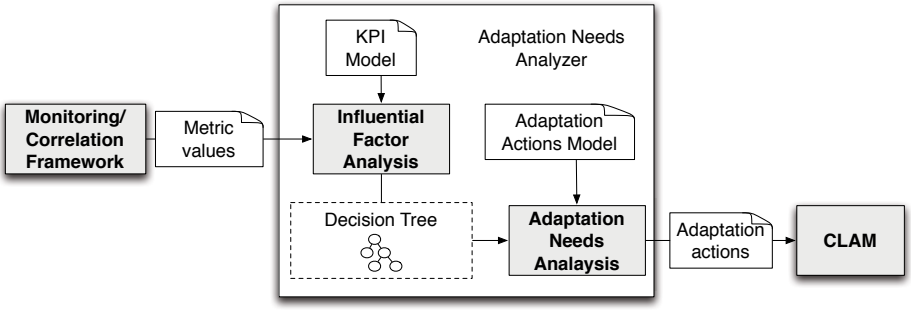


Fig. 4. Adaptation Needs Analysis Framework

The Influential Factor Analysis receives the metric values for a set of process instances within a certain time period. In this context, interesting metrics are measured both on the process level and the service infrastructure level. At the process level, metrics include the durations of external service calls, the duration of the overall business process, the process paths taken, the number of iterations

in loops, and the process' data values. Service infrastructure metrics describe the service invocation properties which include the status of the service invocation (successful, failed), and properties such as the infrastructure node on which the service execution has been performed.

It uses machine learning techniques (decision trees) to find out the relations between a set of metrics (potential influential factors) and the KPI category based on historical process instances [12]. The algorithm is fed with a data set, whereby each data item in this set represents one process instance and the values of all the metrics that were measured for that instance and the KPI category that has been evaluated. The algorithm creates a decision tree in which nodes represent metrics (e.g., the duration of a particular activity), outgoing edges represent conditions on the values of the metric, and leaves represent KPI categories. By following the paths from the root of the tree to its leaves, we can see for which combinations of metrics and values particular KPI categories have been reached (e.g., if duration of activity A was above 3 hours and activity B was executed on node 2 the KPI value was bad).

Based on this analysis the next step is to use the Adaptation Needs Analysis component to identify the adaptation needs, i.e., what is to be adapted in order to improve the KPI [7]. The inputs to this step are the decision tree and an adaptation actions model which has to be manually created by the user. The model contains different adaptation actions, whereby each specifies an adaptation mechanism (e.g., service substitution, process structure change) and how it affects one or more of the metrics used in the Influential Factor Analysis. For example, an adaptation action could be to substitute service A in the process with service B, and its effect could be "service response time < 2 h". The Adaptation Needs Analysis extracts the paths which lead to bad KPI categories from the tree and combines them with available adaptation actions which can improve the corresponding metrics on the path. As a result, we obtain different sets of potential adaptation actions. However, each of these sets does not yet take cross-layer dependencies between adaptation actions into account. This is performed in the next step by the CLAM framework.

5 Identification of Multi-layer Adaptation Strategies

The main aim of the Cross Layer Adaptation Manager (CLAM) [13] is to manage the impact of adaptation actions across the system's multiple layers. This is achieved in two ways: on the one hand CLAM identifies the application components that are affected by the adaptation actions, and on the other hand, it identifies an adaptation strategy that properly coordinates the layer-specific adaptation capabilities. CLAM relies on a model of the multi-layer application that contains the current configuration of the application's components (e.g. business processes with KPIs, available services with stated QoS and general information, available infrastructure resources) and their dependencies (e.g. business activity A is performed by service S). When the CLAM identifies the components that are affected by the adaptation actions, it uses a set of `checkers`,

each associated with a specific application concern (e.g. service composition, service performances, infrastructure resources), to analyze whether the updated application model is compatible with the concern's requirements. The goal is to produce a strategy that is modeled as an Adaptation Tree. The tree's root represents the model's initial configuration; its other nodes contain the configurations of the model, as updated by the adaptation actions, and the checkers that need to be invoked at each step; its edges represent the outcome of the invoked checkers.

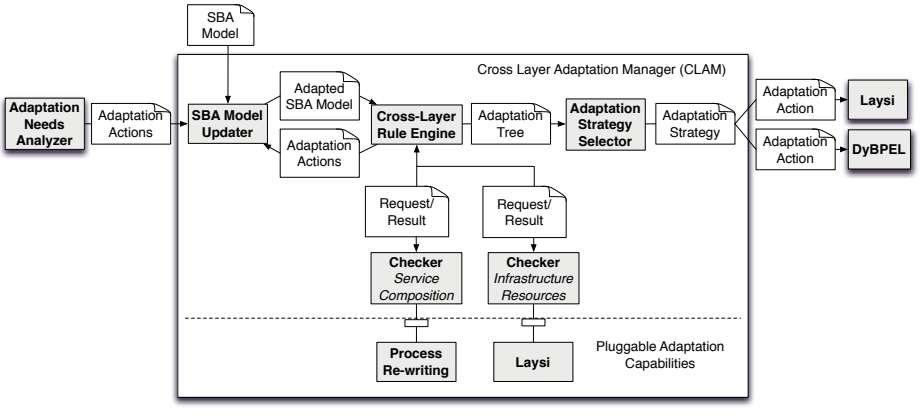


Fig. 5. CLAM: Cross-layer Adaptation Manager

Figure 5 presents an overview of CLAM's architecture. Whenever a new set of adaptation actions is received from the Adaptation Needs Analyzer, the `SBA Model Updater` module updates the current application model by applying the received adaptation actions. CLAM requires that all the adaptation actions be applicable with respect to the current model. However, this is guaranteed in the proposed multi-layer framework by the Adaptation Needs Analyzer.

The adapted model is then used by the `Cross-layer Rule Engine` to detect the components in the layers affected by the adaptation and to identify, through the set of predefined rules, the associated adaptation checkers. If some constraints are violated, the checker is responsible for searching for a local solution to the problem. This analysis may result in a new adaptation action to be triggered. This is determined through the interaction with a set of pluggable application-specific adaptation capabilities.

The Cross-layer Rule Engine uses each checker's outcome to progressively update the strategy tree. If the checker triggers a new adaptation action, the Cross-layer Rule Engine obtains a new adapted model from the Model Updater, and adds it as a new node to the strategy tree, together with the new checkers to be invoked. If the checker reports that the adaptation is not compatible and that no solution can be found, the node is marked as a red leaf; the path in the

tree that leads from the root to that specific node represents an unsuccessful strategy. On the contrary, if all checks complete successfully, the node is a green leaf that can be considered a stable configuration of the application, and the corresponding path in the tree represents an adaptation strategy that can be enacted.

If multiple adaptation strategies are identified, the `Adaptation Strategy Selector` is responsible of choosing the best strategy by evaluating and ranking the different strategies according to a set of predefined metrics. The selected strategy is then enacted passing the adaptation actions to the adaptation enactment tools.

Due to our scenario's requirements we have currently integrated two specific adaptation capabilities into our framework: the `Process Re-Writing` planner, responsible of optimizing service compositions by properly parallelizing sequential activities, and `Laysi`, whose aim is to guarantee a correct and optimized usage of infrastructure resources.

The Process Re-writing Planner is an adaptation mechanism that, given a BPEL process and a set of optimization requirements, automatically computes a new version of the process that maximizes the parallel execution of activities. This is done taking into account a set of data and control flow requirements that characterize the process' correct behavior (e.g. activity A cannot be executed in parallel with B, activity A must follow activity B, etc.), as well as any interaction protocols the partner services may require (e.g. if service S expects activity A to be executed before activity B, than this protocol requirement will be satisfied).

Laysi offers self-management capabilities for service infrastructures and allows new infrastructure level requirements to be evaluated before the actual service invocations take place. Hence, upon receiving the possible parallel execution options from Process Re-writing Planner, the CLAM architecture presents these options as requirements (including the required parallelism and time constraints) to Laysi for all the not-yet executed service calls. In response, Laysi determines the feasibility of the proposed requirements taking into account that a rearrangement of the service infrastructure may be needed. If the system decides to enact the adaptation and the infrastructure needs to be rearranged, Laysi will ensure the next invocation can meet its agreed constraints according to the adaptation enactment tasks specified in the following section.

6 Adaptation Enactment

In our integrated approach we enact software adaptations through `DyBPEL`, and infrastructure adaptations through `Laysi`. CLAM issues specific actions of the chosen adaptation strategy to each tool in a coordinated fashion.

In the proposed integration, DyBPEL is responsible of enacting the process restructuring adaptations identified by the Process Re-writing Planner.

DyBPEL extends an open-source BPEL execution engine (ActiveBPEL) with the capability to modify a process' structure at run time. The change can be applied to a single process instance or to an entire class of processes. DyBPEL

consists of two main components: a `Process Runtime Modifier`, and a `Static BPEL Modifier`. The runtime modifier makes use of AOP techniques to intercept a running process and modify it in one of three ways: by intervening on its BPEL activities, on its set of partnerlinks, or on its internal state. The runtime modifier takes three parameters. The first is an XPath expression that uniquely identifies the point in the process execution in which the restructuring has to be activated. The second is an XPath expression that uniquely identifies the point in the process in which restructuring needs to be achieved (it can be different than the point in which the restructuring is activated). The third is a list of restructuring actions. Supported actions consist of the addition, removal, or modification of BPEL activities, partnerlinks, and data values. When dealing with BPEL activities we must provide the BPEL snippet that needs to be added to the process, or used to modify one of the process' existing activities. When dealing with partnerlinks we must provide the new partnerlink that needs to be added to the process, or used to modify an existing one. When dealing with the process' state we must uniquely identify a BPEL variable within the process to be added or modified, and the XML snippet that will consist of its new value.

When the process restructuring needs to be more extensive, we can use the static BPEL modifier. It supports the same kinds of modifications to the process' activities, partnerlinks, and internal variables, except that the modifications are performed on the process' XML definition. This operation is completely transparent to users. First of all, already running instances are not modified and changes are only applied to new instances. Second, using the same endpoint, all new process requests are forwarded to the newly deployed version of the process.

Regarding infrastructure adaptation, Laysi always performs service requests on a best-effort basis. Each service invocation is handled individually and the various calls are assumed to be independent. Consequently, the performance of the service requests might not be aligned with the higher layers of the service-based system. To provide better alignment with the service composition layer we can specify special constraints about service placement (e.g. service instance A should be hosted within the same provider as service instance B) and availability within the infrastructure (e.g. a service instance should be available before the invocation request is placed in the call queue of Laysi). These constraints are derived directly from the business process and the future interactions between the available service instances hosted by the infrastructure. Laysi constructs the service infrastructure on five layers: meta negotiators, meta brokers, service brokers, automatic service deployers, and the physical infrastructures (grid resources or cloud based virtual machines). These infrastructure layers autonomously adapt themselves to the placed constraints (e.g. placement, availability, CPU, memory, pricing). The autonomous behavior of the infrastructure may involve (*i*) new service instance deployment in high demand situations, (*ii*) service broker replacement in case of broken or low performing physical infrastructures, and/or (*iii*) negotiation bootstrapping if a new negotiation technique is required.

7 The CT Scan Scenario

The application domain considered in this paper concerns the medical imaging procedure for Computed Tomography (CT) Scans. A CT Scan is an X-ray based medical test that, exploiting sophisticated image processing algorithms, produces cross-sectional images of the inside of the body. These images can be further processed to obtain three dimensional views.

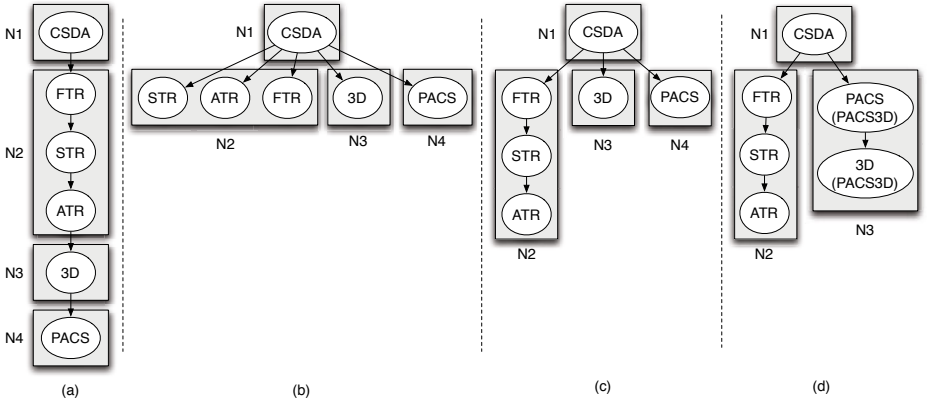


Fig. 6. Evolution of the CT scan scenario

Figure 6(a) describes the typical CT Scan process. White ovals represent software services, while gray rectangles tell us the infrastructure nodes hosting them. During the Cross Sectional Data Acquisition phase (service `CSDA`) the CT scanner acquires X-ray data for individual body cross sections depending on which parts of the body need to be scanned. These data are then used by complex image processing services (offered by various hosts in the infrastructure) to obtain a set of cross-sectional images from different perspectives as well as 3D volumetric information. The services are the Frontal Tomographic Reconstruction service (`FTR`), the Sagittal Tomographic Reconstruction service (`STR`), the Axial Tomographic Reconstruction service (`ATR`), and the 3D volumetric information service (`3D`). Finally, the data is stored to a picture archiving and communication system using the `PACS` service.

These activities require enormous processing power. To keep costs down, the hospital only maintains the resources needed for emergency CT scans. During burst periods, such as during the public opening hours of the CT laboratory, it relies on an infrastructure dynamically extensible with virtual machines from IaaS cloud infrastructures managed by Laysi.

In the following we show how our approach can be used to automatically adapt this multi-layered system. The CT Scan process is initially designed by a domain expert on the basis of the common medical procedure. The obtained process is a simple sequence of actions that does not embed any optimization with

respect to its performance (Figure 6 (a)). The domain expert also specifies his goals for the quality of the medical procedure using a set of KPIs. For instance, an important goal is to ensure that the processing time of a CT scan does not rise above 60 minutes. An advanced user, such as a hospital IT technician, defines a set of adaptation actions that can be used to improve the process' performance: (*i*) the parallelization of process activities; (*ii*) the substitution of some services (for example, the use of a more costly PACS3D service capable of substituting both services PACS and 3D); (*iii*) the deployment of a service onto a new infrastructural node with specific characteristics.

At run time we collect monitoring events both at the software and the infrastructure level and correlate them using EcoWare. After a certain period of time, we notice that the CT process' performance is degrading: in the last 400 scans about 25% have not achieved their desired overall CT scan processing time. In order to identify the reasons for this behavior, the Influential Factor Analysis is fed the following process and infrastructure level metrics: (*i*) the duration of each process activity; (*ii*) the duration of the whole process with respect to the *the type of the CT scan* (whole body, head, kidney etc.) as it determines the amount of work to be done; (*iii*) the particular infrastructure node a service execution has been executed on; (*iv*) the status of a service execution (successful, faulted); and (*v*) the type of infrastructure the services have been executed on (internal or external – available through Laysi).

The Influential Factor Analysis shows that from the 100 scans which violated the KPI target, 90 scans have been "whole body CT scans" executed on an external infrastructure. It also shows that the infrastructure has caused service execution faults only in 12 cases (out of 400). Finally, all scans performed on the internal infrastructure were successful. Based on this analysis, the Adaptation Needs Analysis selects predefined adaptation actions which can improve the "overall process duration in case of whole body CT scans". It selects process activity parallelization as it is the only adaptation which has been specified to have a direct positive effect on this metric.

This adaptation action is passed to the CLAM which updates the process model so that all activities are executed in parallel. The Cross-Layer Rule Engine detects the change in the process model and understands that these changes have to be checked by the composition checker and by the infrastructure checker (as the parallel execution of services has to be supported by the underlying infrastructure). The composition checker invokes the Process Re-Writing Planner which considers the original data- and control-flows of the process. It notices that the activities cannot all be executed in parallel since five of them depend on CSDA's results; thus the planner returns a new adaptation action which ensures that CSDA is executed first, while all the other activities are conducted in parallel. The model is updated in CLAM as shown in Figure 6 (b) and a new node is added to the adaptation tree. In the next step, the Cross-Layer Rule Engine invokes the infrastructure checker component which, through Laysi, discovers that the activities for tomographic reconstruction (i.e. FTR, STR, and ATR) can only be executed on the node N2. The Rule Engine handles this

new adaptation need by invoking again the Process Re-Writing Planner with a new set of control-flow constraints (i.e. FTR, STR, and ATR must be executed sequentially). The resulting process structure is shown in Figure 6 (c), in which, after CSDA, there are three parallel branches. In one of these branches FTR, STR, and ATR are executed sequentially, while, in the other two, the process executes the 3D and PACS services. The model is updated and the infrastructure checker component is invoked again. This new version of the process passes the infrastructure validation and, since there are no more checkers to be invoked, the corresponding strategy is enacted. In particular, the adapted process is handed over to DyBPEL, which manages the transition to the new process definition.

The adapted process is executed and after a certain period of time we notice that the number of KPI violations has been reduced to 10%, and that most KPI violations happen when the PACS service's execution time is too high. Therefore, two alternative adaptation actions are found: either (*i*) move the PACS service instance to another (better performing) node, or (*ii*) replace PACS with the new service PACS3D. Both alternatives are passed to CLAM.

The CLAM Rule Manager invokes the infrastructure checker with the constraint to the Laysi infrastructure stating that the PACS service should never be executed on node N4. Unfortunately, Laysi responds that, due to constraints, this is not possible and that PACS must always be executed on N4. The CLAM Rule Manager drops the first adaptation action alternative as it is not realizable, and repeats the procedure with the second adaptation action, the substitution of the PACS service with a service called PACS3D, capable of providing both storage and 3D reconstruction at a higher cost. This alternative has to be checked both by the composition checker and by the infrastructure checker. The Process Re-writing Planner detects that a new process restructuring is necessary: a new control-flow requirement is introduced by the protocol of the PACS3D service which requires to receive and store all the X-Ray data information (PACS) before computing the 3D Scan (3D). The SBA Model resulting from this new adaptation action is depicted in Figure 6 (d). The parallel branches are now only two, one for FTR, STR, and ATR, and one for PACS3D which is called twice, once to perform 3D reconstruction, and once to perform storage. The infrastructure checker validates the new model and the corresponding strategy is enacted.

8 Related Work

There are not many approaches in literature that integrate multi-layered monitoring and adaptation of service-based systems. There are however many that focus on layer-specific problems. For example, Moser et al. [10] present VieDAME, a non-intrusive approach to the monitoring of BPEL processes. The approach accumulates runtime data to calculate QoS values such as response time, accuracy, or availability. It also provides a dynamic adaptation and message mediation service for partnerlinks, using XSLT or regular expressions to transform messages accordingly. Colombo et al. [3] extend the BPEL composition language with policy (re)binding rules written in the Drools language. These rules take the form

of if-then-else statements, allowing service bindings to depend on process data collected at run time. The approach also provides mediation capabilities through a special-purpose mediation scripting language.

Researchers that do consider multi-layered applications, on the other hand, tend to concentrate either on monitoring them or on adapting them. We present the most prominent research being done in both these fields. Foster et al. [5] have proposed an extensible framework for monitoring business, software, and infrastructure services. The framework allows different kinds of reasoners, tailored to different kinds of services, to be integrated and to collaborate to monitor decomposable service level agreement terms and expressions. The framework automatically assigns the decomposed atomic terms to specific reasoners, yet the approach does not support the correlation of terms monitored at different layers. Mos et al. [9] propose a multi-layered monitoring approach that considers service and infrastructure level events produced by services deployed to a distributed enterprise service bus. Basic computations can be performed on the events to produce aggregate information (e.g., averages) or complex event processing can be used for more complex correlations and verifications. The resulting data are analyzed by comparing them to thresholds, and the knowledge collected at the various levels are presented through appropriately differentiated user interfaces and visualization techniques. The approach does not correlate knowledge collected at the different levels.

Regarding multi-level adaptation, Efstratiou et al. [4] present an approach for adapting multiple applications that share common resources. These applications are not composed, but rather single entities affected by the same contextual attributes. Since these applications live in the same space they need to coordinate how they manage the shared resources to avoid conflicts. However, they expect the users to perceive and model the conflicts manually. Finally, Popescu et al. [11] propose a framework for multi-layer adaptation of service-based systems comprised of organization, coordination and service layers. In this approach a designer needs to prepare a taxonomy of the adaptation mismatches, and then a set of adaptation templates, known as patterns, that define generic solutions for these mismatches. This differs from our proposed approach since we do not require on design-time knowledge but discover our strategies on-the-fly.

9 Conclusion and Future Work

In this paper we have presented an integrated approach for monitoring and adapting multi-layered service-based systems. The approach is based on a variant of the well-known MAPE control loops that are typical in autonomic systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a cross-layered and coordinated fashion. We have also presented initial validation of the approach on a dynamic CT scan scenario.

In our future work we will continue to evaluate the approach through new application scenarios, and through the addition of new adaptation capabilities

and adaptation enacting techniques. We will also integrate additional kinds of layers, such as a platforms, typically seen in cloud computing setups, and business layers. This will also require the development of new specialized monitors and adaptations. Finally, we will study the feasibility of managing different kinds of KPI constraints.

References

1. Baresi, L., Caporuscio, M., Ghezzi, C., Guinea, S.: Model-Driven Management of Services. In: Proceedings of the Eighth European Conference on Web Services, ECOWS, pp. 147–154. IEEE Computer Society (2010)
2. Baresi, L., Guinea, S.: Self-Supervising BPEL Processes. IEEE Trans. Software Engineering 37(2), 247–263 (2011)
3. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 191–202. Springer, Heidelberg (2006)
4. Efstratiou, C., Cheverst, K., Davies, N., Friday, A.: An Architecture for the Effective Support of Adaptive Context-Aware Applications. In: Tan, K.-L., Franklin, M.J., Lui, J.C.-S. (eds.) MDM 2001. LNCS, vol. 1987, pp. 15–26. Springer, Heidelberg (2000)
5. Foster, H., Spanoudakis, G.: SMaRT: a Workbench for Reporting the Monitorability of Services from SLAs. In: Proceedings of the 3rd International Workshop on Principles of Engineering Service-oriented Systems, PESOS, pp. 36–42. ACM (2011)
6. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM TJ Watson Labs (October 2001)
7. Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., Leymann, F.: Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In: ICSOC/ServiceWave Workshops, pp. 395–404 (2010)
8. Kertész, A., Kecskemeti, G., Brandic, I.: Autonomic SLA-Aware Service Virtualization for Distributed Systems. In: Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP, pp. 503–510 (2011)
9. Mos, A., Pedrinaci, C., Rey, G.A., Gomez, J.M., Liu, D., Vaudaux-Ruth, G., Quaireau, S.: Multi-level Monitoring and Analysis of Web-Scale Service based Applications. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 269–282. Springer, Heidelberg (2010)
10. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In: Proceeding of the 17th International Conference on World Wide Web, WWW, pp. 815–824. ACM (2008)
11. Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., Clarke, S.: Taxonomy-Driven Adaptation of Multi-layer Applications Using Templates. In: Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO, pp. 213–222 (2010)
12. Wetzstein, B., Leitner, P., Rosenberg, F., Dustdar, S., Leymann, F.: Identifying Influential Factors of Business Process Performance using Dependency Analysis. Enterprise IS 5(1), 79–98 (2011)
13. Zengin, A., Kazhamiakin, R., Pistore, M.: CLAM: Cross-layer Management of Adaptation Decisions for Service-Based Applications. In: Proceedings of the 9th International Conference on Web Services, ICWS (2011)

Efficient, Interactive Recommendation of Mashup Composition Knowledge

Soudip Roy Chowdhury, Florian Daniel, and Fabio Casati

University of Trento
Via Sommarive 5, 38123 Povo (TN), Italy
{rchowdhury,daniel,casati}@disi.unitn.it

Abstract. In this paper, we approach the problem of interactively querying and recommending composition knowledge in the form of reusable composition patterns. The goal is that of aiding developers in their composition task. We specifically focus on mashups and browser-based modeling tools, a domain that increasingly targets also people without profound programming experience. The problem is generally complex, in that we may need to match possibly complex patterns on-the-fly and in an approximate fashion. We describe an architecture and a pattern knowledge base that are distributed over client and server and a set of client-side search algorithms for the retrieval of step-by-step recommendations. The performance evaluation of our prototype implementation demonstrates that - if sensibly structured - even complex recommendations can be efficiently computed inside the client browser.

1 Introduction

Mashing up, i.e., composing, a set of services, for example, into a data processing logic, such as the data-flow based data processing pipes proposed by Yahoo! Pipes (`http://pipes.yahoo.com/pipes/`), is generally a ***complex task*** that can only be managed by skilled developers. People without the necessary programming experience may not be able to profitably use mashup tools like Pipes – to their dissatisfaction. For instance, we think of tech-savvy people, who like exploring software features, author and share own content on the Web, that would like to mash up other contents in new ways, but that don't have programming skills. They might lack appropriate awareness of which composable elements a tool provides, of their specific function, of how to combine them, of how to propagate data, and so on. The problem is analogous in the context of web service composition (e.g., with BPEL) or business process modeling (e.g., with BPMN), where modelers are typically more skilled, but still may not know all the features of their modeling languages.

Examples of ready mashup models are one of the main sources of ***help*** for modelers who don't know how to express or model their ideas – provided that suitable examples can be found (examples that have an analogy with the modeling situation faced by the modeler). But also tutorials, expert colleagues or

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 374–388, 2011.

friends, and, of course, Google are typical means to find help. However, searching for help does not always lead to success, and retrieved information is only seldom immediately usable as is, since the retrieved pieces of information are not contextual, i.e., immediately applicable to the given modeling problem.

Inspired by a study on how end users would like to be assisted in mashup development [1], we are working toward the interactive, contextual recommendation of composition knowledge, in order to assist the modeler in each step of his development task, e.g., by suggesting a candidate next component or a whole chain of tasks. The knowledge we want to recommend is re-usable ***composition patterns***, i.e., model fragments that bear knowledge that may come from a variety of possible sources, such as usage examples or tutorials of the modeling tool (developer knowledge), best modeling practices (domain expert knowledge), or recurrent model fragments in a given repository of mashup models (community knowledge [2]). The vision is that of developing an assisted, web-based mashup environment (an evolution of our former work [3]) that delivers useful composition patterns much like Google's Instant feature provides search results already while still typing keywords into the search field.

In this paper, we approach one of the core ***challenges*** of this vision, i.e., the fast search and retrieval of a ranked list of contextual development recommendations. The problem is non-trivial, in that the size of the respective knowledge base may be large, and the search for composition patterns may be complex; yet, recommendations are to be delivered at high speed, without slowing down the modeler's composition pace. Matching a partial mashup model with a repository of modeling patterns, in order to identify which of the patterns do in fact represent useful information, is similar to the well-known inexact sub-graph isomorphism problem [4], which has been proven to be NP-complete in general. Yet, if we consider that the pattern recommender should work as a plug-in for a web-based modeling tool (such as Pipes or mashArt [3], but also instruments like the Oryx BPMN editor [`http://bpt.hpi.uni-potsdam.de/Oryx/`]), fast response times become crucial.

We provide the following ***contributions***, in order to approach the problem:

- We *model* the problem of interactively recommending composition knowledge as pattern matching and retrieval problem in the context of data mashups and visual modeling tools (Section 2). This focus on one specific mashup/composition model is without loss of generality as for what regards the overall approach, and the model can easily be extended to other contexts.
- We describe an *architecture* for an assisted development environment, along with a client-side, recommendation-specific knowledge base (Section 3).
- We describe a set of *query* and *similarity search algorithms* that enable the efficient querying and ranking of interactive recommendations (Section 4).
- We study the *performance* of the conceived algorithms and show that interactively delivering composition patterns inside the modeling tool is feasible (Section 5).

In Section 6 we have a look at related works, and in the conclusion we recap the lessons we learned and provide hints of our future work.

2 Preliminaries and Problem Statement

Recommending composition knowledge requires, first of all, understanding how such knowledge looks like. We approach this problem next by introducing the mashup model that accompanies us throughout the rest of this paper and that allows us to define the concept of composition patterns as formalization of the knowledge to be recommended. Then, we characterize the typical browser-based mashup development environment and provide a precise problem statement.

2.1 Mashup Model and Composition Patterns

As a first step toward more complex mashups, in this paper we focus on ***data mashups***. Data mashups are simple in terms of modeling constructs and expressive power and, therefore, also the structure and complexity of mashup patterns is limited. The model we define in the following is inspired by Yahoo! Pipes and JackBe's Presto (`http://www.jackbe.com`) platform; in our future work we will focus on more complex models.

A data mashup model can be expressed as a tuple $m = \langle name, C, F, M, P \rangle$, where *name* is the unique name of the mashup, C is the set of components used in the mashup, F is the set of data flow connectors ruling the propagation of data among components, M is the set of data mappings of output attributes[1] to input parameters of connected components, and P is the set of parameter value assignments for component parameters. Specifically:

- $C = \{c_i | c_i = \langle name_i, desc_i, In_i, Out_i, Conf_i \rangle\}$ is the non-empty set of ***components***, with $name_i$ being the unique name of the component c_i, $desc_i$ being a natural language description of the component (for the modeler), and $In_i = \{\langle in_{ij}, req_{ij} \rangle\}, Out_i = \{out_{ik}\}$, and $Conf_i = \{\langle conf_{il}, req_{il} \rangle\}$, respectively, being the sets of input, output, and configuration parameters/attributes, and $req_{ij}, req_{il} \in \{yes, no\}$ specifying whether the parameter is required, i.e., whether it is mandatory, or not. We distinguish three kinds of components:
  - *Source* components fetch data from the web or the local machine. They don't have inputs, i.e., $In_i = \varnothing$. There may be multiple source components in C.
  - *Regular* components consume data in input and produce processed data in output. Therefore, $In_i, Out_i \neq \varnothing$. There may be multiple regular components in C.
  - *Sink* components publish the output of the data mashup, e.g., by printing it onto the screen or providing an API toward it, such as an RSS or RESTful resource. Sinks don't have outputs, i.e., $Out_i = \varnothing$. There must always be exactly one sink in C.

[1] We use the term *attribute* to denote data attributes in the data flow and the term *parameter* to denote input and configuration parameters of components.

- $F = \{f_m | f_m \in C \times C\}$ are the ***data flow connectors*** that assign to each component c_i it's predecessor c_p $(i \neq p)$ in the data flow. Source components don't require any data flow connector in input; sink components don't have data flow connectors in output.
- $M = \{m_n | m_n \in IN \times OUT, IN = \cup_{i,j} in_{ij}, OUT = \cup_{i,k} out_{ik}\}$ is the ***data mapping*** that tells each component which of the attributes of the input stream feed which of the input parameters of the component.
- $P = \{p_o | p_o \in (IN \cup CONF) \times (val \cup null), CONF = \cup_{i,l} conf_{il}\}$ is the ***value assignment*** for the input or configuration parameters of each component, *val* being a number or string value (a constant), and *null* representing an empty assignment.

This definition allows models that may not be executed in practice, e.g., because the data flow is not fully connected. With the following properties we close this gap:

Definition 1. *A mashup model m is* **correct** *if the graph expressed by F is connected and acyclic.*

Definition 2. *A mashup model m is* **executable** *if it is correct and all required input and configuration parameters have a respective data mapping or value assignment.*

These two properties must only hold in the moment we want to execute a mashup m. Of course, during development, e.g., while modeling the mashup logic inside a visual mashup editor, we may be in the presence of a *partial* mashup model $pm = \langle C, F, M, P \rangle$ that may be neither correct nor executable. Step by step, the mashup developer will then complete the model, finally obtaining a correct and executable one, which can typically be run directly from the editor in a hosted fashion.

Given the above characterization of mashups, we can now define composition knowledge that can be recommended as re-usable ***composition patterns*** for mashups of type m, i.e., model fragments that provide insight into how to solve specific modeling problems. Generically – given the mashup model introduced before – we express a composition pattern as a tuple $cp = \langle C, F, M, P, usage, date \rangle$, where C, F, M, P are as defined for m, *usage* counts how many times the pattern has been used (e.g., to compute rankings), and *date* is the creation date of the pattern. In order to be useful, a pattern must be correct, but not necessarily executable. The size of a pattern may vary from a single component with a value assignment for at least one input or configuration parameter to an entire, executable mashup; later on we will see how this is reflected in the structure of individual patterns.

Finally, to effectively deliver recommendations it is crucial to understand *when* to do so. Differently from most works on pattern search in literature (see Section 6), we aim at an ***interactive recommendation*** approach, in which patterns are queried for and delivered in response to individual modeling actions performed by the user in the modeling canvas. In visual modeling environments, we typically

have $action \in \{select, drag, drop, connect, delete, fill, map, ...\}$, where *action* is performed on an $object \subseteq C \cup F \cup IN \cup CONF$, i.e., on the set of modeling constructs affected by the last modeling action. For instance, we can *drop* a component c_i onto the canvas, or we can *select* a parameter $conf_{il}$ to fill it with a value, we can *connect* a data flow connector f_m with an existing target component, or we can *select* a set of components and connectors.

2.2 Problem Statement

In the composition context described above, providing interactive, contextual development recommendations therefore corresponds to the following problem statement: given a query $q = \langle object, action, pm \rangle$, with pm being the partial mashup model under development, how can we obtain a list of ranked composition patterns $R = [\langle cp_i, rank_i \rangle]$ (the recommendations), such that (i) the provided recommendations help the developer to stepwise draw an executable mashup model and (ii) the search, ranking, and delivery of the recommendations can be efficiently embedded into an interactive modeling process?

3 Recommending Composition Knowledge: Approach

The key idea we follow in this work is not trying to crack the whole problem at once. That is, we don't aim to match a query q against a repository of generic composition patterns of type *cp* in order to identify best matches. This is instead the most followed approach in literature on graph matching, in which, given a graph g_1, we search a repository of graphs for a graph g_2, such that g_1 is a sub-graph of g_2 or such that g_1 satisfies some similarity criteria with a sub-graph of g_2. Providing *interactive* recommendations can be seen as a specific instance of this generic problem, which however comes with both a new challenge as well as a new opportunity: the new challenge is to query for and deliver possibly complex recommendations *responsively*; the opportunity stems from the fact that we have an interactive recommendation consumption process, which allows us to *split* the task into optimized sub-steps (e.g., search for data mappings, search for connectors, and similar), which in turn helps improve performance.

Having an interactive process further means having a user performing modeling actions, inspecting recommendations, and accepting or rejecting them, where accepting a recommendation means weaving (i.e., connecting) the respective composition pattern into the partial mashup model under development. Thanks to this process, we can further split recommendations into what is needed to *represent* a pattern (e.g., a component co-occurrence) from what is needed to *use* the pattern in practice (e.g., the exact mapping of output attributes to input parameters of the component co-occurrence). We can therefore further leverage on the separation of pattern representation and usage: representations (the recommendations) don't need to be complete in terms of ingredients that make up a pattern; completeness is required only at usage time.

3.1 Types of Knowledge Patterns

Aiming to help a developer to stepwise refine his mashup model, practically means suggesting the developer which next modeling action (that makes sense) can be performed in a given state of his progress and doing so by providing as much help (in terms of modeling actions) as possible. Looking at the typical modeling steps performed by a developer (filling input fields, connecting components, copying/pasting model fragments) allows us to define the following ***types of patterns*** (for simplicity, we omit the *usage* and *date* attributes):

- *Parameter value* pattern: $cp^{par} = \langle c_x, \varnothing, \varnothing, p_o^x \rangle$. Given a component, the system suggest values for the component's parameters.
- *Connector* pattern: $cp^{conn} = \langle \{c_x, c_y\}, f^{xy}, \varnothing, \varnothing \rangle$. Given two components, the system suggests a connector among the components.
- *Data mapping* pattern: $cp^{map} = \langle \{c_x, c_y\}, f^{xy}, \{m_n^{xy}\}, \varnothing \rangle$. Given two components and a connector among them, the system suggests how to map the output attributes of the first component to the parameters of the second component.
- *Component co-occurrence* pattern: $cp^{co} = \langle \{c_x, c_y\}, f^{xy}, \{m_n^{xy}\}, \{p_o^x\} \cup \{p_o^y\} \rangle$. Given one component, the system suggests a possible next component to be used, along with all the necessary data mappings and value assignments.
- *Complex* pattern: $cp^{com} = \langle C, F, M, P \rangle$. Given a fragment of a mashup model, the system suggests a pattern consisting of multiple components and connectors, along with the respective data mappings and value assignments.

Our definition of *cp* would allow many more possible types of composition patterns, but not all of them make sense if patterns are to be re-usable as is, that is, without requiring further refinement steps like setting parameter values. This is the reason for which we include also connectors, data mappings, and value assignments when recommending a component co-occurrence pattern.

3.2 The Interactive Modeling and Recommender System

Figure 1 illustrates the internals of our prototype modeling environment equipped with an interactive knowledge recommender. We distinguish between client and server side, where the whole application logic is located in the client, and the server basically hosts the *persistent pattern knowledge base* (KB; details in Section 3.3). At startup, the *KB loader* loads the patterns into the client environment, decoupling the knowledge recommender from the server side.

Once the editor is running inside the client browser, the developer can visually compose components (in the *modeling canvas*) taken from the *component tool bar*. Doing so generates modeling events (the *actions*), which are published on a browser-internal *event bus*, which forwards each modeling action to the *recommendation engine*. Given a modeling *action*, the *object* it has been applied to, and the partial mashup model *pm*, the engine queries the *client-side pattern KB* via the *KB access API* for recommendations (pattern representations).

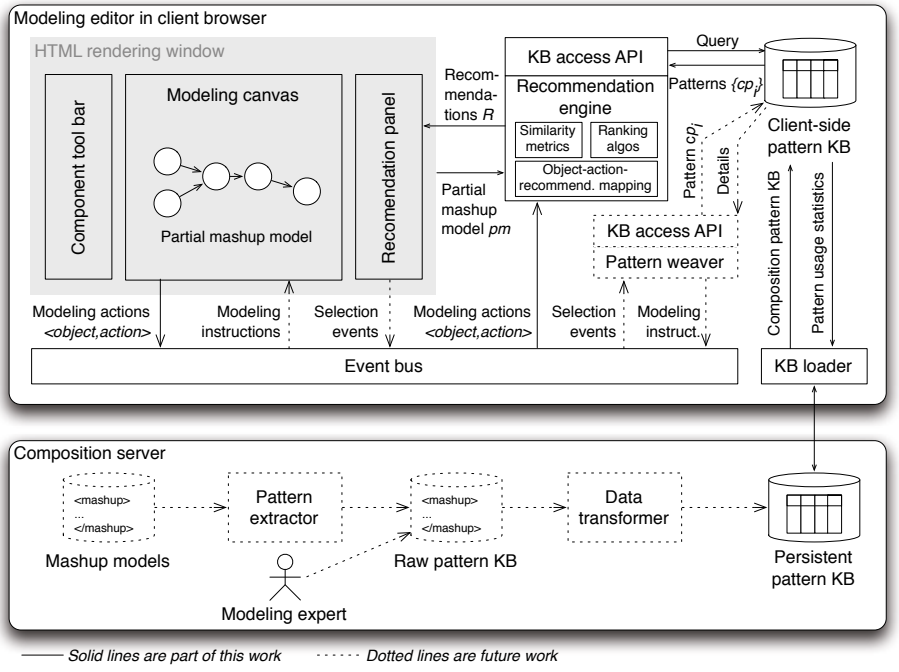


Fig. 1. Simplified architecture of the assisted modeling environment with client-side knowledge base and interactive recommender. We focus on recommendations only and omit elements like the mashup runtime environment, the component library, etc.

An *object-action-recommendation mapping* (OAR) tells the engine which type of recommendation is to be retrieved for each modeling action on a given object.

The list of patterns retrieved from the KB (either via regular queries or by applying dedicated similarity criteria) are then ranked by the engine and rendered in the *recommendation panel*, which renders the recommendations to the developer for inspection. In future, selecting a recommendation will allow the *pattern weaver* to query the KB for the usage details of the pattern (data mappings and value assignments) and to automatically provide the modeling canvas with the necessary modeling instructions to weave the pattern into the partial mashup model.

3.3 Patterns Knowledge Base

The core of the interactive recommender is the KB that stores generic patterns, but decomposed into their constituent parts, so as to enable the incremental recommendation approach. If we recall the generic definition of composition patterns, i.e., $cp = \langle C, F, M, P, usage, date \rangle$, we observe that, in order to convey the structures of a set of complex patterns inside a visual modeling tool, typically C and F (components and connectors) will suffice to allow a developer

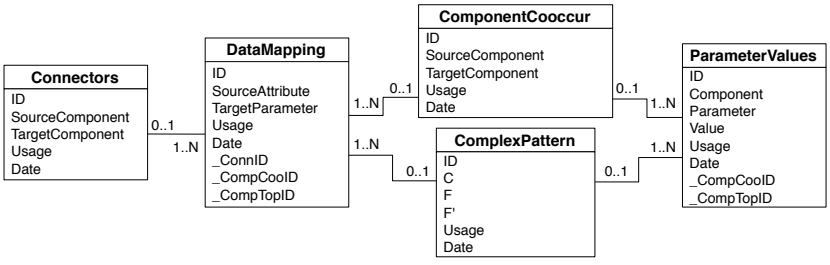


Fig. 2. Model of the pattern knowledge base for client-side knowledge management

to select a candidate pattern. Ready data mappings and value assignments are then delivered together with the components and connectors only upon selection of a pattern by the developer.

This observation leads us to the KB illustrated in Figure 2, whose structure enables the retrieval of the representations of the types of recommendations introduced in Section 3.1 with a one-shot query over a single table. For instance, the entity *Connectors* contains all connector patterns, and the entity *ComplexPattern* contains the structure of the complex patterns (in Section 4 we explain the meaning of the attributes C, F, F'). The KB is partly redundant (e.g., the structure of a complex pattern also contains components and connectors), but this is intentional. It allows us to defer the need for joins to the moment in which we really need to retrieve all details of a pattern, i.e., when we want to use it. In order to retrieve, for example, the representation of a component co-occurrence pattern, it is therefore enough to query the *ComponentCooccur* entity for the *SourceComponent* and the *TargetComponent* attributes; weaving the pattern then into the modeling canvas requires querying $ComponentCooccur \bowtie DataMapping \bowtie ParameterValues$ for the details.

4 Exact and Approximate Search of Recommendations

Given the described types of composition patterns and a query q, we retrieve composition recommendations from the described KB in two ways: (i) we *query* the KB for parameter value, connector, data mapping, and component co-occurrence patterns; and (ii) we *match* the *object* against complex patterns. The former approach is based on *exact* matches with the *object*, the latter leverages on *similarity search*. Conceptually, all recommendations could be retrieved via similarity search, but for performance reasons we apply it only in those cases (the complex patterns) where we don't know the structure of the pattern in advance and, therefore, are not able to write efficient conventional queries.

Algorithm 1 details this ***strategy*** and summarizes the logic implemented by the recommendation engine. In line 3, we retrieve the types of recommendations that can be given (*getSuitableRecTypes* function), given an *object-action* combination. Then, for each recommendation type, we either query for patterns (the

Algorithm 1. getRecommendations

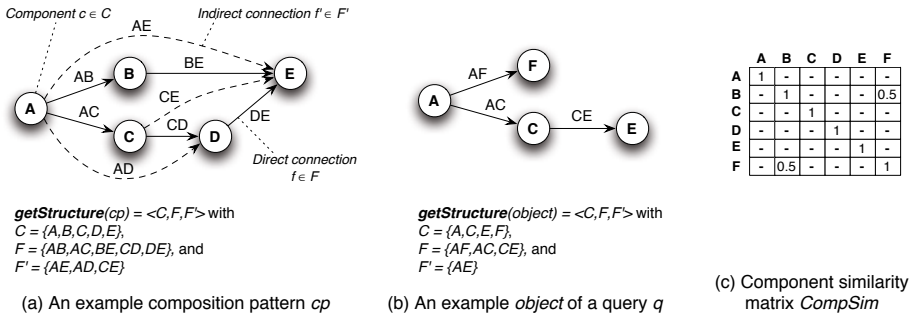
Data: query $q = \langle object, action, pm \rangle$, knowledge base KB, object-action-recommendation mapping OAR, component similarity matrix $CompSim$, similarity threshold T_{sim}, ranking threshold T_{rank}, number n of recommendations per recommendation type
Result: recommendations $R = [\langle cp_i, rank_i \rangle]$ with $rank_i \geq T_{rank}$

```
1  R = array();
2  Patterns = set();
3  recTypeToBeGiven = getSuitableRecTypes(object, action, OAR);
4  foreach recType ∈ recTypeToBeGiven do
5      if recType ∈ {ParValue, Connector, DataMapping, CompCooccur} then
6          Patterns = Patterns∪ queryPatterns(object, KB, recType) ;      // exact query
7      else
8          Patterns = Patterns∪
           getSimilarPatterns(object, KB.ComplexPattern, CompSim, T_sim) ; // similarity
           search
9  foreach pat ∈ Patterns do
10     if rank(pat.cp, pat.sim, pm) ≥ T_rank then
11         append(R, ⟨pat.cp, rank(pat.cp, pat.sim, pm)⟩) ;    // rank, threshold, remember
12 orderByRank(R);
13 groupByType(R);
14 truncateByGroup(R, n);
15 return R;
```

queryPatterns function can be seen like a traditional SQL query) or we do a similarity search (*getSimilarPatterns* function, see Algorithm 2). For each retrieved pattern, we compute a rank, e.g., based on the pattern description (e.g., containing *usage* and *date*), the computed similarity, and the usefulness of the pattern inside the partial mashup, order and group the recommendations by type, and filter out the best n patterns for each recommendation type.

As for the retrieval of ***similar patterns***, our goal was to help modelers, not to disorient them. This led us to the identification of the following principles for the identification of "similar" patterns: preference should be given to exact matches of components and connectors in *object*, candidate patterns may differ for the insertion, deletion, or substitution of at most one component in a given path in *object*, and among the non-matching components preference should be given to functionally similar components (e.g., it may be reasonable to allow a Yahoo! Map instead of a Google Map).

Algorithms 2 and 3 implement these requirements, although in a way that is already optimized for execution, in that they don't operate on the original, graph-like structure of patterns, but instead on a pre-processed representation that prevents us from traversing the graph at runtime. Figure 3(a) illustrates the pre-processing logic: each complex pattern is represented as a tuple $\langle C, F, F' \rangle$, where C is the set of components, F the set of direct connections, and F' the set of indirect connections, skipping one component for approximate search. This pre-processing logic is represented by the function *getStructure*, which can be evaluated offline for each complex pattern in the raw pattern KB; results are stored in the *ComplexPattern* entity introduced in Figure 2. Another input that can be computed offline is the component similarity matrix *CompSim*, which can be set up by an expert or automatically derived by mining the raw pattern KB. For the purpose of recommending knowledge, similarity values should



	A	B	C	D	E	F
A	1	-	-	-	-	-
B	-	1	-	-	-	0.5
C	-	-	1	-	-	-
D	-	-	-	1	-	-
E	-	-	-	-	1	-
F	-	0.5	-	-	-	1

Fig. 3. Pattern pre-processing and example of component similarity matrix *CompSim*. Components are identified with characters, connectors with their endpoints.

reflect semantic similarity among components (e.g., two *flight search* services); syntactic differences are taken into account by the pattern structures. Figure 3(c) illustrates a possible matrix for the components in the sub-figures (a) and (b); similarity values are contained in [0..1], 0 representing no similarity, 1 representing equivalence.

Algorithm 2 now works as follows. First, it derives the optimized structure of *object* (line 2). Then, it compares it with each complex pattern $cp \in CP$ in four steps: (i) it computes a similarity value for all components and connectors of *obj* and *cp* that have an exact match (line 5); (ii) it eliminates all matching components and connectors from the structure of *obj* (lines 6-8); (iii) it computes the best similarity value for the so-derived *obj* by approximating it with other components based on *CompSim* (lines 9-16); and it aggregates to two similarity values (line 17). Specifically, the algorithm substitutes one component at a time in *obj* (using *getApproximatePattern* in line 13), considering all possible substitutes *simc* and their similarity values *simc.sim* obtained from *CompSim*. The actual similarity value between two patterns is computed by Algorithm 3.

Let's consider the pattern, object, and similarity matrix in Figure 3. If in Algorithm 3 we use the weights $w_i \in \{0.5, 0.2, 0.1, 0.1, 0.1\}$ in the stated order, *sim* in line 4 of Algorithm 2 is 0.57 (exact matches for 3 components and 2 connectors). After the elimination of those matches, $obj = \langle\{F\}, \{AF\}, \varnothing\rangle$, and substituting F with B as suggested by *CompSim* allows us to obtain an additional approximate similarity of $approxSim = 0.35$ (two matches and $simc.sim = 0.5$), which yields a total similarity of $sim = 0.57 + 0.35/4 = 0.66$.

5 Implementation and Performance Evaluation

We implemented the *recommendation engine*, the *KB access API*, and the *client-side pattern KB* along with the recommendation and similarity search algorithms, in order to perform a detailed performance analysis. The ***prototype implementation*** is entirely written in JavaScript and has been tested with a Firefox 3.6.17 web browser. The implementation of the *client-side KB* is based

Algorithm 2. getSimilarPatterns

```
Data: query object object, set of complex patterns CP, component similarity matrix
      CompSim, similarity threshold T_sim
Result: Patterns = {⟨cp_i, sim_i⟩} with sim_i ≥ T_sim
1  Patterns = set();
2  objectStructure = getStructure(object) ;   // computes object's structure for comparison
3  foreach cp ∈ CP do
4      obj = objectStructure;
5      sim = getSimilarity(obj, cp) ;                // compute similarity for exact matches
6      obj.C = obj.C − cp.C ;          // eliminate all exact matches for C, F, F' from obj
7      obj.F = obj.F − cp.F − cp.F';
8      obj.F' = obj.F' − cp.F' − cp.F;
9      approxSim = 0;          // will contain the best similarity for approximate matches
10     foreach c ∈ obj.C do
11         SimC = getSimilarComponents(c, CompSim) ;  // get set of similar components
12         foreach simc ∈ SimC do
13             approxObj = getApproximatePattern(obj, c, simc) ;   // get approx. pattern
14             newApproxSim = simc.sim*getSimilarity(approxObj, cp) ; // get similarity
15             if newApproxSim > approxSim then
16                 approxSim = newApproxSim ;  // keep highest approximate similarity
17     sim = sim + approxSim * |obj.C|/|objectStructure.C| ;   // normalize and aggregate
18     if sim ≥ T_sim then
19         Patterns = Patterns ∪ ⟨cp, sim⟩ ;      // remember patterns with sufficient sim
20 return Patterns;
```

Algorithm 3. getSimilarity

```
Data: query object object, complex pattern cp
Result: similarity
1  initialize w_i for i ∈ 1..5 with Σ_i w_i = 1;
2  sim_1 = |object.C ∩ cp.C|/|object.C| ;                          // matches components
3  sim_2 = |object.F ∩ cp.F|/|object.F| ;                          // matches connectors
4  sim_3 = |object.F ∩ cp.F'|/|object.F| ;          // allows insertion of a component
5  sim_4 = |object.F' ∩ cp.F|/|object.F'| ;          // allows deletion of a component
6  sim_5 = |object.F' ∩ cp.F'|/|object.F'| ;      // allows substitution of a component
7  similarity = Σ_i w_i * sim_i;
8  return similarity;
```

on Google Gears (`http://gears.google.com`), which internally uses SQL Lite (`http://www.sqlite.org`) for storing data on the client's hard drive. Given that SQL Lite does not support set data types, we serialize the representation of complex patterns $\langle C, F, F' \rangle$ in JSON and store them as strings in the respective *ComplexPattern* table in the KB; doing so slightly differs from the KB model in Figure 2, without however altering its spirit. The implementation of the *persistent pattern KB* is based on MySQL, and it is accessed by the *KB loader* through a dedicated RESTful Java API running inside an Apache 2.0 web server. The prototype implementation is installed on a MAC machine with OS X 10.6.1, a 2.26 GHz Intel Core 2 Duo processor, and 2 GB of memory. Response times are measured with the FireBug 1.5.4 plug-in for Firefox.

For the generation of realistic ***test data***, we assumed to be in the presence of a mashup editor with 26 different components ($A-Z$), with a random number of input and configuration parameters (ranging from $1-5$) and a random number of output attributes (between $1-5$). To obtain an *upper bound* for the performance of the *exact queries* for parameter value, connector, data mapping, and

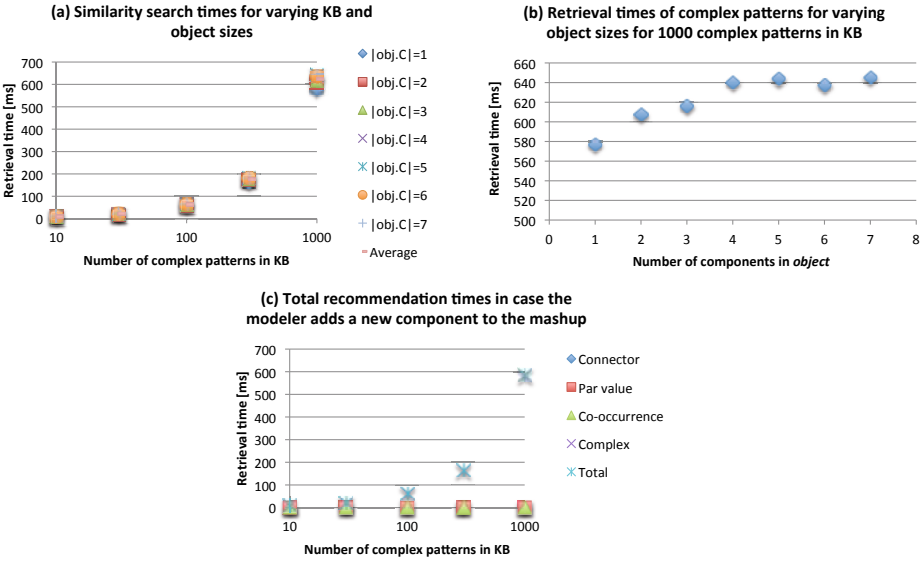


Fig. 4. Performance evaluation of the client-side knowledge recommender

component co-occurrence patterns, we generated, respectively, $26 * 5 = 130$ parameter values for the 26 components, $26 * 25 = 650$ directed connectors, $650 * 5 = 3250$ data mappings, and 650 component co-occurrences. To measure the performance of the *similarity search* algorithms, we generated 5 different KBs with 10, 30, 100, 300, 1000 complex patterns, where the complexity of patterns ranges from $3 - 9$ components. The patterns make random use of all available components and are equally distributed in the generated KBs. Finally, we generated a set of query objects with $|obj.C| \in \{1..7\}$.

In Figure 4, we illustrate the tests we performed and the respective ***results***. The first test in Figure 4(a) studies the performance in terms of pattern retrieval times of Algorithm 2 for different *KB sizes*; the figure plots the retrieval times for different *object sizes*. Considering the logarithmic scale of the x-axis, we note that the retrieval time for complex patterns grows almost linearly. This somehow unexpected behavior is due to the fact that, compared to the number of patterns, the complexity of patterns is similar among each other and limited and, hence, the similarity calculation can almost be considered a constant. We also observe that there are no significant performance differences for varying object sizes. In Figure 4(b) we investigate the effect of the object size on the performance of Algorithm 2 only for the KB with 1000 complex patterns (the only one with notable differences). Apparently, also the size of the query object does not affect much retrieval time. Figure 4(c), finally, studies the performance of Algorithm 1, i.e., the performance perceived by the user, in a typical modeling situation: in response to the user placing a new component into the canvas, the recommendation engine retrieves respective parameter value, connector, co-occurrence, and complex patterns (we do not recommend data mappings for single components);

the overall response time is the sum of the individual retrieval times. As expected, the response times of the simple queries can be neglected compared to the one of the similarity search for complex patterns, which basically dominates the whole recommendation performance.

In summary, the above tests confirm the validity of the proposed pattern recommendation approach and even outperform our own expectations. The number of components in a mashup or composition tool may be higher, yet the number of really meaningful patterns in a given modeling domain only unlikely will grow beyond several dozens or 100. Recommendation retrieval times of fractions of seconds will definitely allow us – and others – to develop more sophisticated, assisted composition environments.

6 Related Work

Traditionally, ***recommender systems*** focus on the retrieval of information of likely interest to a given user, e.g., newspaper articles or books. The likelihood of interest is typically computed based on a *user profile* containing the user's areas of interest, and retrieved results may be further refined with collaborative filtering techniques. In our work, as for now we focus less on the user and more on the partial mashup under development (we will take user preferences into account in a later stage), that is, recommendations must match the partial mashup model and the object the user is focusing on, not his interests. The approach is related to the one followed by research on *automatic service selection*, e.g., in the context of QoS- or reputation-aware service selection, or adaptive or self-healing service compositions. Yet, while these techniques typically approach the problem of selecting at runtime a concrete service for an abstract activity, we aim at interactively assisting developers at design time with more complex information in the form of complete modeling patterns.

In the context of ***web mashups***, Carlson et al. [5], for instance, react to a user's selection of a component with a recommendation for the next component to be used; the approach is based on semantic annotations of component descriptors and makes use of WordNet for disambiguation. Greenshpan et al. [6] propose an auto-completion approach that recommends components and connectors (so-called glue patterns) in response to the user providing a set of desired components; the approach computes top-k recommendations out of a graph-structured knowledge base containing components and glue patterns (the nodes) and their relationships (the arcs). While in this approach the actual structure (the graph) of the knowledge base is hidden to the user, Chen et al. [7] allow the user to mashup components by navigating a graph of components and connectors; the graph is generated in response to the user's query in form of descriptive keywords. Riabov et al. [8] also follow a keyword-based approach to express user goals, which they use to feed an automated planner that derives candidate mashups; according to the authors, obtaining a plan may require several seconds. Elmeleegy et al. [9] propose MashupAdvisor, a system that, starting from a component placed by the user, recommends a set of related components (based on conditional co-occurrence probabilities and semantic matching); upon

selection of a component, MashupAdvisor uses automatic planning to derive how to connect the selected component with the partial mashup, a process that may also take more than one minute. Beauche and Poizat [10] apply automatic planning in the context of ***service composition***. The planner generates a candidate composition starting from a user task and a set of user-specified services.

The ***business process management*** (BPM) community more strongly focuses on patterns as a means of knowledge reuse. For instance, Smirnov et al. [11] provide so-called co-occurrence action patterns in response to action/task specifications by the user; recommendations are provided based on label similarity, and also come with the necessary control flow logic to connect the suggested action. Hornung et al. [12] provide users with a keyword search facility that allows them to retrieve process models whose labels are related to the provided keywords; the algorithm applies the traditional TF-IDF technique from information retrieval to process models, turning the repository of process model into a keyword vector space. Gschwind et al. [13] allow users in their modeling tool to insert control flow patterns, as introduced by Van der Aalst et al. [14], just like other modeling elements. The proposed system does not provide interactive recommendations and rather focuses on the correct insertion of patterns.

In summary, the mashup and service composition approaches either focus on single components or connectors, or they aim to automatically plan complete compositions starting from user goals. The BPM approaches do focus on patterns as reusable elements, but most of the times pattern similarity is based on label/text similarity, not on structural compatibility. We assume components have stable names and, therefore, we do not need to interpret text labels.

7 Conclusion and Future Work

In this paper, we focused on a relevant problem in visual mashup development, i.e., the recommendation of composition knowledge. The approach we followed is similar to the one adopted in data warehousing, in which data is transformed from their operational data structure into a dimensional structure, which optimizes performance for reporting and data analysis. Analogously, instead of querying directly the raw pattern knowledge base, typically containing a set of XML documents encoding graph-like mashup structures, we decompose patterns into their constituent elements and transform them into an optimized structure directly mapped to the recommendations to be provided. We access patterns with fixed structure via simple queries, while we provide an efficient similarity search algorithm for complex patterns, whose structure is not known a-priori.

We specifically concentrated on the case of client-side mashup development environments, obtaining very good results. Yet, the described approach will perform well also in the context of other browser-based modeling tools, e.g., business process or service composition instruments (which are also model-based and of similar complexity), while very likely it will perform even better in desktop-based modeling tools like the various Eclipse-based visual editors. As such, the pattern recommendation approach discussed in this paper represents a valuable, practical input for the development of advanced modeling environments.

Next, we will work on three main aspects: The complete development of the interactive modeling environment for the interactive derivation of *search queries* and the automatic *weaving* of patterns; the *discovery* of composition patterns from a repository of mashup models; the fine-tuning of the similarity and ranking algorithms with the help of suitable *user studies*. This final step will also allow us to assess and tweak the set of proposed composition patterns.

Acknowledgments. This work was partially supported by funds from the European Commission (project OMELETTE, contract no. 257635).

References

1. De Angeli, A., Battocchi, A., Roy Chowdhury, S., Rodríguez, C., Daniel, F., Casati, F.: End-user requirements for wisdom-aware eud. In: IS-EUD 2011. Springer, Heidelberg (2011)
2. Roy Chowdhury, S., Rodríguez, C., Daniel, F., Casati, F.: Wisdom-aware computing: On the interactive recommendation of composition knowledge. In: WESOA 2010, pp. 144–155. Springer, Heidelberg (2010)
3. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
4. Hlaoui, A., Wang, S.: A new algorithm for inexact graph matching. In: ICPR 2002, vol. 4, pp. 180–183 (2002)
5. Carlson, M.P., Ngu, A.H., Podorozhny, R., Zeng, L.: Automatic Mash up of Composite Applications. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 317–330. Springer, Heidelberg (2008)
6. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for mashups. In: VLDB 2009, vol. 2, pp. 538–549 (2009)
7. Chen, H., Lu, B., Ni, Y., Xie, G., Zhou, C., Mi, J., Wu, Z.: Mashup by surfing a web of data apis. In: VLDB 2009, vol. 2, pp. 1602–1605 (2009)
8. Riabov, A.V., Boillet, E., Feblowitz, M.D., Liu, Z., Ranganathan, A.: Wishful search: interactive composition of data mashups. In: WWW 2008, pp. 775–784. ACM (2008)
9. Elmeleegy, H., Ivan, A., Akkiraju, R., Goodwin, R.: Mashup advisor: A recommendation tool for mashup development. In: ICWS 2008, pp. 337–344. IEEE Computer Society (2008)
10. Beauche, S., Poizat, P.: Automated Service Composition with Adaptive Planning. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 530–537. Springer, Heidelberg (2008)
11. Smirnov, S., Weidlich, M., Mendling, J., Weske, M.: Action Patterns in Business Process Models. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 115–129. Springer, Heidelberg (2009)
12. Hornung, T., Koschmider, A., Lausen, G.: Recommendation Based Process Modeling Support: Method and User Experience. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 265–278. Springer, Heidelberg (2008)
13. 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)
14. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases 14, 5–51 (2003)

A Semantic and Information Retrieval Based Approach to Service Contract Selection*

Silvia Calegari, Marco Comerio, Andrea Maurino,
Emanuele Panzeri, and Gabriella Pasi

University of Milano-Bicocca,
Department of Informatics, Systems and Communication (DISCo), Milano, Italy
{calegari,comerio,maurino,panzeri,pasi}@disco.unimib.it

Abstract. Service contracts represent the agreement between the service provider and potential service consumers to use a specific service under given conditions; for each service multiple service contracts are available. In this paper we investigate a new approach to support the service contract selection by exploiting preferences both explicitly defined by a user and implicitly inferred from his/her context. The core of our approach is the use of multi-constraint queries expressed on punctual values and on textual descriptions. Both semantic-based and information retrieval (IR) techniques are applied. Experimental evaluations show the effectiveness of the proposed approach.

1 Introduction

The visionary idea of Service-Oriented Computing (SOC) is a service ecosystem where application components are assembled with little effort into a loosely-coupled network of services to create agile applications that might span organizations and computing platforms [1]. One of the building block of SOC is *service discovery* that is the activity of locating a machine-processable description of a service that meets certain functional requirements [2].

Since more than one service is likely to fulfill the functional requirements, some ranking mechanisms are needed in order to provide support for the (semi) automatic selection of a restricted number of services (usually one) among the discovered ones. Broadly speaking we can identify two phases in the service discovery activity: the first one is devoted to identify services that satisfy the functional requirements, while the second one (also called *service selection*) is in charge of ranking retrieved services according to non-functional properties (NFPs) that represent the description of the service characteristics (e.g., availability, performance, price) that are not directly related to the provided functionality. As in the real world also in SOC ecosystem, NFPs can be enclosed in *service contracts* representing the agreement between the service provider and potential service consumers. In the last years, increasing research efforts are

* This work is partially supported by the Italian MoSeForAgroFood project (funded by the Lombardy region), and by the SAS Institute srl (Grant Carlo Grandi).

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 389–403, 2011.

aimed at defining solutions for service contract management [3]. For each service, multiple service contracts are available and each service contract can be offered to specific user categories under predefined applicability conditions. More specifically, constraints based on NFPs consist in the specification of *contractual terms*. They can be expressed by numeric values defined in different units (e.g., price in Euro or in USD), or by qualitative values (e.g., trust is *high*, software is *open source*). The *service contract selection* is the activity of ranking service contracts according to the constraints on NFP explicitly specified by the user, and/or implicitly inferred from user information.

Service contract selection is definitively one of the most important enabling factors for supporting flexible and dynamic business processes and agile applications. Nevertheless, conversely to the real world where contract selection is a largely studied problem [4,5], this activity has not yet been extensively investigated, and current approaches [6,7,8,9] lack, among others, in providing support for the formulation of user requests, in the evaluation of applicability conditions and in managing the heterogeneity of NFPs that can be specified in a service contract. The management of NFPs is a complex task since there exists no standard terminology for describing these properties. This means that service providers and consumers specify their service contracts as they wish, thus raising the term ambiguity problem when multiple services governed by different contracts are utilized. In fact, similar properties may have different names (e.g., in different languages or domains) or the same name may refer to different properties (e.g., in different domains a property may have different implications). This current lack of agreed terminology, combined with a major absence of trust in claims about service contracts, renders service contract selection difficult if not impossible in commercial organizations.

In this paper we present a new approach to service contracts selection based on the exploitation of preferences explicitly defined by a user and implicitly inferred from his/her context and the use of both semantic-based and information retrieval (IR) techniques to rank service contracts. In particular, the main contributions of our approach are:

- *multi-constraint query formulation*: the constraints on NFP composing the user query are defined by considering preferences explicitly specified by the user, and implicitly inferred from user information (e.g., personal information specified at registration-time, historical information related to formerly service used). The constraints can be either expressed as data constraints or keyword-based constraints and they can be defined on a wide set of NFPs.
- *hybrid approach to service contract ranking*: the ranking of service contracts is based on the combination of semantic and information retrieval techniques to evaluate the degree of matching between contractual terms and user preferences.

The rest of the paper is organized as follows: Section 2 presents the state of the art of service contract selection and related fields. Section 3 describes the proposed approach. In Section 4, and 5 our hybrid service contract selection approach is described. Sections 6 and 7 present an exhaustive example and the

experimental evaluations of our approach. Conclusions and future works conclude the paper in Section 8.

2 State of the Art

The agreement between a service provider and a service consumer can be established by using different approaches (e.g., policies [10] and service level agreements [11]). Even if some differences exist among these approaches, the common term *service contract* is generally used [3].

Currently, service contract selection is executed by either non-semantic or semantic approaches. Non-semantic approaches (e.g.,[6,7]) are characterized by a high efficiency but low precision due to the management of only syntactic service contract descriptions. The evaluation of degrees of matching between requested and offered contractual terms related to qualitative NFPs is reduced to the syntactic comparison among values, raising semantic misunderstandings and inefficient selections. Semantic approaches (e.g., [8,9]) are based on automated reasoning techniques on service contract descriptions. These techniques are particularly suitable to mediate different terminologies and data models. Therefore, reasoning techniques can be used for the evaluation of degrees of matching of contractual terms related to qualitative NFPs in order to exploit semantic relations between NFP values. However, the evaluation based on logical reasoning is characterized by a low efficiency since many reasoners show poor effectiveness when dealing with non trivial numeric functions (e.g., weighted sums) which are needed to manage more properties at the same time.

The most important problems in both semantic or non-semantic approaches above mentioned are: (i) *expressivity* as the possibility to evaluate qualitative descriptions by means of logical expressions on ontology values, and quantitative descriptions by mean of expressions including ranges and inequalities; (ii) *extensibility* as the possibility to define parametric degree of matching evaluation by customizing evaluation functions and (iii) *flexibility* as the possibility to perform evaluation in case of incomplete specifications.

For example, the approaches in [6,7,8] present some limitations in expressivity and extensibility. The NFP-based service selection approach proposed in [6] considers the evaluation of qualitative properties but it does not consider the semantic relations among property values. The framework described in [7] allows the definition of requested contractual terms only using the *equal* operator and the selection process is simplified and modelled as the problem to maximize the difference between prices (associated with each contract using a pricing function) and score (associated with each requested contract and stating the maximum price for which the costumer is willing to carry out the trade). Finally, the semantic approach in [8] is applicable only for properties characterized by ordered scale values and fixed tendencies (e.g., the cost must always be minimized) limiting the freedom of the user in defining his/her preferences.

The approach to Web service selection based on the usage of axioms for requested and offered contractual terms defined in [9] lacks in flexibility. The exploitation of axioms supports complex and conditioned term definition (e.g., if

the client is older than 60 or younger than 10 years old the invocation price is lower than 10 euro) but forces the user to specify all the necessary information (e.g., age) in advance.

In [12], an hybrid approach to Web service contract selection that combines logic-based and algorithmic techniques and offers high levels of expressivity, extensibility and flexibility is proposed and tested. The limitation of the approach is that applicability conditions on service contracts are not evaluated and the approach lacks in providing support for the formulation of user requests. In this paper, we extend the approach in [12] by means of IR techniques.

3 The Proposed Approach

The aim of the whole service contract selection process is to propose to the user a list of service contracts ranked according to his/her preferences. The process is composed of set-up time and run time activities and it is based on the software architecture shown in Figure 1. At set-up time, the user interacts with the registration module in order to create his/her *user profile*. At run time the user specifies preferences on functional and non-functional properties in order to perform the service discovery and the service contract selection. At set-up time, during the registration, the user selects from a list one of the pre-defined profiles expressed in natural language. The pre-defined profiles help the user to provide relevant information on generic characteristics (e.g., spoken languages, used devices). Then, the user completes the registration by inserting personal information such as (i) his/her personal data, (ii) his/her agenda in order to know at which time the user is located in a particular location, and (iii) preferences

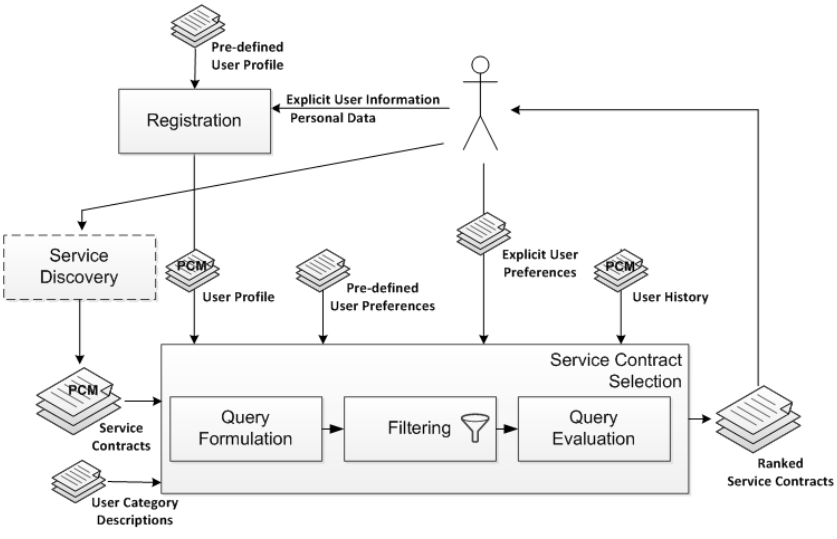


Fig. 1. The proposed approach to service contract selection

on specific properties such as the preferred payment method. Preferences are specified by means of textual descriptions. The user information gathered in this phase (i.e., personal information, selected profiles and textual descriptions) are jointly considered to define the *user profile*. It is worth noting that our approach to build the user profile is not tailored to any specific context model, and several context models can be applied to our approach.

Once the phase of information gathering is completed, the user can access to the next phase. At run time, when the user looks for a service offering a specific functionality, the service discovery component (not discussed in this paper) is invoked. Each discovered service is associated with different service contracts representing different NFPs, and applicable to different user categories. In order to support the user in choosing the contract that best complies to his/her preferences, the service contract selection module is invoked. This module is composed of three components that support:

- *Query Formulation*: the user selects pre-defined preferences (e.g., *I want to receive information on my mobile phone*) from a list, and he/she personalizes them by writing a text into a textual area, like the following *"I want a blanket insurance on the service delivery"*. User preferences, user profile and information extracted from the user history (i.e., information on past interactions between the user and services) represent the *contextual user information* that are used to formulate the multi-constraint query.
- *Filtering*: service contracts are filtered complying to the user category and contextual user information. The result is a set of *filtered service contracts*.
- *Query Evaluation*: the multi-constraint query is evaluated against the filtered service contracts. A ranked list of service contracts is returned to the user.

In our approach, we adopt the Policy Centered Meta-model (PCM)[1] as a meta-model for service contracts and contextual user information specifications. As shown in [13], the PCM outperforms other models by supporting: (i) expressive descriptions addressing qualitative contractual terms by means of logical expressions, quantitative terms by means of expressions including ranges and inequalities and, (ii) structured descriptions that aggregate different term descriptions into a single entity with an applicability condition.

The Query Formulation phase will be detailed in Section 4, whereas Filtering and Query Evaluation will be described in Section 5.

4 Multi-constraint Query Formulation

The simplest query formulation allows the user to select a query from a predefined list. This list is made up of the most frequent user queries, and each of them is formally defined by the service provider in the PCM format in order to easily represent the query constraints. But each predefined query is presented to

[1] The PCM formalizations in OWL and WSML-Flight are available at: http://www.siti.disco.unimib.it/research/ontologies/

the user as a textual description to ease the selection process. A query is formulated by means of constraints on data values: we allow the specification of both *precise* and *flexible* constraints. Precise constraints are specified on a selected attribute by a specific value of the attribute domain, e.g. *insurance=damage*. Flexible constraints can be specified on attributes with a numeric domain by a linguistic label which constraints the values of the attribute domain, e.g. *price = at most 40 €*. Formally, such a linguistic label is associated with the membership function of a fuzzy subset of the domain. Additional details related to the definition of query constraints are presented in Section 5.

In Listing 1, a PCM-formulation of the pre-defined query *"I'm looking for a CHEAP (i.e.,* price at most 40€*) delivery service by having an INSURANCE on damage"* is shown. In the above example, both a flexible (*price=at most 40€*), and a precise constraint (*insurance=damage*) are specified.

Listing 1. An example of user query in PCM format

```
nonFunctionalProperties
   dc#description hasValue "Request Instance for Logistic Operator"
endNonFunctionalProperties

pcm#hasNfp hasValue requestedPrice
pcm#hasNfp hasValue requestedInsurance

instance requestedPrice memberOf nfpo#PriceRequest
pcm#hasExpression hasValue requestedPriceExpression

instance requestedPriceExpression memberOf nfpo#PriceExpression
pcm#hasOperator hasValue pcm#atMost
pcm#hasParameters hasValue 40
pcm#hasUnit hasValue nfpo#euro

instance requestedInsurance memberOf nfpo#InsuranceRequest
pcm#hasExpression hasValue requestedInsuranceExpression

instance requestedInsuranceExpression memberOf nfpo#InsuranceExpression
pcm#hasOperator hasValue pcm#exist
pcm#hasParameters hasValue ins#damage
```

At this point, a user can personalize the selected query in three ways: (1) by modifying the pre-defined constraints, (2) by adding further constraints, and/or (3) by adding a short textual description. This way the user can provide more details and/or refine the constraints about the required service contracts. As an example, the value assigned to the precise constraint *insurance=damage* can be replaced by *insurance=fire&theft*.

Once the user has completed the formulation of his/her query, some additional constraints are automatically added based on the information obtained from the personal context, where constraints on both the user information and the user history are examined. The analysis of the user information (stored at registration time) determines additional precise constraints like the list of information channels that can be used to deliver information to the user. Instead, from the user history implicit user preferences are extracted, such as how many

times a specific service contracts has been employed by the user in the past (for an example see Section 6).

5 Filtering and Query Evaluation

As explained in Section 4, the *PCM-based multi-constraint query* is composed of two types of constraints: constraints on punctual values (data), and constraints on textual descriptions. In the following subsections, we will describe how the query is evaluated for filtering and ranking the service contracts.

5.1 Service Contract Filtering

The first step executed by the query evaluation process is the service contract filtering; such filtering is based on the user category affiliation and aimed to filter out from the set of service contracts the ones that do not relate to the current user. This is done by matching the user category to the contract applicability and then by removing service contracts that require categories that the current user does not belong to. For each service contract a category is defined by a set of applicability conditions (e.g.: user age, VAT owner) that a user must have. A user is associated with a category if and only if all the conditions are respected. As an example, if the category called *SeniorUser* has the applicability condition *"User must be at least 65 years old"*, and the current user is 35 years old, then service contracts related to this category will be filtered out.

5.2 Constraints Evaluation

The NFP expressed in a service contract are defined by both specific data (such as prices, insurance, ...) and textual descriptions. With each query constraint a constraint evaluation function (in short CF) is associated. The evaluation of a constraint produces a matching degree, in the interval $[0, 1]$, between the constraint itself and a service contractual term. In the following sub-sections, the evaluation functions are described in relation to the different types of constraints.

Flexible Constraints on numeric data values. As explained in Section 4, we allow the specification of flexible constraints on data values. The evaluation of these constraints, formally defined as fuzzy subsets of the considered attribute domains, is performed by means of membership functions that express the compatibility between the flexible constraints and the related attribute domains. We define a membership function as a parametric linear function the value of which is in the interval $[0, 1]$. An example of membership function for the *price=at most 40€* constraint is depicted in Figure 2: service contracts with prices lower (or equal) than the required one (e.g., 40€) will have a matching degree of 1, whereas service contracts with prices higher than 60€will have a matching degree of 0. The flexibility of the adopted solution is for the range of values between (40€,60€) where the matching degree will decrease as the price will continue to increase.

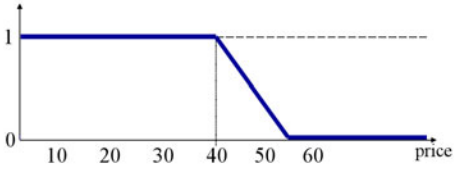


Fig. 2. Constraint evaluation function for *"at most 40"* constraint

Concept-based constraint evaluation. Service contracts could include NFPs that use concepts to represent their values. For example, the *Insurance* NFP assumes values (e.g., blanket, fire&theft) characterized by relations among them (e.g., a blanket insurance includes a fire&theft insurance). The evaluation of concept-based NFP constraints makes use of an ontology/thesaurus that maps all the possible values with all the relations among them. The matching degree is evaluated by the distance from the required value and the one provided by the service contract. Given the taxonomic hierarchy, the matching degree is maximum if the value required by the user is the same or a descendant of the one provided by the service contract. As opposite, if the required value is an ancestor of the provided one, the resulting matching degree is calculated according to the distance between the two values and it is normalized in the $[0, 1]$ set. Examples of concept-based constraint evaluation are in [12].

Set-based constraint evaluation. A third type of values that can be associated with NFP is called *Set-based*. Set-based constraints are defined by a set of values that have to be matched against the offered set based NFPs. As an example, the matching degree for the *InformationChannel* constraint is computed by applying the following formula:

$$CF_{InfoChannel}(sc, q) = \frac{|q_{InfoChannel} \cap sc_{InfoChannel}|}{|q_{InfoChannel}|}, \tag{1}$$

where sc is a service contract, q is the user query, $q_{InfoChannel}$ is the set of the Information channels specified in the user query, and $sc_{InfoChannel}$ is the set of the Information Channels provided by the service contract. The matching degree for the Information Channel will be in the set $[0, 1]$, where 1 represents a fully satisfied constraint and 0 will be returned for service contracts that do not provide any of the required characteristics in the user PCM-based query.

Keyword-based Evaluation. For the service contract textual description evaluation an IR approach is adopted to compute the user based query and the service contract matching; the relevance is estimated by a matching degree (in the set [0,1]) between the user needs and the textual description. The service contract description is a plain text that describes the service functionalities and characteristics in natural language. To index the textual description, simple IR techniques are applied, such as keyword extraction (words and terms are identified and extracted from the service contract textual description), and stop words

removal (the terms that are non-significant or do not provide any meaning are removed), respectively. For sake of simplicity we do not give a formal description of each of the above IR techniques, for more information and further details we recommend the reader to refer to the IR literature [14]. A user query is specified as a set of keywords that represent the main features that the service contract should have.

The previously cited IR functionalities enable to estimate the *relevance* degree between the user query keywords and the keywords extracted from the service contract description. To this aim we use a classical IR model for relevance evaluation called the *Vector Space Model* that represents each set of terms (or keywords) as vectors and can evaluate the relevance degree by the *similarity* between two vectors using a vector distance such as the Cosine similarity. It is worth noting that in reference to the term ambiguity problem raised in Section 1, some term disambiguation techniques [15] could be applied either at the indexing phase or at the query formulation. We will address this issue in a future research.

5.3 Overall Degree of Matching

The proposed aggregation function, a linear combination where the previously described constraint evaluation functions are aggregated to compute the overall service contract score, is defined by the following formula:

$$DoM(sc, q) = \frac{[\sum_{i=1}^{nc} CF_i(sc, q)] + CosSim(\overrightarrow{sc}, \overrightarrow{q})}{nc + 1}, \tag{2}$$

where nc is the number of constraints, CF_i is the constraint evaluation function for the query constraint i and $CosSim(\overrightarrow{sc}, \overrightarrow{q})$ is the service contract textual description evaluation performed using the Cosine Similarity on $\overrightarrow{sc}$ (i.e., keyword vector related to the service contract), and $\overrightarrow{q}$ (i.e., the keyword vector of the query). In the Formula 2 the overall service contract degree of matching is calculated as the average of the query constraint evaluation scores.

6 An Exhaustive Example

The logistic operator is the domain chosen to provide a complete example of the approach described in this paper. In the logistic operator domain a service provider offers one or more facilities to potential users. For instance, a service provider can offer freight transportation of cumbersome goods and traceability information to the consumers through different channels (e.g. SMS, e-mail, phone call). A transportation service is characterized by a set of functionalities, and it is associated with one or more service contracts. Furthermore, a service contract contains one or more *contractual terms* and it is addressed to specific user categories. Examples of NFPs on which contractual terms can be defined are:

- *payment method*: how the user can perform the payment (e.g., credit card, electronic transfer, cash on delivery);
- *insurance*: the type of prevention applied to the service;
- *price*: the amount of money that must be paid for the transportation and the traceability service;
- *hours to delivery*: the number of hours required for the service fulfilment;
- *information channels*: the channels (e.g., SMS, e-mail, phone call) used to send traceability information to the user.

Table 1 shows a set of service contracts for two hypothetical providers defined on the basis of the above mentioned NFP list. For example, (i) *pay-flex* offers maximum flexibility with respect to payment methods; (ii) *high-trace* is characterized by maximum flexibility with respect to information channels and languages; (iii) *secure* offers a maximum insurance coverage; (iv),(v) *fast-plus* and *fast* support fast transportation, and (vi) *cheap* performs transportation at lower price. Each contract is characterized by advantageous/disadvantageous contractual terms (e.g., the *fast* service contract offers a fast delivery but at an higher price).

Table 1. Examples of service contracts traceable freight transportation services

Provider A						
Contract	**PayMeth**	**Insurance**	**Price**	**HToDel**	**InfoC**	**Vector**
pay-flex	credit card, elect.transf, cash	Fire&theft	30	24-48	SMS, e-mail	*traceability, cheap, english*
high-trace	credit card, elect.transf	Fire&theft	35	48-72	SMS, e-mail, call	*traceability, english, italian*
secure	credit card, elect.transf	Blanket	35	48-72	SMS	*secure, traceability, english*
Provider B						
Contract	**PayMeth**	**Insurance**	**Price**	**HToDel**	**InfoC**	**Vector**
fast-plus	credit card	Fire&theft	40	12-24	SMS	*fast, traceability, english*
fast	credit card	Fire&theft	40	24-36	SMS	*fast, traceability, english*
cheap	credit card	-	20	72-96	SMS	*cheap*

Each service provider specifies the user categories that can access each offered service contract; such user categories are usually hierarchical in the sense that a higher level category includes the facilities of a lower level category. Examples of user categories are sketched in Table 2. The affiliation to the *BusinessOne* category is addressed to users that are VAT owners and mobile phone owners, instead the *BusinessPlus* is dedicated to users who respect all the BusinessOne conditions, and who have also used service contracts offered by a specific provider for at least 30 times in the past. *SilverUser* and *BronzeUser* categories have memberships conditions defined exclusively on the number of historical service utilizations. Finally, the *SeniorUser* category presents a condition on membership based on the user's age. Notice that BusinessPlus/BusinessOne and SilverUser/BronzeUser are hierarchical categories (e.g. a BusinessPlus user is also a BusinessOne user, but not viceversa).

Table 2. Examples of user categories

Contract	Category	Condition
pay-flex	*BusinessPlus*	VAT owner, mobile phone owner, 30 shipments
high-trace,secure	*BusinessOne*	VAT owner, mobile phone owner
fast-plus	*SilverUser*	20 shipments
fast	*BronzeUser*	10 shipments
cheap	*SeniorUser*	$\geq$ 65 years old

Let us suppose that the customer "Mario Rossi" has interacted several times with our system by selecting the appropriate service contracts for his specific tasks. In particular, the user has used *high-trace* and *secure* contracts from Provider A for 5 times and *fast* contract from Provider B for 20 times. The identification of the categories for Mario Rossi with respect to the service contracts is performed. The information considered from the user profile are the user's age, as well as the VAT and mobile phone information. From the history, the information that he has used for 10 times a service from ProviderA reserved to Business One users, and for 20 times a service from ProviderB reserved to Bronze users are considered. Thus, by analyzing the above user context and these conditions, the selected categories for Mario Rossi are: *BusinessOne, SilverUser,* and *BronzeUser*, respectively. For the filtering phase, the service contracts listed in Table 1 are filtered by using the previously obtained user categories affiliation. In Table 3 the user category affiliation has been associated with the related Service Contract. Thus, the service contracts *cheap* (with *SeniorUser* category) and *pay-flex* (with *BusinessPlus* category) will be filtered out and they will not be further analyzed in the ranking process.

Table 3. Example of service contracts category filtering

Provider	Contract	Category	User Membership
Provider A	*pay-flex*	*BusinessPlus*	no
Provider A	*high-trace*	*BusinessOne*	yes
Provider A	*secure*	*BusinessOne*	yes
Provider B	*fast-plus*	*SilverUser*	yes
Provider B	*fast*	*BronzeUser*	yes
Provider B	*cheap*	*SeniorUser*	no

Let us now suppose that Mario Rossi interacts with the system to formulate a query. He selects the following pre-defined query: *"I am a user who needs to perform a transportation of a valuable good. I am looking for a FAST (at most 48 hours) delivery service having a blanket INSURANCE. I would like to receive TRACEABILITY information about the transportation"*. The user decides to modify the query by introducing some specific constraints; in particular he modifies the flexible constraint FAST into *at most 24 hours*, and he adds new constraints not defined in the query, such as *price at most 40€*, and *payment method=credit card and electronic transfer*. The user's query is enriched with the information provided by the user at registration time: the user is interested

in *secure* and *cheap* services and his preferred information channels are *phone call* and *e-mail*.

For sake of simplicity, the evaluation process of each constraint will be described for one of the contracts listed before: the *fast-plus* contract. The same evaluation process will be then applied to the other contracts. By considering the *fast-plus* contract its NFPs evaluations are commented here below:

- **Hours to Delivery:** the evaluation of this NFP produces a matching degree of 1, since the service contract provides the delivery in 24hours as requested in the user query. The matching degree is calculated as explained in Section 5.2.
- **Insurance:** the *Fire & theft* insurance type provided by this contract is a subset of the insurance type required by the user (*Blanket*); the matching degree is 0.33 as the given contracts covers only one third of the *Blanket* one (the *Blanket* insurance is composed by the *Fire & theft*, *Damage* and *Loss* sub-insurances).
- **Price:** the service contract price (40€) is equal to what required from the user: the matching is fulfilled and the constraint evaluation function is 1.00 according to Section 5.2.
- **Payment Method:** the payment methods offered from this contract match only partially the user query: the *fast-plus* contract provides only the *credit card* method. The matching degree is 0.50.
- **Info. Channel:** the service do not provide any of the user requested Information Channels, for this reason the constraint evaluation function of this constraint is 0.00.
- **Description:** the service contract description contains only the terms *traceability* and *fast* defined in the user query. The estimated relevance degree is 0.50.

The GDoM matching degree for the *fast-plus* contract is finally evaluated as described in Formula 2; where $\sum_{i=1}^{nc} CF_i(sc, q) = 2.83$, and the final DoM degree is evaluated as $DoM(sc, q) = \frac{2.83+0.50}{6} = 0.555$.

7 Experimental Evaluation

In order to assess the effectiveness of the proposed service contract selection strategy we adopt the normalized discounted cumulative gain (NDCG) measure [16]. This metric has been designed in order to compare IR methods with respect to their ability to favour relevant search results. DCG, *discounted cumulative gain*, measures the gain of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom with the gain of each result discounted at lower ranks. In our scenario a result refers to a service contract, and for our tests we adopt the modified NDCG formulation proposed in [17]. This modification explicitly models a judgment value in addition to the ranking obtained after the application of the methodology presented in Section 5, and it normalizes the DCG values by comparing them with respect to an ideal

rank. The ideal rank is obtained as an agreement of a pool of experts. In detail, we asked to 3 experts, given both a set of queries and a user profile, to independently indicate a judgement for each service contract with the consequence to obtain 3 ideal ranks. After this, the experts have indicated a common assessor on them to provide a unique ideal rank.

Given a ranked result set of service contracts $\mathcal{S}_r$, and an ideal ordering of the same set of service contracts $\mathcal{S}_i$, the (DCG) at a particular rank threshold k is defined as $DCG(\mathcal{S}_r, k) = \sum_{i=1}^{k} \frac{2^{jdg(i)}-1}{log(1+i)}$, where $jdg(i)$ is the judgement (0=Bad, 1=Fair, 2=Good, 3=Excellent) at position i in set $\mathcal{S}_r$.

The ideally ordered set $\mathcal{S}_i$ contains all service contracts rated for the given query sorted descending by the judgement values. Formally, the NDCG at a particular rank threshold k is defined as:

$$NDCG(\mathcal{S}_r, k) = \frac{DCG(\mathcal{S}_r, k)}{DCG(\mathcal{S}_i, k)}, \tag{3}$$

Higher NDCG values correspond to better agreements with human judgements.

7.1 Experiments

To the best of our knowledge there is no benchmark defined to compare different service contract selection tools, consequently we have simulated the interaction of a user with our system. Thus, we have defined 32 service contracts from 5 distinct providers, and the NFPs on which contractual terms have been defined are those specified in Section 6. We asked to a user to perform three queries by increasing the complexity of each request. This means that for each new query a new constraint has been added. In details, the first query, i.e. Q_1= *"I am looking for a SECURE and FAST delivery service"*, has been selected by the user from the provided list without specifying any further detail. As indicated in Section 4, for each pre-defined query the constraints on attributes are identified; in query Q_1 the constraints are $insurance = blanket$ and $delivery <= 48\, hours$, respectively. In the second query, Q_2, the user specifies his/her meaning for the data *fast* as 24 for indicating a delivery services at the most of 24 hours. At the end, for the third query, Q_3, the user adds the data "traceability" in the free-text area in addition to 24 as a punctual value for the data-field *fast*. In order to show the effectiveness of our approach, we have performed simulations in different conditions by considering for each of them the above three queries as follows: (i) without considering the user context model (i.e., "case 1"), (ii) only by having the user information taken at registration time (i.e., "case 2"), (iii) only by considering the user history (i.e., "case 3"), (iv) only by considering information based on punctual values (i.e., "case 4"), (v) only by analyzing information obtained from textual descriptions (i.e., "case 5"), and at the end (vi) all the information provided in the previous steps that characterize the system described in this work (i.e., "case 6"). Thus, we have compared and evaluated the six approaches by applying the NDCG metric at various rank cuts (@5, @10, and @20). Fig. 3 shows the NDCG average values obtained for the above cases at different @-cuts.

By analyzing case 1, it emerges how the knowledge of user information allows to obtain better results in all the other cases where no additional information is considered with respect to the queries. In our system the usage of data prevails with respect to the textual description, and this implies a better performance in case 2 and case 4 with respect to case 5. Another consideration can be made by analyzing the use of the history information, case 3, where lower values are obtained with respect to the information taken at registration time. This means that our strategy gives more importance to the personal information of the user (i.e., his/her role/job, info languages, ...). By considering all cases our method (case 6) outperforms the other ones. This means that the proposed methodology produces higher NDCG as it preserves the ranking given by the ideal ranking better than the other cases.

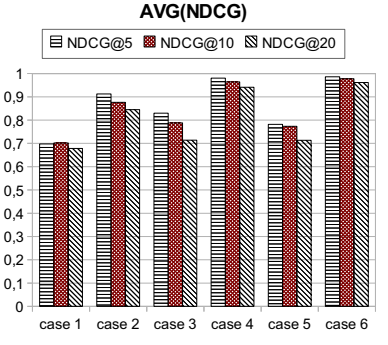


Fig. 3. Comparison of all the considered cases

8 Conclusions and Future Works

Service contract selection is an important factor to enhance service discovery. In this paper we have proposed a novel approach to support service contract selection based on semantic and IR techniques. The approach exploits precise and flexible preferences both explicitly defined by a user and implicitly inferred from his/her context. The user's preferences on the NFPs are formulated by means of a multi-constraint query that is used to filter and rank the service contracts offered by discovered services. The filtering is performed by evaluating the user categories, and the ranking is performed by aggregating the single constraint matching degrees of each service contract. Experimental results show the effectiveness of our approach to rank 32 service contracts from 5 distinct service providers according to 3 multi-constraint queries formulated by the user.

Our future research will address the problem of building a large benchmark of real service contracts to make comparative evaluations of different approaches possible. Moreover, we are investigating how to handle the management of qualitative NFPs (e.g., security and trust) which cannot be directly quantified. Finally, we are also studying how to integrate our approach with the aggregated search of data and services presented in [18].

References

1. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F., Kramer, B.: Service Oriented Computing Research Roadmap. In: Dagstuhl Seminar Proceedings 05462 (SOC) (2006)
2. Toma, I., Roman, D., Fensel, D.: On describing and ranking services based on non-functional properties. In: Third International Conference on Next Generation Web Services Practices (NWESP 2007), pp. 61–66. IEEE Computer Society, Washington, DC, USA (2007)
3. Comerio, M., Truong, H.-L., De Paoli, F., Dustdar, S.: Evaluating Contract Compatibility for Service Composition in the seCO$_2$ Framework. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 221–236. Springer, Heidelberg (2009)
4. Wang, Y., Tang, B., Huan, Z.: Study on contract selection decision based on hybrid supply chains. In: Proc. of the Inter. Conf. on Measuring Technology and Mechatronics Automation. ICMTMA 2010, Washington, DC, USA, pp. 572–575 (2010)
5. Talluria, S., Leea, J.Y.: Optimal supply contract selection. International Journal of Production Research 48(24), 7303–7320 (2011)
6. Yu, H.Q., Reiff-Marganiec, S.: A method for automated web service selection. In: Proc. of the Congress on Services (SERVICES), pp. 513–520 (2008)
7. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: Proc. of the 16th International Conference on World Wide Web (WWW 2007), pp. 1013–1022. ACM, New York (2007)
8. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A QoS-Aware Selection Model for Semantic Web Services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 390–401. Springer, Heidelberg (2006)
9. Garcia, J.M., Toma, I., Ruiz, D., Ruiz-Cortes, A.: A service ranker based on logic rules evaluation and constraint programming. In: Proc. of 2nd Non Functional Properties and Service Level Agreements in SOCWorkshop (NFPSLASOC), Dublin, Ireland (2008)
10. Bajaj, S., Box, D., Chappell, D., et al.: Web Service Policy 1.2 - Framework (2006), `http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/`
11. 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)
12. Palmonari, M., Comerio, M., De Paoli, F.: Effective and Flexible NFP-Based Ranking of Web Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 546–560. Springer, Heidelberg (2009)
13. De Paoli, F., Palmonari, M., Comerio, M., Maurino, A.: A Meta-Model for Non-Functional Property Descriptions of Web Services. In: Proc. of the IEEE International Conference on Web Services (ICWS), Beijing, China, pp. 393–400 (2008)
14. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
15. Stokoe, C.: Word sense disambiguation in information retrieval revisited. In: ACM SIGIR, pp. 159–166 (2003)
16. Järvelin, K., Kekäläinem, J.: Cumulated gain-based evaluation of ir techniques. ACM Transaction on Information Systems (TOIS) 20(4), 422–446 (2002)
17. Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S.: Diversifying search results. In: WSDM 2009, pp. 5–14. ACM, New York (2009)
18. Palmonari, M., Sala, A., Maurino, A., Guerra, F., Pasi, G., Frisoni, G.: Aggregated search of data and services. Information Systems 36(2), 134–150 (2011)

Modeling and Managing Variability in Process-Based Service Compositions

Tuan Nguyen, Alan Colman, and Jun Han

Faculty of Information and Communication Technology,
Swinburne University of Technology, Melbourne, Australia
{tmnguyen,acolman,jhan}@swin.edu.au

Abstract. Variability in process-based service compositions needs to be explicitly modeled and managed in order to facilitate service/process customization and increase reuse in service/process development. While related work has been able to capture variability and variability dependencies within a composition, these approaches fail to capture variability dependencies between the composition and partner services. Consequently, these approaches cannot address the situation when a composite service is orchestrated from partner services some of which are customizable. In this paper, we propose a feature-based approach that is able to effectively model variability within and across compositions. The approach is supported by a process development methodology that enables the systematic reuse and management of variability. We develop a prototype system supporting extended BPMN 2.0 to demonstrate the feasibility of our approach.

Keywords: Process variability, service variability, variability management, service composition, feature modeling, model mapping, Software Product Line (SPL), Model Driven Engineering (MDE).

1 Introduction

Process-based service compositions are efficient approaches for developing composite services and applications using process modeling techniques. The two de facto standards for this purpose are BPMN (Business Process Modeling Notation) for modeling purposes and BPEL (Business Process Execution Language) for execution purposes. Generally, in both techniques, each composite service is described by a process model which specifies the flow of activities (i.e. *control flow*), the interaction between the process and partner services (i.e. *message flow*), and the way data is moved throughout the process (i.e. *data flow*).

Due to the diversification and the personalization of service consumption, service variability has become an important factor in the lifecycle of service development [1, 2]. Service variability is defined as the ability of a service/process to be efficiently extended, changed, customized or configured for use in a particular context [3]. Such variability can originate from a service provider wishing to provide different versions of the same service for different market segments or with different pricing models, or

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 404–420, 2011.

from service consumers wishing to customize a service to match their particular business requirements.

Service variability brings about a new type of service, namely *customizable service,* in service ecosystems [4]. *A customizable service* is a service whose *runtime customization* by a consumer will result in a particular service variant matching the consumer's requirements [5-7]. For services with a large number of service variants, the deployment of customizable services, instead of conventional services, will much benefit service consumers. This is because there is disadvantage with either deploying an all-in-one non-customizable service or deploying all service variants separately. In the first case, the resulting non-customizable service has a large service description most of which is not relevant to one particular consumer. In the second case, it is difficult for service consumers to recognize the similarity and difference among those service variants in order to select the most appropriate one [2].

Modeling and managing variability in customizable composite services are challenging. There are two key concerns that need to be addressed [8]. Firstly, how to model *variation points* and *variants*? Secondly, how to capture *dependencies* among variabilities? Variability dependencies describe such relationships as the binding of variants at one or several variation points requires or excludes the binding of variants at other variation point(s). We identified in our previous work that, in the service computing context, besides *variability intra-dependencies* which represent dependencies within a service composition, there are *variability inter-dependencies* which represent dependencies between the composition and its customizable partner services [9]. Variability inter-dependencies reflect the situation when the runtime resolution of variability in the composition requires the runtime resolution of variability at partner services. And this process may also cause a *ripple effect* in the service ecosystem since service composition is recursive.

In terms of variability management, Software Product Line (SPL) is a successful paradigm that builds upon techniques for systematic identification and management of variability [10]. Many related efforts have exploited concepts and techniques from SPL in addressing variability in process-based service compositions, e.g. [11-14]. These approaches are able to capture variability and variability dependencies within the control flow and the data flow of a process model. However, all these efforts fail to capture *variability inter-dependencies*. Consequently, these approaches are not capable of managing variability in such service compositions that are aggregated from customizable partner services.

To address this problem, we propose a comprehensive approach to modeling and managing variability in process-based service compositions. In particular, we extend the BPMN 2.0 metamodel to incorporate variation points and variants with respect to not only control flow and data flow but also message flow. We then extend a feature modeling technique from SPL to capture variability dependencies within and across service compositions. We also specify a process development methodology that elaborates how to systematically model and manage variability at design time, as well as instantiating variability at runtime. The methodology builds upon Model Driven Engineering (MDE) techniques to automate large parts of its operations.

The structure of the paper is as follows. Section 2 presents a discussion of related work. In section 3, we describe a motivating scenario, followed by the explanation of techniques underpinning our research in section 4. Section 5 presents our approach to modeling variability and variability dependencies. We describe an approach to developing service compositions with managed variability in section 6. The prototype system is described in section 7 before our conclusion of the paper in section 8.

2 Related Work

A number of works has been proposed for modeling and managing variability in process-based service compositions [1, 2, 11-16]. In general, they can be classified into two categories.

The first category consists of work that aims to extend BPEL [12, 13, 15]. In particular, Chang [15] and VxBPEL [12] extend the XML schema for BPEL in order to incorporate information about variation points and variants into the business process definition. In contrast, Mietzner [13] uses a separate variability descriptor to define the location of variation points in the business process definition and possible variants. In general, the advantage of extending BPEL is that an executable process variant can be automatically derived by resolving all variation points. However, VxBPEL and Chang's work suffer from tangled and scattered business process definitions. Mietzner's work overcomes this problem by using a separate variability description. Nevertheless, since variability is modeled at the implementation level, these approaches become very complex due to the large number of variation points.

Work in the second category focuses on extending process models described using BPMN or UML Activity diagrams [1, 2, 11, 14, 16]. The general approach for these efforts is to extend the process metamodel so that variation points and variants can be explicitly introduced. Since variability is modeled at the architectural level, the number of variation points is much smaller than the ones at the process definition level. Therefore, these approaches overcome the complexity issue of the ones in the first category. However, except [14, 16], all other works only focus on variation points and variants with respect to the control flow of process models. Works in [14, 16] takes a step further to consider the data flow as well. Consequently, only these works can support the derivation of executable process variants.

Although *variability intra-dependencies* have been considered in most of the related work, e.g. [2, 12], the major issue with work in both categories is that they are not able to capture *variability inter-dependencies*. All work builds on an assumption that all partner services are not customizable. Consequently, those approaches are not applicable to composite services orchestrated from partner services some of which are customizable.

3 Motivating Scenario

A Content Management System (CMS) Provider wants to develop a composite service which allows various Content Providers to post news entries (cf. Figure 1).

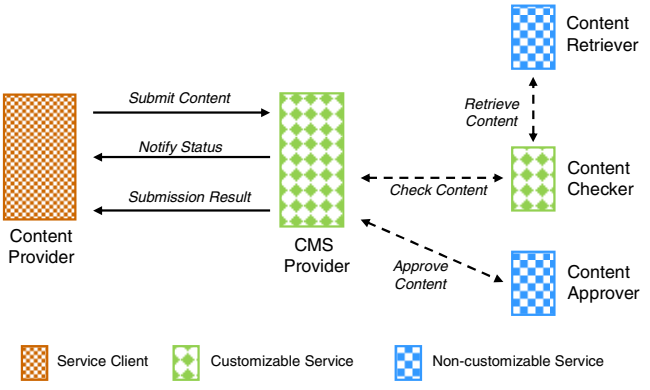


Fig. 1. A news posting composite service

Due to different requirements from Content Providers, the CMS Provider will support the following variability in its business process:

- Content Providers may choose to directly send news entries or specify an external URL resource as the content source.
- Content Providers may also opt to receive posting status update from the CMS Provider.

There are many services available that the CMS Provider may reuse in implementing its process. For example, there are services for checking the correctness of the news entry (e.g. grammar check) and there are services for approving the news submission (e.g. checking the publishing policies). In this case study, the CMS Provider will utilize two of those services, namely ContentChecker service and ContentApprover service. While the ContentApprover service is a *non-customizable service* accepting the news content and returning the approving result, the ContentChecker service is a *customizable service*. It has two service variants. The first service variant accepts the news content, performs the checking and then returns the result. The second variant accepts a URL and invokes the ContentRetriever service for retrieving the content before performing the checking. The utilization of this ContentChecker service frees the CMS Provider from the overhead of retrieving the content in a case a URL is provided from a Content Provider. Consequently, variability in the CMS Provider will depend on the variability in the ContentChecker.

4 Underpinnings of Our Approach

In this section, we explain the techniques that underpin our approach. In particular, we briefly describe feature modeling techniques from SPL, our solution for describing variability of customizable services based on the concept of features, and how the service variability description is utilized to support runtime service customization.

4.1 Feature Modeling Technique

Feature modeling are techniques in SPL for capturing the commonalities and differences among a family of software products [17]. Features are visible characteristics that are used to differentiate one family member from others. A feature model is represented as a hierarchically arranged set of features with *composed-by* relationship between a parent feature and its child features. In addition, there are cross-tree *constraints* that typically describe inclusion or mutual exclusion relationships. A feature model is an efficient abstraction of variability and provides an effective means for communicating variability between different stakeholders. In addition, it helps to drive the design and the development of variability throughout all stages of the product line development.

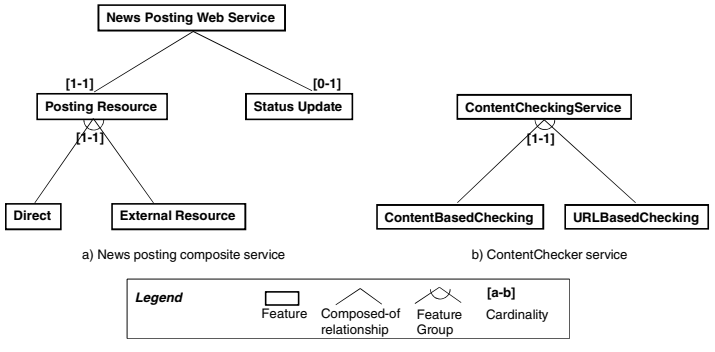


Fig. 2. Examples of feature model

While there are many manifestations of feature modeling techniques, e.g. [18-20], in our work we exploit Czarnecki's cardinality-based feature modeling technique [21]. The main reason for this choice is that the concepts of *feature cardinality* and *group cardinality* well suit the needs of service customization. A *feature cardinality*, associated with a feature, determines the lower bound and the upper bound of the number of the feature that can be part of a product. A *group cardinality*, associated with a parent feature of a group of features, limits the number of child features that can be part of a product when the parent feature is selected.

Figure 2a demonstrates a feature model representing variability of the news posting composite service. Based on their cardinality, *"Posting Resource"* is a mandatory feature, while *"Status Update"* is an optional feature. In addition, *"Posting Resource"* is a group of alternative features. It means that, all consumers need *"Posting Resource"* capability, which can be either *"Direct"* or *"External Resource"*, while they can opt to have *"Status Update"* capability when consuming the service. Similarly, Figure 2b demonstrates the feature model for the ContentChecker service. Its variability is represented as a group of two alternative features, *"ContentBasedChecking"* and *"URLBasedChecking"*.

4.2 Feature-Based Service Variability Description

In order to describe variability of a services and facilitate service customization, we define a new language, namely WSVL (Web Service Variability description Language), based on the concept of features. Due to space limitation, we briefly describe the language through the example of the ContentChecker service without going into the detail of motivations and requirements behind it. The language (cf. Figure 3) has three parts. Firstly, the *ServiceDescription* part describes the capability of the service and represents the superset of the capability of all service variants. In this scenario, the service description consists of two operations, *ContentBasedCheck* and *URLBasedCheck*, using different message formats for realizing two service variants. The service description is only expressed at the abstract level for modeling purposes. Once a service variant is derived, its complete service description, described in WSDL, will be generated. Secondly, the *FeatureDescription* part describes the variability of the service in term of features. It is actually the serialization of the feature model for the corresponding service (cf. Figure 2b). And thirdly, the *MappingDescription* part describes the mapping from variant features in the feature description part to variable capability in the service description part as a set of links. For example, the first link shows the mapping between the feature *"ContentBasedChecking"* and the corresponding operation, *"ContentBasedCheck"*. In general, a link represents 1-to-m mapping between a feature and service capabilities. The service variability description provides information on what capability of the corresponding service is available in a service variant given a feature configuration[1].

Fig. 3. Service variability description for the ContentChecker service

[1] A feature configuration is a specialized form of a feature model in which all variability is resolved, i.e. all variant features are selected or removed.

4.3 Feature-Based Service Customization Framework

In previous work, we developed a feature-based service customization framework that allows service consumers to customize a service at the business level [5]. In particular, based on the service variability description, service consumers can select features they need and unselect features they do not need. Feature selection has to conform to feature cardinality, group cardinality and constraints described in the feature model to generate a valid feature configuration. The feature configuration is then communicated back to the service provider so that the service provider can generate a service interface description and a service implementation bound to the service interface description. This service variant is then dynamically deployed to an endpoint so that the service consumer can invoke. In previous work, we have focused on how to model, manage and instantiate variability at the service interface level. The work in this paper complements that work in addressing the issues of how to model and manage variability in the service implementation (i.e. business process), and then generating a variant based on a particular feature configuration. In addition, the work in this paper also exploits that technique for customizing partner services.

5 Modeling Variability in Process-Based Service Compositions

As explained, variation points and variants need to be explicitly introduced into process models. To this end, there are two requirements for our approach. Firstly, the complexity in modeling variability needs to be alleviated. Modeling variability in business processes is challenging because of the existence of a large number of variation points and variants. Therefore, it is important to reduce the number of variation points and variants that need to be considered. Secondly, the approach needs to support variability instantiation, i.e. the runtime derivation of executable process variants for customization purposes.

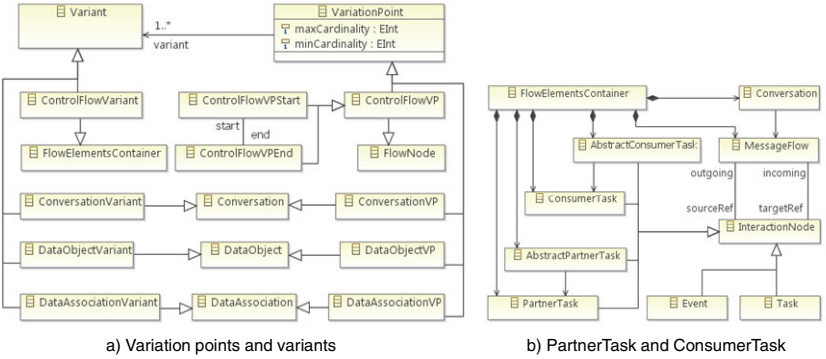


Fig. 4. Process metamodel extension

In order to satisfy these two requirements, we have decided to model and manage variability within process models described by BPMN. The advantage of using BPMN or UML Activity diagram over BPEL is that the number of variation points and variants within a BPEL definition is much more than the ones within a BPMN or UML model. And we selected BPMN over UML Activity because of its wide acceptance and well support. At the time we developed our approach, BPMN 2.0 has been released and it provided a sufficient metamodel for modeling process-based service compositions. However, there is no significant difference between the two. One can easily apply the solution we present here over to UML Activity diagrams and achieve similar results. In addition, we exploit MDE techniques in our approach to automate large parts of the solution and facilitate not only variability management but also variability instantiation.

5.1 Extending BPMN for Representing Variation Points and Variants

Our key idea for introducing variability modeling capability into process modeling is to define a general metamodel for variation points and variants, then weave this metamodel into the BPMN 2.0 process metamodel to make it capable of supporting variability. The extension will focus on all three aspects of service compositions: control flow, data flow, and message flow. In addition to modeling variability in the control flow and data flow as done in related work, our approach takes a further step to capture variability in the message flow as well. Hence, the approach is capable of not only supporting executable process variant derivation, but also capturing variability inter-dependencies. The result of this is shown in Figure 4.

The general variability metamodel is composed of two elements: *VariationPoint* and *Variant*. A *VariationPoint* represents any place in the process model where variability can occur. Each *VariationPoint* is associated with a set of *Variants* from which one or several will be bound to *VariationPoint* when the variability is resolved. The attributes *minCardinality* and *maxCardinality* define how many *Variants* should be bound to one *VariationPoint*. These attributes have the same semantics as the cardinality concept adopted in the feature modeling technique.

Variation point in the control flow can be interpreted as any location in the process model at which different execution paths can take place. Therefore, we introduce new *FlowNode* elements, namely *ControlFlowVP,* and its two direct inheritances, namely *ControlFlowVPStart* and *ControlFlowVPEnd*, for representing starting point and end point of each variability. Variants in the control flow can be arbitrary process fragments. Therefore, *ControlFlowVariant* is inherited from *FlowElementContainer*.

Variability in data flow can be considered as different ways for storing data (i.e. *DataObject*) or different ways for moving data around (i.e. *DataAssociation*). Since variants in data flow are usually alternative variants, we model both variation points and variants as inherited elements from the same element type. That is, for variability of *DataObject*, we define both variation points, i.e. *DataObjectVP*, and variants, i.e. *DataObjectVariant*, as inherited elements from *DataObject*. A similar approach applies with *DataAssociation*, *DataAssociationVP*, and *DataAssociationVariant*.

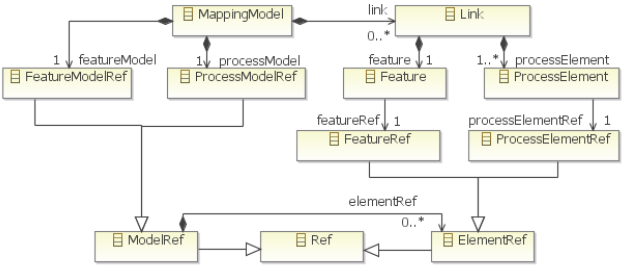


Fig. 5. Mapping metamodel

Variability in message flow can be seen as alternative *Conversations* between two parties, i.e. the process and a partner service (or a consumer). Therefore, in a similar fashion to modeling variability in data flow, we model both variation points, i.e. *ConversationVP*, and variants, i.e. *ConversationVariant*, as inherited elements from *Conversation*. In addition, we introduce new elements, namely *PartnerTask* and *AbstractPartnerTask* (cf. Figure 4b). A *PartnerTask* models a task performed by a partner service. An *AbstractPartnerTask* models a set of alternative *PartnerTasks* and it represents a variable capability provided by a partner service. The introduction of *PartnerTask* and *AbstractPartnerTask* facilitates the modeling of variability inter-dependencies since variability inter-dependencies will be the mapping between these elements and variant features of partner services, namely *variant partner features*. In a similar fashion, we introduce *ConsumerTask* and *AbstractConsumerTask* for the interaction between the business process and its consumers. These extended elements facilitate the generation of the service variability description for this service composition.

5.2 Modeling Variability Intra-dependencies

Variability intra-dependencies represent dependencies among variation points and variants within a process model. Therefore, the intuitive way for modeling variability intra-dependencies is to model variability constraints as elements of the process models. However, the disadvantage of this approach is twofold. Firstly, the resulting process model will be swamped with dependencies information and become too complex. Secondly, since variability intra-dependencies are embedded in process model definitions, it becomes harder to identify conflicts in such dependencies [22].

In fact, variability in the process model is the *realization of variability* in the feature model of the service composition. In other words, the identification and modeling of variation points and variant in a process are driven by variant features in the feature model. *Variability intra-dependencies among different variants come from the fact that those variants realize variant features.* Therefore, we exploit the model mapping technique for relating variation points and variants in the process model with variant features in the feature model. We refer to this type of mapping model as *FeatureTask mapping model*. Due to mentioned *realization relationships*, those

mappings along with feature constraints in the feature model account for all variability intra-dependencies in the process model. In addition, this approach has the following advantages in comparison with embedding constraints in the process definition. On the one hand, it helps to separate variability constraint information from the process model, thus simplifies the definition. On the other hand, the validation of process configuration is led to the validation of a feature configuration, which is well-studied in SPL [23].

The mapping metamodel for this purpose is shown in Figure 5. A *MappingModel* relates variant features in a feature model, referenced by *FeatureModelRef*, with variants in a process model, referenced by *ProcessModelRef*. It is composed of *Links* and each *Link* consists of a *Feature* and at least one *ProcessElement*. *Feature* and *ProcessElement* reference elements in the feature model and the process model respectively. In this way, each *Link* enables a feature to be mapped to one or several variant process elements in the process model.

5.3 Modeling Variability Inter-dependencies

Variability inter-dependencies represent dependencies between variability in the process model and variability in partner services. Since variability of partner services can be described using feature models (cf. Figure 3), in a similar fashion as modeling variability intra-dependencies, we exploit the model mapping technique to model these dependencies. The main difference between the mapping model for variability intra-dependencies and the mapping model for variability inter-dependencies is the origin of variant features. While variability intra-dependencies is modeled with respect to variant features in the feature model of the service composition, variability inter-dependencies is modeled with respect to variant partner features.

In particular, a mapping model for variability inter-dependencies captures the correspondence between *PartnerTasks* within the process model and variant partner features. We refer to this mapping model as *PartnerTaskFeature mapping model*. It should be noted that between *PartnerTasks* and variant partner features, there does not exist a "natural" *realization relationship* as the ones for variability intra-dependencies. If the identification and modeling of *PartnerTasks* and *AbstractPartnerTasks* are driven by the variant partner features, such *realization relationships* establish. Otherwise, the identification and modeling of *PartnerTasks* and *AbstractPartnerTasks* are independent of variant partner features, and *realization relationships* may not exist. In this way, all variability inter-dependencies exist by chance. Therefore, high inter-dependency between the service composition and partner services represents high chance of reuse of service variability from partner services toward the service composition. In later section, we describe a process development methodology that systematically increases the chance of reuse of service variability.

Since variability in the feature model of the business process is mapped to variability in the process model, i.e. *FeatureTask mapping model*, and a part of variability in the process model, i.e. *PartnerTasks*, is mapped to variability in partner feature models, i.e. *PartnerTaskFeature mapping model*, it is possible to generate the

mapping from variant features in the feature model of the service composition to variant partner features. This mapping model conforms to a similar mapping metamodel as Figure 5 and allows us to capture variability inter-dependencies at the highest level of abstraction, i.e. the feature level. We refer to this type of mapping model as *FeatureFeature mapping model*. In summary, there are two types of feature mapping models for representing variability inter-dependencies: *PartnerTaskFeature* and *FeatureFeature mapping models*.

6 A Bottom-Up Process Development Methodology

In this section, we describe a methodology for developing service compositions with systematic management and reuse of variability (cf. Figure 6). One key feature of the methodology is that, it increases the chance of reusing service variability provided by partner services. To this end, variability information from partner services is explicitly utilized in driving the identification and modeling of variability within the business process.

6.1 Overview

In the first activity, the capability of the service composition is modeled using the feature modeling technique. The result of this activity is a feature model capturing commonalities and variabilities of the composite service to be. Given a model of desired features, the next activity will be the selection of partner services that can be used for the service composition. There are two types of services that will be selected: *(conventional) non-customizable partner services* and *customizable partner services*. The explicit selection of customizable partner services helps to reduce overhead of addressing variability within the service composition. Customizable partner services come with service variability descriptions.

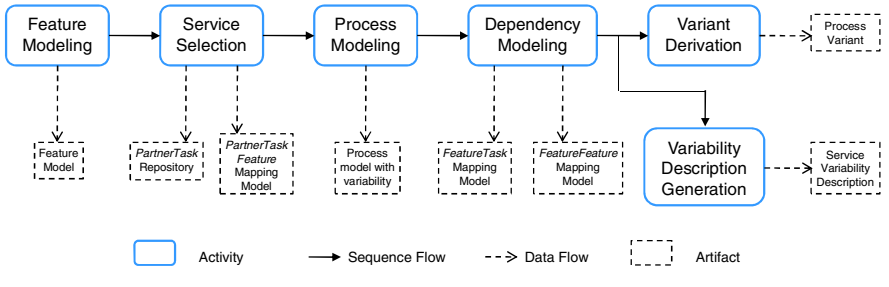


Fig. 6. A methodology for developing process-based service compositions

During the second activity, both non-customizable partner services and customizable partner services are transformed into a set of partner tasks that will be selectable for modeling the process. *A partner task is an operation provided by a partner service that is responsible for an atomic message flow between the partner*

service and the service composition. While non-customizable partner services are transformed to a set of non-customizable partner tasks, results of transforming customizable partner services are sets of alternative partner tasks. For each set of alternative partner tasks, we also generate an abstract partner task representing all partner tasks in the set. Since the variability of customizable partner services are expressed as feature models with mapping to customizable capabilities, we also derive mapping models that represent the correspondence between alternative partner tasks and variant partner features, i.e. *PartnerTaskFeature mapping models*. Consequently, results of the service selection activity are a repository of (alternative) partner tasks and *PartnerTaskFeature mapping models*. It should be noted that in this methodology, the *PartnerTaskFeature mapping model* is intentionally generated before modeling the process.

In the third activity, the business process for the service composition is modeled using the extended metamodel. The identification of variation points and variants are based on the feature model identified in the first activity. Tasks from the partner task repository will be used to model the message flow between the service composition and partner services. The selection of (alternative) partner tasks and abstract partner tasks from the partner task repository will not only facilitate the reuse of variability provided by partner services in the process modeling, but also enable the use of already generated *PartnerTaskFeature mapping model* in capturing variability inter-dependencies. The result of this activity is a process model with variability.

In the next activity, the model mapping technique is exploited to first model variability intra-dependencies. That is, all variation points and variants in a process model are mapped to variant features in the feature model of the business process. The result is a *FeatureTask mapping model*. Since the *PartnerTaskFeature mapping model* is already produced, model transformation techniques are utilized to automatically generate *FeatureFeature mapping model* as mentioned.

The resulting software artifacts of the first four activities will be used in two different ways. Firstly, they are used for the derivation of process variants given a particular feature configuration as the result of a customization (i.e. *Variant Derivation* activity). Secondly, those software artifacts are used to generate the variability description of the resulting service composition (i.e. *Variability Description Generation* activity) which can contribute to other service compositions. Due to space limitation, we just describe the first usage in the following subsection.

6.2 Deriving Executable Process Variants

While the modeling of variability and variability dependencies is a design time process, the derivation of an executable process variant usually happens at runtime. This is triggered when the service composition is customized by consumers or the service provider itself. As explained, the customization is performed using the feature model of the service composition (cf. section 4.3) and generally requires the runtime customization of respective partner services.

Given a feature configuration of the composition, we exploit model transformation techniques as follows to derive a particular executable process variant:

1. The *FeatureTask mapping model* is referenced for specializing the process model. The process model is actually a model template which is the superset of all process variants. Therefore, those process elements, which are mapped to selected features, are maintained while those process elements, which are mapped to removed features, are purged from the process model. The result of this task is an abstract process variant which does not have variability but still contains partner tasks and consumer tasks. The detail of specializing a model template can be found in our previous work [5].
2. The *FeatureFeature mapping model* is referenced for generating a feature configuration for each customizable partner service. These feature configurations are used to customize corresponding partner services and produce particular partner service variants.
3. From the abstract process variant and partner service variants, an executable process variant is generated. We presume the use of BPEL for the executable process. It is important to note that the existence of partner tasks in the abstract process variant will help to create partner links and accurate service invocation between the process variant and partner service variants.
4. Finally, based on the information of consumer tasks in the abstract process variant, the service interface description of this process variant is generated.

At the end of this activity, a fully executable process variant that matches the given feature configuration is generated along with a service interface description. The process variant will invoke a set of automatically customized partner service variants.

7 Prototype Implementation

We have developed a prototype system for validating the feasibility of our approach. Key components are an Eclipse plugin for modeling business processes along with their variability (cf. Figure 7) and a model mapping tool for capturing all types of

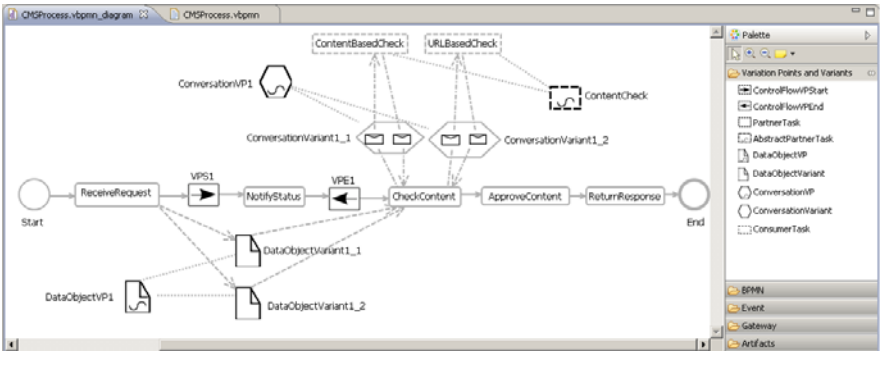


Fig. 7. A screenshot of modeling business processes with variability

variability dependencies (cf. Figure 8). Using our business process modeling tool, we successfully modeled the case study with all possible variation points and variants. This case study, despite of its simplicity, cannot be modeled by any approach in related work because all those approaches do not cater for the variability of partner services. In the following paragraphs, we introduce these key components.

Figure 7 is a screenshot of our process modeling tool. From the BPMN 2.0 metamodel, we extracted a subset that contains all model elements relevant to service compositions. We then introduced our process metamodel extension into the extracted metamodel. Our Eclipse plugin enables the development of any business process conforming to the extended metamodel. Modelers can select existing and new process elements from the right Palette tool. The screenshot displays the process model for the case study with three variation points: one variation point in the control flow, namely *VPS1* and *VPE1* for *ControlFlowVPStart* and *ControlFlowVPEnd*, one variation point in the data flow, namely *DataObjectVP1*, and one variation point in the message flow, namely *ConversationVP1*. *ConversationVP1* is associated with two alternative variants, namely *ConversationVariant1_1* and *ConversationVariant1_2*. While *ConversationVariant1_1* represents the message flow between the *"CheckContent"* task and the "*ContentBasedCheck"* partner task, *ConversationVariant1_2* represents the message flow between the same *"CheckContent"* task and the *"URLBasedCheck"* partner task. *"ContentBasedCheck"* and *"URLBasedCheck"* are alternative partner tasks associated with the same *AbstractPartnerTask*, namely *"ContentCheck"*. These *PartnerTasks*, *AbstractPartnerTask,* as well as the *PartnerTaskFeature* mapping model are generated from the service variability description of the ContentChecker service (cf. Figure 3). Modeling variability in this way enables the capturing of variability inter-dependencies between the CMS Provider and the ContentChecker service.

Figure 8 is a screenshot depicting how to capture variability intra-dependencies, i.e. *FeatureTask* mapping model, using our model mapping tool. In the screenshot, the feature model is presented in the left panel, while the process model is presented in the right panel and the middle panel presents the mapping model. Three mapping links are created in the screenshot. The first link associates the "Direct" feature with one *DataObjectVariant* and one *ConversationVariant*. Similarly, the second link associates the "External Resource" feature also with one *DataObjectVariant* and one *ConversationVariant*. The third link associates the "Status Update" feature with one *ControlFlowVP*. The creation of model elements is based on the context menus as shown in Figure 8. This component is implemented as an extension to Atlas Model Weaver (AMW) [24]. We then perform model transformations using Atlas Tranformation Language (ATL) [25] to derive *FeatureFeature* mapping model. As explained, *PartnerTaskFeature* mapping model and *FeatureFeature* mapping model account for variability inter-dependencies.

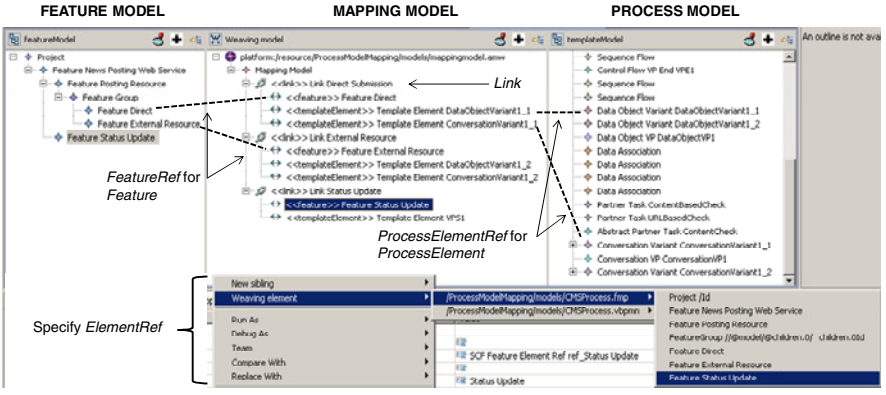


Fig. 8. A screenshot of a mapping model between a feature model and a process model

8 Conclusion

In this paper, we have proposed a feature-oriented approach to modeling and managing variability in process-based service compositions. We have extended the BPMN 2.0 metamodel for introducing variation points and variants in all three aspects of service compositions, i.e. control flow, data flow, and message flow. These extensions enable not only comprehensive modeling of variability, but also the generation of executable process variants as the result of a service customization. In addition, we have introduced a feature mapping technique for capturing not only variability intra-dependencies among variants within a process model, but also variability inter-dependencies between variants in a process model and variants in partner services. Consequently, our approach is able to address the situation when a customizable composition is orchestrated using partner services some of which are customizable. This is not achievable using existing approaches.

We have also described a methodology that facilitates the development of business processes conforming to the extended process metamodel with systematic variability management. The key advantage of the methodology is the systematic exploitation of variabilities provided by partner services to increase the chance of reusing variability. The methodology exploits MDE techniques for automating most parts, especially the generation of executable process variants. In addition, we present a prototype system for demonstrating the feasibility of our approach.

As future work, we plan to develop techniques for the generation of service variability description from the process model leading to a framework for the recursive delivery of customizable services in service ecosystems.

Acknowledgments. This research was carried out as part of the activities of, and funded by, the Smart Services Cooperative Research Centre (CRC) through the Australian Government's CRC Programme (Department of Innovation, Industry, Science and Research).

References

1. Sun, C.-A., et al.: Modeling and managing the variability of Web service-based systems. Syst. & Softw. 83(3), 502–516 (2009)
2. Hallerbach, A., et al.: Capturing variability in business process models: the Provop approach. Softw. Maint. & Evol.: Res. & Pract. 22(6-7), 519–546 (2010)
3. Svahnberg, M., et al.: A taxonomy of variability realization techniques: Research Articles. Softw. Pract. Exper. 35(8), 705–754 (2005)
4. Barros, A.P., et al.: The Rise of Web Service Ecosystems. IT Prof. 8(5), 31–37 (2006)
5. Nguyen, T., et al.: A Feature-Oriented Approach for Web Service Customization. In: IEEE Int. Conf. on Web Services, pp. 393–400 (2010)
6. Stollberg, M., Muth, M.: Service Customization by Variability Modeling. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 425–434. Springer, Heidelberg (2010)
7. Liang, H., et al.: A Policy Framework for Collaborative Web Service Customization. In: Proc. of the 2nd IEEE Int. Sym. on Service-Oriented System Engineering (2006)
8. Schmid, K., et al.: A customizable approach to full lifecycle variability management. Science of Computer Programming 53(3), 259–284 (2004)
9. Nguyen, T., et al.: Managing service variability: state of the art and open issues. In: Proc. of the 5th Int. Workshop on Variability Modeling of Software-Intensive Systems (2011)
10. Pohl, K., et al.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
11. Hadaytullah, et al.: Using Model Customization for Variability Management in Service Compositions. In: IEEE Int. Conf. on Web Services 2009 (2009)
12. Koning, M., et al.: VxBPEL: Supporting variability for Web services in BPEL. Information and Software Technology 51(2), 258–269 (2009)
13. Mietzner, R., et al.: Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In: IEEE Int. Conf. on Services Computing 2008 (2008)
14. Razavian, M., et al.: Modeling Variability in Business Process Models Using UML. In: 5th Int. Conf. on Information Technology: New Generations 2008 (2008)
15. Chang, S.H., et al.: A Variability Modeling Method for Adaptable Services in Service-Oriented Computing. In: Proc. of the 11th Int. Conf. on Software Product Line (2007)
16. Schnieders, A., et al.: Variability Mechanisms in E-Business Process Families. In: Proc. of Int. Conf. on Business Information Systems, pp. 583–601 (2006)
17. Kang, K.C., et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, in Technical Report, Softw. Eng. Inst., CMU. p. 161 pages (November 1990)
18. Kang, K.C., et al.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5, 143–168 (1998)
19. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
20. Griss, M.L., et al.: Integrating Feature Modeling with the RSEB. In: Proc. of the 5th Int. Conf. on Software Reuse (1998)
21. Czarnecki, K., et al.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
22. Sinnema, M., Deelstra, S., Nijhuis, J., Dannenberg, R.B.: COVAMOF: A Framework for Modeling Variability in Software Product Families. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 197–213. Springer, Heidelberg (2004)

23. Benavides, D., et al.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
24. Didonet, M., et al.: Weaving Models with the Eclipse AMW plugin. In: Proceedings of Eclipse Modeling Symposium, Eclipse Summit Europe (2006)
25. Jouault, F., et al.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)

QoS-Driven Proactive Adaptation of Service Composition

Rafael Aschoff and Andrea Zisman

Department of Computing, City University London,
London, EC1V 0HB, United Kingdom
{abdy961,a.zisman}@soi.city.ac.uk

Abstract. Proactive adaptation of service composition has been recognized as a major research challenge for service-based systems. In this paper we describe an approach for proactive adaptation of service composition due to changes in service operation response time; or unavailability of operations, services, and providers. The approach is based on exponentially weighted moving average (EWMA) for modelling service operation response time. The prediction of problems and the need for adaptation consider a group of services in a composition flow, instead of isolated services. The decision of the service operations to be used to replace existing operations in a composition takes into account response time and cost values. A prototype tool has been implemented to illustrate and evaluate the approach. The paper also describes the results of a set of experiments that we have conducted to evaluate the work.

Keywords: Proactive adaptation, response time, cost, spatial correlation.

1 Introduction

A major research challenge for service-based systems is the support for service compositions that need to adapt autonomously and automatically to new situations [9][10][23][28]. Some approaches for adaptation of service compositions have been proposed in [1][2][16][29]. However, these approaches support adaptation of service compositions in a reactive way, which is time-consuming and may lead to several unwanted consequences (e.g. user and business dissatisfaction, loss of money, loss of market opportunities). Therefore, it is important to provide approaches that consider adaptation of service composition in a proactive way, predicting problems in a composition before they occur. Some initial works for proactive adaptation of service composition have been proposed in [8][15][19][30]. Overall, these few approaches are fragmented, limited, and in their initial stages.

We define *proactive adaptation of service composition* as the detection of the need for changes and implementation of changes in a composition, before reaching an execution point in the composition where a problem may occur. For example, the identification that the response time of a service operation in a composition may cause the violation of the service level agreement (SLA) for the whole composition, requiring other operations in the composition to be replaced in order to maintain the SLA; or the identification that a service provider P is unavailable requiring other

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 421–435, 2011.

services in the composition from provider P to be replaced, before reaching the execution part in the composition where services from P are invoked.

Proactive adaptation of service composition includes four main steps, namely (i) prediction of problems, (ii) analysis of the problems triggered by prediction, (iii) decision of actions to be taken due to the problems, and (iv) execution of the actions. As defined in [26], the prediction of problems is concerned with the identification of the occurrence of a problem in the near future based on an assessment of the current state of the system. More specifically, in the scope of service-based systems, problem prediction is concerned with the assessment of what is the impact of a service misbehaviour, or of a group of services, in other parts of the service composition.

In this paper we describe *ProAdapt*, a framework for proactive adaptation of service composition due to changes in service operation response time; or unavailability of operations, services, or providers. ProAdapt provides adaptation of a composition during its execution time and for future executions of the composition. The framework is based on function approximation and failure spatial correlation techniques [26]. The approach uses exponentially weighted moving average (EWMA) [22] for modelling expected service operation response time, and monitors operation requests and responses to identify the availability of services and their providers.

In ProAdapt, the need for adaptation considers a group of operations in a composition flow, instead of isolated operations, in order to avoid replacing an operation in a composition when there is a problem, and this problem can be compensated by other operations in the composition flow. The framework also identifies other operations that may be affected in a composition flow due to problems caused by a specific one. For example, when the observed response time of an operation is greater than its expected response time, the approach verifies the implication of this response time discrepancy in the service composition, instead of triggering immediate replacement of the operation. This verification considers service level agreements (SLAs) specified for the whole composition and (variable) observed values of quality aspects of the operations in the composition. When it is necessary to replace an operation, or a group of operations, the candidate operations to be used in the composition are selected based on both response time and cost constraints. This is because, in practice, there is a strong correlation between the response time and the cost for an operation.

The remainder of this paper is structured as follows. In Section 2 we present ProAdapt framework. In Section 3 we describe the proactive adaptation process for predicting and analysing problems in service composition, and for deciding and executing the adaptation actions. In Section 4 we discuss implementation and evaluation aspects of our work. In Section 5 we give an account of related work. Finally, in Section 6 we discuss concluding remarks and future work.

2 Proactive Adaptation Framework

Fig. 1 shows the overall architecture of ProAdapt framework with its main components (represented as rectangles), namely: *execution engine*, *specification translator*, *service discovery*, *monitor*, and *adaptor*. It also shows the different types

of data used as input or generated as output by the main components (represented as documents). We describe below each of these main components.

The *execution engine* receives and executes service composition specifications. We assume service composition specifications represented as BPEL4WS [5] due to its wide use and acceptance. The service composition specifications provide the flow of the application and do not have information of the exact services that need to be invoked in a composition. Instead, it contains abstract partner links information. The exact services will be instantiated by the adaptor component.

The *specification* translator is responsible to parse a service composition specification in BPEL4WS and the service level agreement (SLA) for the composition and to create a *composition model template* that will be used to generate *execution models* of the composition. An example of an execution model is described below.

The *service discovery* component identifies possible candidate service operations to be used in the composition, or to be used as replacement operations in case of problems. We assume the use of the service discovery approach [27][31] that has been developed by one of the authors of this paper to assist with the identification of candidate operations. This approach advocates a proactive selection of service operations that match functional, behavioural, quality, and contextual aspects. Details of this approach are out of the scope of this paper. The identified operations are used to create and adapt execution models by the adaptor component.

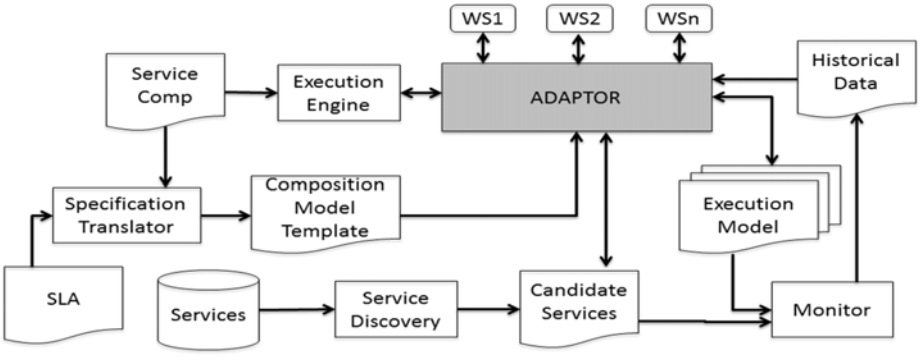


Fig. 1. ProAdapt architecture overview

The *monitor* verifies the QoS aspects of the operations used in the instantiated execution models and the replacement candidate operations, and provides historical data of these QoS aspects. The adaptor uses the historical data to predict and analyse the need for adaptation. The current implementation of the framework uses a simple monitor that we have developed that intercepts calls to the services, calculates the response time that it takes from the invocation of an operation and the receipt of its results, and accumulates the calculated response times as historical data.

The *adaptor* is the main component of our framework. It (a) receives calls from the execution engine to invoke operations in the composition and provides the results to the execution engine; (b) instantiates composition model templates and generates the execution models with real endpoint service operations to be invoked in the composition and other information; (c) predicts and analysis problems that may exist in a composition; and (d) decides on and executes actions to be taken.

Execution Model. We advocate the use of an *execution model* for each execution of a service composition. This is necessary to provide information to support prediction and analysis for adaptation, given the lack of such information in BPEL4WS [5] specifications. An execution model is a graph representation of the service composition specification with information about the (i) execution flow, (ii) deployed endpoint service operations, (iii) state of a service operation in a composition (e.g., completed, to be executed, and executing), (iv) observed QoS values of a service operation after its execution, (v) expected QoS values of a service operation, and (vii) SLA parameter values for the service operations and the composition as a whole.

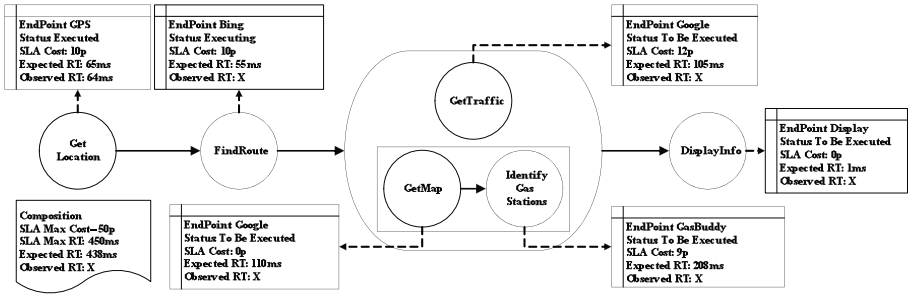


Fig. 2. Example of composition execution model

Fig. 2 shows an example of an execution model for a *RoutePlanner* service composition scenario to assist users to request information from a PDA about optimal routes to be taken when driving. As shown in the figure, the composition offers services to support the identification of a driver's current location, identification of an optional route for a certain location, display of maps of the area and route to be taken, provision of traffic information throughout the route to be taken, computation of new routes at regular intervals due to changes in traffic, and provision of information about near gas stations. In Fig. 2, for each service operation, we show its deployed endpoint, status, SLA cost values, and expected and observed response time (RT) values. We also show these values for the whole composition.

3 Proactive Adaptation Process

In ProAdapt, the adaptation process may be triggered due to (i) changes in the response times values of operations in a service composition that affects SLA values of a composition, (ii) unavailability of operations in a composition, (iii) unavailability of services, or (iv) unavailability of providers. When one of cases (i) to (iv) occurs, the adaptor verifies if the composition needs to be changed and modifies the composition during its execution time, if necessary. More specifically, the changes are performed in the execution models by trying to identify operations that can replace existing operations participating in the remaining parts of the execution model such that the aggregation of the response time and cost values of these replacement operations, together with the response time and cost values of the operations that have already been executed in the composition and the ones that are still to be executed, comply with the SLA values of response time and cost for the composition.

Response Time Modeling. In order to guarantee compliance of the SLA response time and cost values in an execution model (EM), it is necessary to consider the aggregation of the response time values of the participating operations; and the time for the adaptor to identify and analyse problems and perform changes in the execution model when necessary, as specified in the function below.

$$T(EM) = Aggreg(RT(Set(O))) + T(Adapt), \quad \text{where:} \tag{1}$$

- T(EM) is the time to complete the execution model EM;
- RT(Set(O)) is the response time of the operations in EM;
- Aggreg() is a function that returns the aggregated values of the response time of the operations depending on the execution logic of the model;
- T(Adapt) is the time required by the adaptor.

The aggregation of the response time values of the service operations in the execution model considers different execution logics in a model such as sequence, parallel, conditional selection, and repeat logics. In the example in Fig. 2, sequence execution logics are composed of operations (a) *GetLocation* and *FindRoute*, and (b) *GetMap* and *IdentifyGasStation*; while a parallel execution logic is found in operations *GetMap* and *IdentifyGasStation* with operation *GetTraffic*. In the case of sequence execution logic, the aggregated response time is calculated as the sum of the response times of the operations in the sequence; in the case of parallel execution logic, the aggregated response time is calculated as the maximum of the response time values for the operations in the parallel execution logic.

The aggregated response times of the operations in an execution model and candidate replacement operations is calculated based on *expected* response time values of the operations not yet executed and the *observed* response time values of the operations already executed. As outlined in [8], the response time for an operation request combines the time for executing the operation and the network time for sending the operation request and receiving its response. We observed that there are also other times that should be considered such as the time of marshaling/unmarshaling a request, and the time that a request may need to stay in a queue in both client and server sides. We define the response time of an operation as:

$$RT(O) = PT(O) + DT(O), \quad \text{where:} \tag{2}$$

- PT(0) is the processing time for an operation O, which is given by service providers.
- DT(0) is the variable delay time associated with an operation O, including the network, queue, and marshaling/unmarshaling times.

As shown above, the response time is considered a variable parameter that can be affected due to changes in the network and system resources. Therefore, in order to identify *expected* response time values, it is necessary to use techniques that predict the behaviour of random parameters. ProAdapt uses exponentially weighted moving average (EWMA) [22] technique for this prediction due to its simplicity.

The expected response time value of an operation changes with time. At time t0, an operation expected response time value is its processing time. At time ti (i>0), the expected response time is given by the EWMA function [22] below:

$$Ev(O(ti)) = Obv(O(ti\text{-}x))(1\text{-}\alpha) + Ev(O(ti\text{-}x))*\alpha + \beta*SD, \quad \text{where} \tag{3}$$

- Ev(O(ti)) is the expected response time value of operation O at time ti of execution;
- Obv(O(ti-x)) is the last observed response time value of O at time ti-x of execution (0<x<i; and ti-x<ti);
- Ev(O(ti-x)) is the expected response time value of O at time ti-x of execution (0<x<i; and ti-x<ti);
- α is a weight given for the past expected response time value;
- β*SD is a threshold calculated based on the standard deviation (SD) of previous observed response time values of O and β is as a constant parameter.

For each operation in an execution model, a set of candidate replacement operations is identified by the service discovery tool (see Fig. 1), based on functional and behavioural matching. The identified candidate operations are ordered by the weighted sum of the normalised response time and cost values of an operation, as per the function below. The weights are used to specify priorities in QoS values.

$$V(O) = wRT * Norm(Ev(O)) + wC * Norm(C(O)), \quad \text{where:} \tag{4}$$

- O is a candidate service operation;
- Norm(Ev(O)) is the normalised value for the expected response time of O;
- Norm(C(O)) is the normalised value for the cost of operation O;
- wRt and wC are weights used for response time and cost, with wRt+wC = 1.

Execution of Adaptation Process. For each user Ui of a service composition, an execution model EM_{Ui} is created. The initial endpoint operations used in an execution model is identified by the adaptor based on available operations, current expected response times for these operations, and SLA values of the composition.

During the execution of EM_{Ui}, the model is updated in several ways by: (a) changing the status of its operations (e.g., from "to be executed" to "executing" and "executed"), (b) calculating the observed and expected response time values of the operations as per the functions defined above, and (c) changing operations in not yet executed paths in the model, if necessary. For any of cases (i) to (iv) that may trigger the need for adaptation, the process tries to identify other parts in the execution model that may be affected by a problem. The process is based on spatial correlations of operations, services, and providers (dependencies that may exist between these elements). For example, when a service S becomes unavailable, the process considers all other operations of S in the model since these operations may not be able to be executed. Similarly, when a provider P is unavailable, the process considers all services and operations in the model from P; and when an operation O becomes unavailable, the process considers future invocations of O in the model.

We describe below the process for each case (i) to (iv). Suppose O an operation in the model being invoked by the adaptor (status "executing"); S the service associated with O; P the provider of S; Obv(O) the observed response time for O; and Ev(O) the last expected response time for O calculated using function (3).

Case (i): *Changes in the response time of O*

In this case, if the Obv(O)<=Ev(O), there is no need for adaptation and the model continues its execution. Otherwise, (Obv(O)>Ev(O)), the process verifies if the model's SLA response time is affected. This analysis is done by using function (1) above. If the SLA value is maintained, the process continues its execution. If not, the

process considers operations in the model that have not yet been executed and tries to find possible combinations of replacement operations that provide the functionality of those operations, and maintain the SLA response time and cost values. The operations in the model are considered inside the smaller possible execution logic. If a combination cannot be found, the process identifies the best possible combination.

Case (ii): *O is unavailable*
In this case, S has been changed and the adaptor receives a message from S about the unavailability of O. The process tries to identify another operation O' in the set of candidate replacement operations with the same functionality of O and acceptable expected response time and cost values. If O' exists, O is replaced by O'. The process also identifies other parts in the model not yet executed that use O and possible replacement operations for O in these parts. If these replacements operations are identified, they are use to replace O in those parts. The replacement operations are identified as in case(i) above. O is also removed from the set of candidate replacement operations.

Case (iii): *S is unavailable*
In this case, P informs the adaptor that S is not available. The process detects all other operations in the model associated with S that have not yet been executed, if any; identifies replacement operations for them from the set of candidate replacement operations; changes the execution model by replacing the operations; and removes the operations of S from the set of candidate replacement operations.

Case (iv): *P is unavailable*
In this case, the component receives a "*connection exception*" message indicating that the provider is unavailable. Similar to case (iii), the process detects all operations in the execution model associated with P that have not yet been executed; identifies replacement operations from the set of candidate replacement operations; changes the execution model by replacing the operations; and removes the operations of P from the set of candidate replacement operations.

For cases (ii) to (iv) above, if the identified replacement operations match the functionality of the operations to be replaced, but do not maintain the SLA values of the composition, the process identifies the best possible combination of replacement operations to be used. When there are no replacement operations that match the functionality, the execution model cannot be adapted and the process terminates.

In order to illustrate consider case (i) above. Suppose the observed value of *FindRoute* operation in Fig. 2 as 68ms. In this case, the response time of the execution model will be 451ms (see function 1), which is higher than the composition's SLA time value (450ms). The process tries to identify replacement operations in the following parallel execution logic (*GetTraffic* and *GetMap*, and *IdentifyGasStation* operations) that can guarantee the SLA value. If no solution is available for the operations in the parallel execution logic, the process includes operation *DisplayInfo* and considers the combined execution logics for analysis. Suppose that a replacement operation for *GetMap* is identified with expected response time value of 90 ms, which is used to compensate for the high response time of *FindRoute*.

4 Implementation Aspects and Evaluation

A prototype tool of the framework has been implemented in Java (J2SE). The execution engine and the adaptor components were implemented as a single component for simplicity. The tool assumes service compositions in BPEL4WS[5] exposed as Web Services using SOAP protocol, and participating operations and user requests emulated using soapUI[11]. The service discovery tool was also implemented in Java and is exposed as a web service using Apache Axis2. The external service registry uses eXist database [12]. In order to evaluate ProAdapt we focused on three cases described below.

Case (1): Demonstration that the framework provides time reduction when compared to non-proactive approaches;
Case (2): Demonstration that the framework manages to adapt compositions ensuring the SLA values of the composition, and considering prioritization of QoS values;
Case (3): Analysis of the performance of the framework.

The evaluation was executed in a service composition with 12 operations and different types of execution logics, as shown in Fig. 3. We assumed the same syntactic and semantic characteristics for the 12 operations, with each operation having two input parameters and producing one output result. The size of each message representing an operation request (or an operation response) was around 60 bytes. Each operation has different associated costs and processing time values, as summarised in Table 1. We assumed SLA values for cost as 2800 pence and response time as 3.5 seconds for the whole composition; and the weights for the cost and response times as 0.9 and 0.1, respectively.

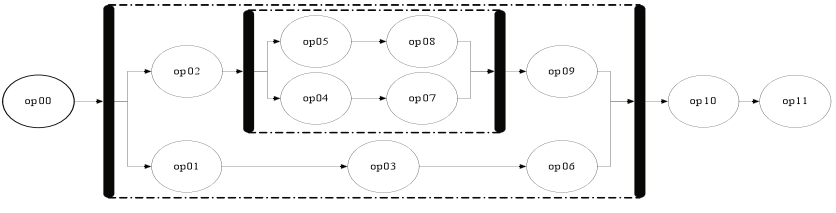


Fig. 3. Experiment service composition

We used an environment with five different machines, namely: (a) *client machine* responsible to create simultaneous requests to the service composition, simulating several concurrent users; (b) *adaptor engine machine* connected to three service providers; and (c) one machine for each *service provider* P1, P2, P3. Table 1 presents a summary of the specification of each machine and the speed of the network links between the machines. We used different speeds for the network links to emulate bottleneck situations that may occur when using an Internet environment. Each service provider contains four different services, with each service implementing three different operations in the composition in Fig. 3. The operations in the four services in provider P1 are similar in terms of their functionalities to the ones in providers P2 and P3, in order to simulate possible candidate replacement operations. We assumed different costs and processing time values for the operations in the various providers as summarised in Table 1.

We appreciate that in a real scenario the operations used in a composition may be from different providers. In the experiment, for the initial execution model we assumed all operations from the same provider (P1) to enforce a more realistic bottleneck situation. This does not invalidate our experiments since it is important to consider the network capacity between the adaptor and providers.

For the EWMA function (see Section 3), we used a threshold of 1.5 and the weight for past expected response time values of 0.6. These values were identified after executing the operations in the composition in Fig. 3 several times, and verifying that with these values the expected response times of the operations were below their observed times for 95% of the cases. We describe below how we conducted the experiments and their results for each of cases (1) to (3) above.

Table 1. Configuration of experiment environment

Machine	Configuration	Services/ Operations	Cost (pence)	Processing Time (ms)	Network Links (Mb/s)
Client (C)	Turion 1GHz 2GB RAM				C-A = 3.0
Adaptor (A)	Core 2.33 GHz 3GB RAM				
Provider P1	Pentium 4.3 GHz 1GB RAM	S0:O00, O04, O08 S1:O01, O05, O09 S2:O02, O06, O010 S3:O03, O07, O011	100	150	A-P1 = 1.0
Provider P2	Core 1.86 GHz 2GB RAM	S0':O00, O04, O08 S1':O01, O05, O09 S2':O02, O06, O010 S3':O03, O07, O011	150	100	A-P2 = 1.5
Provider P3	Pentium 3.0 GHz 3GB RAM	S0":O00, O04, O08 S1":O01, O05, O09 S2":O02,O06, O010 S3":O03,O07, O011	300	50	A-P3 = 3.0

Case (1): In this case, we compared the execution times of the composition in Fig. 3 when there is no need for adaptation with the execution times when there are problems and adaptation is required. More specifically, we analysed the times for the situations in which the need to execute adaptation is triggered by problems with (a) operations, (b) services, and (c) providers, as described in Section 3. In all these situations, we assumed the processing times of the operations in providers P2 and P3 as the same as P1. This is necessary to create a homogeneous environment and avoid using a replacement operation with faster processing time and, therefore, diminishing the impact of the time wasted when trying to invoke an unavailable operation.

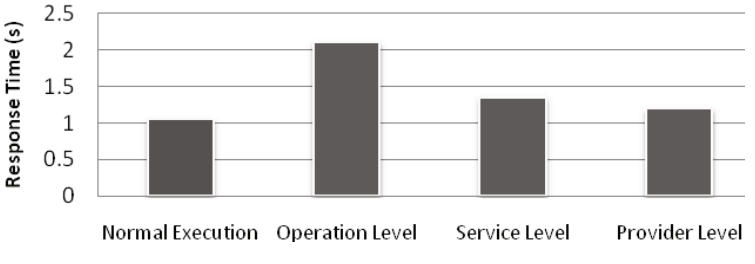


Fig. 4. Impact of spatial correlation on composition response times

For problems at the operation level (case (a)), we assumed no spatial correlation and analysed the time to execute the composition when replacement operations need to be identified for all the 12 operations in the composition. For problems at the service level (case (b)), we considered the spatial correlation between operations from the same service, and analysed the time to execute the composition when each of the four services have a problem and adaptation is required. For problems at the provider level (case (c)), we considered spatial correlation between services, and analysed the time to execute the composition when a new provider needs to be identified.

Fig. 4 presents the results of this experiment. As shown in the figure, for case (a) the time to execute the composition and change all operations without considering any spatial correlation is two times more than the time to execute the composition without any need for adaptation (normal execution). The results also show that for case (b), the time to execute the composition is improved by 36% when compared to case (a), and that for case (c) this time is improved even further by 43% when compared to case (a). We verify an improvement of using our proactive adaptation approach when compared to the situation in which a non-proactive approach is used.

Case (2): In this case, we simulated the client machine to support an increase in the number of users invoking the composition in an incremental way. More specifically, we analysed the framework in 30 intervals of ten seconds each (total of 300 seconds) with a rate of one user per second in the first interval of ten seconds, two users per second in the second interval, and an increment of one extra user every ten seconds reaching 30 users per second in the last interval. It is worth noting that for each user request performed by our client machine, 12 operations are executed and, therefore, the number of simultaneous operation executions in the experiment is greater than the number of user requests in the various intervals. Moreover, at a certain time t in the experiment, the number of users invoking the composition and consuming resources is greater than the number of new users at t, since the time to execute the composition without any problem is more than 1 second (see Fig. 4).

The above simulation was used in order to provide an environment that could on its own create problems to the composition in terms of response time and cost values due to the number of users and network resources being consumed and, therefore, allow the verification of the behaviour of ProAdapt in cases of problems.

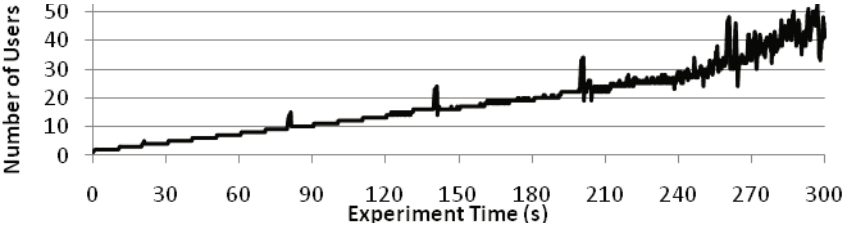


Fig. 5. Number of simultaneous users consuming resources during the experiment

Fig. 5 shows the number of concurrent users consuming resources during the different times of the experiment. As expected, the accumulated number of users is greater than the rate of new users invoking the composition. The graph in Fig. 5

shows a larger number of accumulated users towards the end of the experiment since during this time there are a larger number of invocations for the operations in the compositions causing degradation in the response times of the operations and, therefore, delaying the execution of the compositions.

Fig. 6 shows the time to execute each execution of the composition (represented as squares) during the whole experiment, and the compositions that were able to adapt themselves and finish within the SLA response time values (dotted line in the graph). As shown, the majority of the executions managed to adapt themselves and finish within the SLA response time value. The graph also shows a stable response time for the executions in the beginning of the experiment and oscillations in the response times starting at 200 seconds of the experiment. This is because between 20 and 25 service composition requests per second provider P1 reaches its full capacity, causing degradation in the operations response times, and eventually, the need to adapt the composition executions so that the SLA values are maintained.

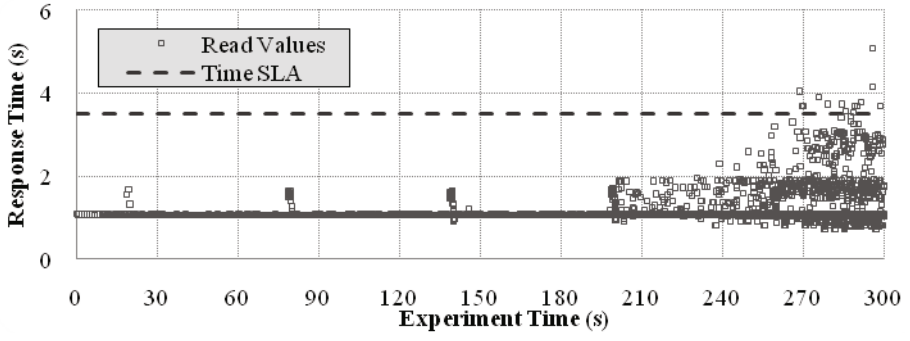
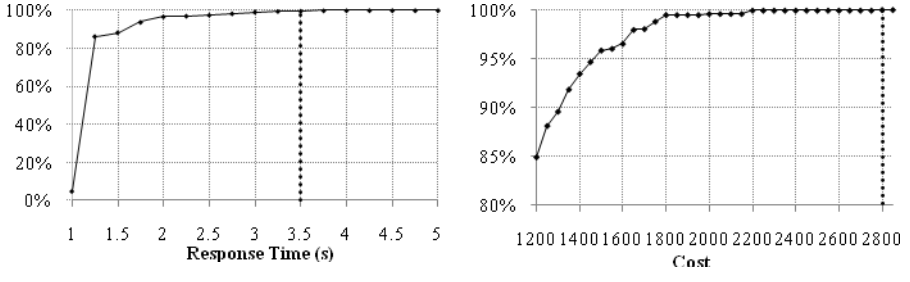


Fig. 6. Variation of composition response times during the experiment



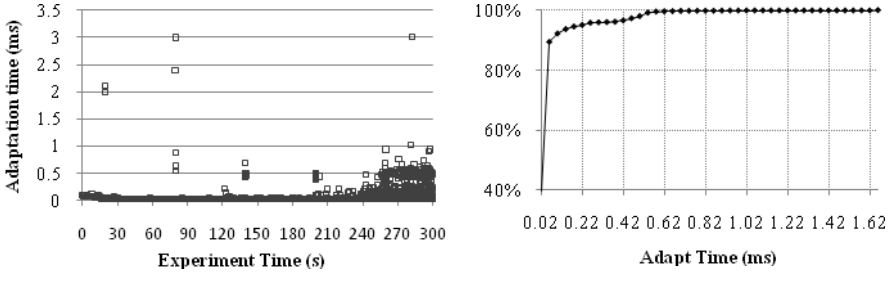
(a) (b)

Fig. 7. Cumulative frequency distribution of (a) response times and (b) costs

The cases in which the executions did not finish before the SLA value (cases above dotted line in Fig. 6), were due to bottlenecks in the providers and lack of available operations that could be executed faster and with the cost values specified in the experiment. In order to verify the impact of these cases, we present the cumulative frequency distribution for the response times of the executions in Fig. 7(a). As shown,

the response times of the executions where below 1.5 seconds for 80.65% of the cases, and in 99.69% of the cases the executions respected the SLA response time value. Therefore, from a total of 4650 user requests performed during 300 seconds of experiments, only 14 user requests could not be executed for the given SLA value.

Similarly, Fig. 7(b) presents the cumulative frequency distribution for the costs of the executions. As shown, the cheapest executions (1200 pennies) occurred in 85% of the cases. The graph also shows that the SLA cost value was respected in all 4650 requests, which was expected since the framework identifies only operations that respect the cost values when adapting the composition.



(a) (b)

Fig. 8. (a) Adaptation time and (b) cumulative adaptation time over the experiment

Case (3): In this case, we analysed the time spent by ProAdapt to adapt the compositions. Fig. 8(a) shows the times of the adaptor component in milliseconds for all executions in our experiment, while Fig. 8(b) shows the cumulative frequency distribution. As shown, in the majority of the cases the overall adaptation time is very small and does not cause a significant increase on the overall response time of the composition execution. The experiment shows that only in few cases the time for the adaption process was 3 milliseconds at most, which is very low when compared with the time to execute the composition when there is no need for adaptation. This is due to the way the process is implemented in which compositions are analysed based on execution logics and without looking for the optimal combination of operations, but for combinations that meet given SLA values.

Overall, the results of our experiments are very positive and demonstrate that the framework can support proactive adaptation of service composition during execution time, due to different QoS characteristics. The experiments also show that the performance of the adaptation process is good and that the process does not cause penalties when changes in the composition are necessary.

5 Related Work

Some approaches have focussed on dynamic service composition, in which services are identified and aggregated during runtime in support of certain functional and quality characteristics of the desired systems [1][3][6][7][13][24][25].

Approaches for reactive adaptation of service composition were proposed in [1][2][16][18][29]. These approaches propose changes in service composition based on pre-defined policies [2], self-healing of compositions based on detection of exceptions and repair using handlers [29], context-based adaptation of compositions using negotiation and repair actions [1]; and key performance indicator (KPI) analysis and the use of adaptation strategies related to the KPI fulfilment [16].

Exceptions to the reactive approaches are found in the works in [8][14][15][17][19][21][28]. As in the case of ProAdapt, the work in [8] is based on prediction of performance failures to support self-healing of compositions. The work uses semi-Markov models for performance predictions, service reliability model, and minimization in the number of service re-selection in case of changes. The decision to adapt is based on the performance of a single service, while our framework considers a group of related service operations in a composition, avoiding unnecessary changes to the composition. Moreover, the work in [8] does not support unavailability and malfunctioning of operations, services, and providers, as well as spatial correlations between these elements in a composition.

In [17] the PREvent approach is described to support prediction and prevention of SLA violations in service compositions based on event monitoring and machine learning techniques. The prediction of violations is calculated only at defined checkpoints in a composition based on regression classifiers prediction models.

The works in [14][19][28] advocate the use of testing to anticipate problems in service compositions and trigger adaptation requests. The approach in [28] supports identification of nine types of mismatches between services to be used in a composition and their requests based on pre-defined test cases. In [14][19] test cases are created during the deployment of service compositions and used to identify violations after a service is invoked for the first time. However, the creation of test cases is not an easy task and the work does not specify how to generate new test cases for a modified composition.

In [15] the authors describe a two-stage adaptation approach due to dependability requirements of service-oriented systems. The work combines proactive adaptation to support self-protection of the system and reactive adaptation to support self-healing of the system. The paper does not describe the advantages of combining both approaches and lacks details of the proactive approach.

Similar to our framework, some works have been proposed to support prediction of response time on the web [21][30]. In [21] the authors describe a probability function for web-access response time that uses file-size cumulative function and delay probability density function. The work in [30] presents a framework for performance evaluation of web services based on queuing networks and fork-and-join. As in the case of our framework, this work considers the execution of a service and, therefore its response time, from the moment a service request leaves a client machine to the moment service results return to the client. Our framework complements the above works and applies prediction of operation response time.

Similar to our approach, in [4] the authors advocate that the management of service compositions during runtime needs to consider the structure of a composition and the dependencies between the participating services, and propose an approach that determines the impact of each service in a composition on its overall performance. The reactive region-based reconfiguration approach presented in [18] also has similarities with our work since it considers services that are in certain regions in a

composition. However, in [18] the reconfiguration approach is used only for future executions of the composition, instead of current executions as in our work.

Our framework complements existing works for service composition adaptation and contributes to the challenge of supporting adaptation in a proactive way during execution time, taking into consideration SLA values for the whole composition.

6 Conclusion and Final Remarks

In this paper we presented ProAdapt, a framework for proactive adaptation of service composition due to changes in service operation response times; or unavailability of operations, services and providers. The framework uses function approximation and failure spatial correlation of operations, services, and providers to predict problems. It also uses exponentially weighted moving average (EWMA) to model response times of operations. The adaptation process is performed during the execution of a composition and considers a group of operations in a composition to verify if a problem can be compensated by other operations in a composition flow. Replacement operations are selected based on their response times and cost values. A prototype implementation of the framework has been developed and used to evaluate the framework with positive results

Currently, we are extending the framework to support proactive adaptation due to other types of QoS aspects and other circumstances. Examples of these circumstances are availability of new (better) service operations than the ones used in a composition, and changes in requirements or emergence of new requirements for the system. We are also investigating other ways of adapting compositions including changes in the structure of the composition's workflow (e.g., replacement of one service by a group of services, or vice-versa). We are expanding our prototype tool to support analysis of the impact of a problem in conditional and repetition execution logics in service compositions, and integrating the adaptor component with our proactive service discovery framework for identification of candidate replacement operations.

References

1. Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software 24(6) (2007)
2. Baresi, L., Di Nitto, E., Ghezzi, C., Guinea, S.: A Framework for the Deployment of Adaptable Web Service Compositions. Service Oriented Computing and Applications Journal (to appear)
3. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for QoS-aware Web Service Composition. In: IEEE International Conference on Web Services (2006)
4. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.C.: Analyzing Impact Factors on Composite Services. In: IEEE Int. Conf. on Services Computing (September 2009)
5. BPEL4WS, `http://www.download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf`
6. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-Aware Replanning of Composite Web Services. In: IEEE Int. Conf. on Web Services (2005)
7. Colombo, M., Di Nitto, E., Muri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In: Proc. of the 4th Int. Conf. on Service Oriented Computing (2006)
8. Dai, Y., Yang, L., Zhang, B.: QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction. Journal of Computer Science and Technology 24(2) (March 2009)

9. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A Journey to Highly Dynamic, Self-Adaptive, Service-based Applications. Automated Software Engineering Journal 15, 313–341 (2008)
10. Dustdar, S., Papazoglou, M.P.: Services and Service Composition – An Introduction. IT Information Technology 2, 86–92 (2008)
11. Eviware. soapUI; the Web Services Testing tool, http://www.soapui.org
12. eXist, http://exist.sourceforge.net
13. Fujii, K., Suda, T.: Semantics-based Dynamic Web Service Composition. Int. Journal of Cooperative Inf. Systems 15(3), 293–324 (2006)
14. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A Framework for Proactive Self-Adaptation of Service-based Applications Based on Online Testing. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 122–133. Springer, Heidelberg (2008)
15. Jun, N., Bin, Z., Xiamgyu, Z., Zhiliang, Z., Dancheng, L.: Two-Stage Adaptation for Dependable Service-Oriented System. In: International Conference on Service Sciences (2010)
16. Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., Leymann, F.: Adaptation of Service-based Applications Based on Process Quality Factor Analysis. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 395–404. Springer, Heidelberg (2010)
17. Leitner, P., Michlmayr, A., Rosenber, F., Dustdar, S.: Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In: Int. Conf. on Web Services (2010)
18. Lin, K.J., Zhang, J., Zhai, Y., Xu, B.: The Design and Implementation of Service Process Reconfiguration with End-to-end QoS Constraints in SOA. Journal of Service Oriented Computing and Applications 4 (2010)
19. Metzer, A., Sammodi, O., Pohl, K., Rzepka, M.: Towards Pro-active Adaptation with Confidence Augumenting Service Monitoring with Online Testing. In: Software Engineering for Adaptive and Self-managing Systems, SEMAS, South Africa (May 2010)
20. Mitchell, T.M.: Machine Learning. McGraw-Hill International Editions (1997)
21. Miyagi, M., Ohkubo, K., Kataoka, M., Yoshizawa, S.: Performance Prediction Method for Web-Access response Time Distribution Using Formula. In: Network Operations and Management Symposium (2004)
22. NIST/SEMATECH eHandbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook
23. Papazoglou, M.P., Traverso, P., Dustdar, S., Leyman, F., Kramer, B.: Service-Oriented Computing Research Roadmap, http://tinyurl.com/6jhvd44
24. Pernici, B. (ed.): MAIS Project. Mobile Information Systems – Infrastructure and Design for Flexibility and Adaptability. Springer, Heidelberg (2006)
25. Pistore, M., Marconi, A., Bertolini, P., Traverso, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: Int'l Joint Conf. Artificial Intelligence (2005)
26. Salfner, F., Lenk, M., Malek, M.: A Survey of Online Failure Prediction Methods. ACM Computing Surveys 42(3) (2010)
27. Spanoudakis, G., Zisman, A.: Discovering Services during Service-based System Design using UML. IEEE Transactions of Software Engineering 36(3), 371–389 (2010)
28. Tosi, D., Denaro, G., Pezzè, M.: Towards Autonomic Service-Oriented Applications. International. Journal of Autonomic Computing (IJAC), 58–80 (2009)
29. WSDiamond, http://wsdiamond.di.unito.it/status.html
30. Youcef, S., Bhatti, M.U., Mokdad, L., Monfort, V.: Simulation-based Response-time Analysis of Composite Web Services. In: 10th IEEE International Multitopic Conference
31. Zisman, A., Dooley, J., Spanoudakis, G.: A Framework for Dynamic Service Discovery. In: Int. Conf. on Automated Software Engineering, Italy (2008)

A Quality Aggregation Model for Service-Oriented Software Product Lines Based on Variability and Composition Patterns

Bardia Mohabbati[1], Dragan Gašević[1,2], Marek Hatala[1], Mohsen Asadi[1], Ebrahim Bagheri[2,3], and Marko Bošković[1,2]

[1] Simon Fraser University, Canada
{mohabbati,mhatala,masadi}@sfu.ca
[2] Athabasca University, Canada
{dragang,ebagheri,marko.boskovic}@athabascau.ca
[3] University of British Columbia, Canada

Abstract. Quality evaluation is a challenging task in monolithic software systems. It is even more complex when it comes to Service-Oriented Software Product Lines (SOSPL), as it needs to analyze the attributes of a *family* of SOA systems. In SOSPL, variability can be planned and managed at the architectural level to develop a software product with the same set of functionalities but different degrees of non-functional quality attribute satisfaction. Therefore, architectural quality evaluation becomes crucial due to the fact that it allows for the examination of whether or not the final product satisfies and guarantees all the ranges of quality requirements within the envisioned scope. This paper addresses the open research problem of aggregating QoS attribute ranges with respect to architectural variability. Previous solutions for quality aggregation do not consider architectural variability for composite services. Our approach introduces variability patterns that can possibly occur at the architectural level of an SOSPL. We propose an aggregation model for QoS computation which takes both variability and composition patterns into account.

Keywords: Software Product Line (SPL), Service-Oriented Architecture (SOA), non-functional properties, QoS aggregation, process family, service variability, variability management, feature modeling.

1 Introduction

The Service-Oriented Architecture (SOA) paradigm enables the realization of Software-as-a-Service (SaaS). Some service providers have already moved towards the adoption of customizable product-development models to efficiently tailor solutions for their stakeholders. Within this process, they need to consider, manage and withstand both variable functional and non-functional (quality) requirements to produce new applications systematically [1]. *Software Product Line Engineering (SPLE)* provides a platform to capture both functional and non-functional aspects and allows for the rapid customization of new products. Several researchers have proposed to integrate SOA and SPLE paradigms into *Service-Oriented Software Product Lines (SOSPLs)* as a way

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 436–451, 2011.

to formalize *customizable product-development* and take the benefits and synergies of both paradigms [1,2].

Researchers have explored various strategies for the realization of software applications, e.g., how the most appropriate services can be selected for a given product line and how they can be efficiently composed. However, previous works often fails to consider Quality-of-Service (QoS) in the context of product lines. Quality evaluation is a challenging task in monolithic software systems and it is even more complex when it comes to SOSPL, as it needs to analyze the attributes of a family of SOA systems.

This paper contributes a solution to the following open research problem: *How can the quality attributes of a software product line be aggregated with respect to architectural variability?* The novelty of our approach is in accounting for variability during architecture quality aggregation, which has not been considered in any related work, to the best of our knowledge. Our work focuses on the development of a framework for computing the quality ranges of features in an SOSPL by aggregating QoS properties at the architectural level. Building on our previous work [3,4], we assume that QoS dimensions are captured in terms of quantitative properties. In particular, this paper makes the following contributions:

1. The introduction and classification of a set of possible *variability patterns* that occur at the architectural level of an SOSPL. This can be seen as a catalog of patterns for variability modeling;
2. The development of a *quality model framework* for the aggregation of QoS based on different architectural patterns;
3. The formalization of a *computational model* for architectural quality evaluation, which takes into account both variability and composition patterns and allows for tradeoff analysis and architectural decision making between options that provide similar functional properties but different levels of quality.

The reminder of this paper is organized as follow: Section 2 describes SOSPL and its related conceptual modeling and formalism. QoS aggregation and computation model for SOSPL architecture is described in Section 3. The discussion and complexity evaluation of methods is presented in Section 4. The related work is discussed in Section 5. Finally, Section 6 presents the conclusion and future work.

2 Service-Oriented Software Product Lines

SPLE has been recognized as a successful approach to variability management and reuse engineering, which enables mass customization, enhances software quality and reduces the time-to-market of new software products [5]. Different software products derived from a software product line are distinguishable based on their included features. A *feature* reflects the stakeholders' requirements. It is an increment in the product functionality and offers a configuration option [5]. Given this definition for a feature, SPLE relies on the essential concepts of *commonality* and *variability* of features among products.

SPLE consists of two main lifecycles: *Domain Engineering* and *Application Engineering* [5]. Domain Engineering is concerned with the analysis and identification of

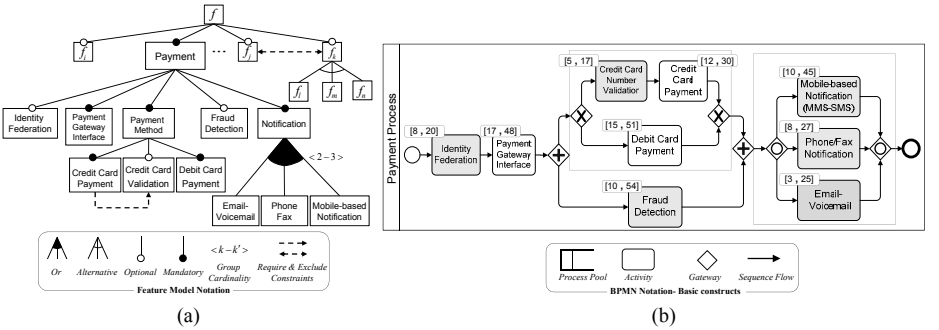
the scope of the product line and the capturing of the entire domain of interest through modeling of common and variation points. An Application Engineering cycle builds the understanding of specific requirements of different stakeholders, for whom the customization and configuration of the product line is carried out. We have presented the details of these two distinctive lifecycles in [3].

Given that software product line models are often abstract representations of a domain/application of interest, it is important that they are interrelated with solution space models that would allow their actual operationalization. To this end, many researchers and practitioners have already investigated the importance of leveraging the synergies between SOA and SPL to create Service-Oriented Software Product Lines (SOSPLs) [1,2]. Such approaches benefit from SOA principles to provide an actual implementation of SPL products. Let us review an illustrative example in this regard.

2.1 Illustrative Example

In Fig.1, we show a simplified business process model for e-Payment in a global online retailer scenario to illustrate the concepts and further discussions. As shown, different features from the feature model (on the left) that can be used within the business process model (on the right) to implement the functionality of the payment process. These features can be realized using appropriate services, which can have different QoS characteristics. For example, the activities represented in gray color in the process model indicate optional features; i.e., those features can be optionally included or excluded from the target product based on stakeholders' requirements. For instance, the `Notification` feature can be included in a product by selecting one of the `Mobile-based notification`, `Phone-Fax notification`, or `Email/Voicemail notification` features, which have different range of quality values, since they are implemented by different services. Hence, the QoS characteristics of a developed product are closely dependent upon the features that get included in a final product.

A feature model such as the one in Fig.1(a) is a model of a family of products (SPL), while each variant (customized service) is a member of that family. Variation points are



(a) (b)

Fig. 1. a) Feature model representing structural variability in business process family b) E-payment feature and its process flow

those places in the design of the SOSPL architecture where a specific decision has been narrowed down to several options. However, the options to be selected for a particular application w.r.t. stakeholder's requirements are left open for configuration. Hence, the variation points provide the possibility to derive different products, i.e., different final composite services. In SOSPL, particularly during the domain engineering cycle, determining the implied QoS ranges for individual features, based on the underlying architecture and implementation, helps domain engineers to ensure that the product line architecture will fulfill and deliver the upper and lower bounds or values of quality requirements requested by stakeholders. In other words, the aggregation of the QoS properties of a feature model based on the QoS characteristics of its features, as derived from underlying processes and services implementing those features, provides the means to estimate the likely lower and upper bounds of QoS properties for potential products that will be derived from that product family. Furthermore, in the context of SOSPL, quality range computation through the construction of a generic evaluation model enables us to keep track of the product line quality ranges even during or after specifications of the service quality have changed. Therefore, the main contribution of this work is an introduction of the process how these QoS ranges are computed in the presence of variability.

2.2 Feature Modeling

Feature modeling is one of the important techniques for capturing, modeling and describing the commonalities and differences between the products of a family based on their features. A feature model is a means for describing a permissible configuration space of all the products of a family in terms of its features and their relationships. A feature model consists of both formal semantics and graphical representation, which is a rooted directed acyclic graph (DAG). Fig.1(a) depicts a part of the feature model in our example, which describes common and variable features and represents architectural variability in the reference business process model. Parent-child relationships in the feature diagram indicate the refinement of application functionality. As not all features are assumed to be present in every product, this differentiation is expressed by a classification of feature types and relationships, which drive *architectural variability patterns* as follows:

1. **Mandatory-Optional:** A mandatory feature must be included in every member of a product line if its parent feature is selected. An optional feature may or may not be included if their parent is included;
2. **Or groups:** Or feature groups are non-empty subsets of features that can be included if a parent feature is included;
3. **Alternative groups:** Alternative feature groups indicate that from a set of alternative features exactly one feature must be included if the parent of that set is included.

We formally define a feature model as follows:

Definition 1 (Feature Model). *A feature model* $FM{=}G_{FM}(V, E)$ *is a directed acyclic graph consisting of the set of vertices* V *representing features and edges* $E \subseteq V{\times}V$ *representing the parent-child relations between the features, such that* $E{=}\{\overset{\bullet}{f}, \overset{\circ}{f}, f_{or}, f_{xor}\}$,

where $\overset{\bullet}{f}$ *and* $\overset{\circ}{f}$ *denote mandatory and optional parent-child feature relations, respectively;* f_{or} *and* f_{xor} *denote Or and Alternative group relations between parent-child features with common parents, respectively.*

In general, *cardinality* is also defined among a group of $n > 1$ sub-features, which is denoted by $<k-k'>$, where $1 \leq k \leq k' \leq n$. Hence, $<k-k'>$ cardinality defined over *Or feature groups* indicates at least k and at most k' features can be included out of the n features in a group if the parent is selected. In addition, *integrity constraints*, i.e., the *includes and excludes* relations, can be defined over features of a feature model. They are the means to express that the presence of a certain feature in the product imposes the presence or exclusion of another feature.

As it can be observed, except for the mandatory feature type, all the other types of features imply architectural variability.

2.3 Reference Business Process Model

A template-based approach, where a reference model is designed as a template and is further customized for various purposes, has been widely adopted by practitioners [6]. The reference model contains a union of the business processes for the entire product line in a superimposed way. The reference model provides the common business logic for orchestration and choreography of services. The design of reference models is accomplished in the course of the domain engineering lifecycle [5].

The configuration (tailoring and customization) of a reference models is performed by the selection/elimination of features from the feature model. In other words, due to the fact that architectural variations in the reference model are encoded as features, the various parts of the reference business processes are organized in variation points, which are managed and configured by means of feature models. It should be noted that we distinguish between design and runtime variability. Feature models capture and encapsulate only architectural variability at design time. In contrast, business process models describe behavioral variability, i.e., how features are composed, which drives runtime variability through composition patterns (discussed in the next section).

We consider a business process model as a workflow which is formally defined as follows:

Definition 2 (Business Process Model). *Business process model BP is defined as a directed acyclic graph* $G_{BP} = (V, E)$*, where* $V = \{n_\epsilon, V_\sigma, V_A, V_G\}$ *denotes a set of disjoint nodes,* n_ϵ *is a unique initial state,*V_σ *is a final state,* V_A *is a set of activities, and* E *represents the edges (transitions) between the nodes.* V_G *is a set of nodes as gateways.*

A business process is viewed as a series of activities where an activity represents a functional abstraction of services. An activity can be 1) atomic (a.k.a., task) or 2) non-atomic (a.k.a., sub-process). Each activity is delegated and bound to one or more services that provide the required functionality with different quality properties. Gateways, as routing constructs, represent a control flow of branchings, i.e., routing points. In this paper, we impose the following well-formedness conditions on a business process structure [7]: i) a business process model has a single source node, i.e., a node with no incoming

edge; ii) every activity node has a single incoming and a single outgoing edge; iii) for every node with multiple outgoing arcs (i.e., a split), there is a corresponding node with multiple incoming arcs (a join), such that the set of nodes between the split and the join forms a single-entry-single-exit region [8].

In our work, architectural and behavioral variability are described by means of two models, i.e., feature models and business process models, respectively. Hence, we will assume that there is a mapping model available which interconnects these two models [3,6]. This injective mapping (i.e., one-to-one) reciprocally links each feature in the feature model to the corresponding activity in the reference business process model.

3 Quality of Service Aggregation and Computation for Product Line Architecture

In this section, we describe our proposed quality aggregation model for product line architectures. We will cover the following issues in order to provide a model for aggregating and computing QoS range values in the presence of variability: 1) quality criteria and quality range values for SOSPL; 2) the combination of variability and composition patterns; 3) based on the combination of variability and composition patterns; and 4) computational algorithms for computing aggregate quality range values.

3.1 Quality Criteria for Service-Oriented Product Line

Different fields of research and standards have proposed diverse definitions and ontologies for describing QoS properties based on their target application domains [9].

We consider some quantitative QoS characteristics of Web services, which have been taken into consideration as selection criteria in the research literature [10,9,11,12]. Specifically, *cost* and *response time* will be the two indexed QoS properties included in our work, which will be denoted by q_{pr} and q_{rt}, respectively, throughout the paper. Of course, our approach is not limited to these two QoS types, but these are the only ones discussed here due the limited space of the paper.

Let us now proceed with some formal definitions as a basis for our work.

Definition 3 (Quality Range). *The quality range values of the i^{th} quality property (dimension) is defined as $q_i^{\mathrm{R}} = [q_i^{\mathrm{LB}}, q_i^{\mathrm{UB}}]$, where q_i^{LB} and q_i^{UB} are lower and upper bound values of the quality property, respectively.*

The above definition shows that each property such as response time or cost can be described by a range of numerical values. This range specifies both lower and upper bounds for that quality property. In order to be able to compute such a quality range, appropriate aggregation operators are needed. We consider the following three types of quality aggregation operators for computing the quality ranges of a software product line:

- ***Summation:*** The quality range values of the product line is determined by a sum of the QoS range values of the quality attributes of services. An example would be cost;

- ***Multiplication:*** The range values of quality attributes are determined by production of the QoS values of the services, for instance, reliability and availability;
- ***Min-Max:*** The quality range values of the product line are computed with respect to *critical paths* [9,13] in the business process structure, for instance, response time (i.e., execution duration).

In order to employ the above operators, we consider the following. We assume that for each activity $a_n \in V_A$ in a business process model BP, there is a bounded set of candidate services, $S_{a_n} = \langle s_{n1}, \ldots, s_{nm} \rangle$, in which all of the candidates provide the same functionality, but with different degrees of QoS properties. The quality of a service s is a vector $Q_s = \langle q_1(s), \ldots, q_k(s) \rangle \in \mathbb{R}$, where the function $q_i(s)$ determines the values of the i^{th} quality property.

The quality of each activity a_n is defined as a matrix $[Q_{a_n}]_{i \times j}; 1 \leq i \leq k, 1 \leq j \leq m$, where each row corresponds to a quality property q_i, while each column corresponds to a service candidate. Thereby, the range of the i^{th} quality property for feature f_n corresponding to activity a_n is obtained by the quality range function $q_i^{\mathrm{R}}(f_n) = [q_i^{\mathrm{LB}}, q_i^{\mathrm{UB}}]$, where $q_i^{\mathrm{LB}} = Q_{an}^{\min}$ and $q_i^{\mathrm{UB}} = Q_{an}^{\max}$.

For example, let us assume that there are five service candidates in S_{a_i} binding to activity a_k, mapped to feature f_k, and that their service cost values (q_{pr}) are given by vector $Q_{a_k}(i,j) = \langle 100, 250, 65, 130, 95 \rangle$. The cost range values of feature f_k could be set as follows: $q_{pr}^{\mathrm{R}}(f_k) = [65, 250]$, because we are interested in the lower and upper bound values for the quality range.

3.2 Combining Variability and Composition Patterns

In essence, the aggregation model for quality computation in the context of SOSPL depends on: a) structural variability captured by a feature model; and b) behavioral variability captured by a business process model, which describes the composition structure.

Composition patterns[1], which have their roots in workflow management systems[14], aim at building composition structures that are derived from the requirements in the process modeling phase.

In other words, these patterns describe the behavior of features during execution time. They represent the abstract control flow and execution sequence of features within the reference business process model for the whole family. Similar to [12], we consider composition patterns that address the behavioral structure of a composition. These patterns can be grouped into two main groups: a) *sequential patterns* and b) *parallel patterns*. These patterns are defined in terms of how the process flow proceeds in sequences and splits into branches for executing the activities and how they merge or converge. For our work, we consider the combination of parallel split, convergence and synchronization patterns.

Fig.2 illustrates three variability patterns (left side), as described in Sec.2.2, in combination with nine composition patterns (CP1-CP9) represented using BPMN notation (right side). We perform quality aggregation based on a set of proposed aggregation rules described below.

[1] We use the terms *workflow* and *composition patterns*, interchangeably.

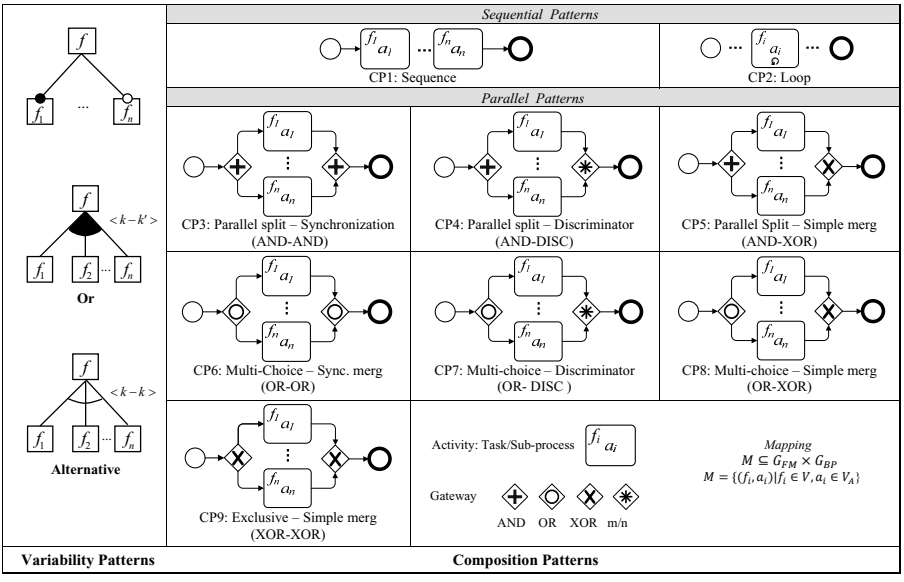


Fig. 2. Variability and composition patterns

3.3 Aggregation Rules Based on Variability and Composition Patterns

Based on the patterns described above, we define aggregation rules for each QoS property by primarily taking into account the variability patterns which may occur within each composition pattern. In the following, we present the aggregation rules for two numerical QoS properties: cost and response time (i.e., execution time). The cost of a feature is the cost which can be associated with the deployment, execution, management, maintenance and monitoring of a service. The aggregation rules for availability and throughput are presented in a longer version of the current paper which is accessible online[2]. The summary of the aggregation rules that we have defined are given in Tables 1 and 2.

According to Definition 3, the definition of a lower bound, q_i^{LB}, for different quality properties must indispensably consider the mandatory features for sequential and parallel split patterns (CP1-CP4). In addition to mandatory features, the optional features generally contribute to the upper bound range value, q_i^{UB}. For instance, in the sequential patterns, the cost of feature f should be determined by the sum of the cost values of each mandatory feature for the lower bound; while the upper bound is determined by the accumulated cost for mandatory as well as optional features.

By adopting a hierarchical approach, described in the next section, the range values (upper and lower bounds) for QoS properties are computed for a combination of variability and composition patterns, based on our formulated aggregation rules. To determine the upper and lower bounds for QoS range values, $q_i^{\mathrm{R}}(f)$, for a parent feature

[2] `http://qos-sospl.sourceforge.net/`

Table 1. Aggregation rules based on *Mandatory-Optional* variability patterns

Mandatory-Optional Variability Pattern		QoS Properties		Cost (q_c)	Response Time (q_{rt})
Seq. Patterns	1	Sequence	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$
	2	Arbitrary Cycle	$\left[cq_{pr}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f} \;,\; cq_{pr}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right]$	$\left[cq_{rt}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f} \;,\; cq_{rt}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right]$
Parallel Patterns	3	AND-AND	$\left[\sum_{i=1}^{n} q_{pr}^{LB}(f_i):\forall f_i \in \dot{f} \;,\; \sum_{i=1}^{n} q_{pr}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right]$	$\left[\max\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	4	AND-DISC		$\left[\min\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	5	AND-XOR		
	6	XOR-XOR	$\left[\min\left(q_{pr}^{LB}(f_i): f_i \in \dot{f}\vee\hat{f}\right) \;,\; \max\left(q_{pr}^{UB}(f_i): f_i \in \dot{f}\vee\hat{f}\right)\right]$	$\left[\min\left(q_{rt}^{LB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right) \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	7	OR-XOR		
	8	OR-OR	$\left[\min\left(\sum_{f_i \in F_{Sub}} q_{pr}^{LB}(f_i):\forall F_{Sub} \in F_{C_m^n}\right) \;,\; \max\left(\sum_{f_i \in F_{Sub}} q_{pr}^{UB}(f_i):\forall F_{Sub} \in F_{C_m^n}\right)\right]$	$\left[\min\left(\max\left(q_{pr}^{LB}(f_i): f_i \in F_{Sub}, \forall F_{Sub} \in F_{C_m^n}\right)\right)^{1} \;,\; \max\left(q_{rt}^{UB}(f_i):\forall f_i \in \dot{f}\vee\hat{f}\right)\right]$
	9	OR-DISC		

[1]For example, assuming three features (m =3) out of seven (n=7) in CP6-CP7, the minimum of 3rd quickest should be considered

f (see Fig.2), the aggregation rules are applied on the basis of each composition pattern according to the variability patterns.

To further introduce the principles of our aggregation model, the following explanation is provided. Feature set $F_{C_l^n}$ contains all of the permissible combinations of the feature sets, where the number of distinct l-element subsets is a binomial coefficient denoted as C_l^n. To compute the quality range values, the aggregation operators are applied to each member set of $F_{Sub} \in F_{C_l^n}$. For instance, for the lower and upper range values of cost (q_{pr}), the summation operator is first applied to each element of F_{Sub}, which results in a new set. For this new set, the min-max operator is then applied.

The mandatory and optional features in the Multi-choice parallel patterns (cf. CP6, CP7 and CP8) follow different aggregation rules. To determine the lower and upper bounds, the aggregation model also requires knowing which paths in the business process flow will be chosen at runtime particularly for OR-Splits. We assume that an execution of all possible choices for an OR-Split gateway is equally probable. The business rules defined over business process models (i.e., OR-Split gateway) specify how many paths (m) can be executed at runtime. This results in a feature set $F_{Sub} \in F_{C_m^n}$ where $k \leq m \leq k' \leq n$.

For instance, in our example, assume that two notification features `Email-voicemail` and `Mobile-based notification` are included in an instance of the reference business process. Hence, the decision concerning which notification service should be invoked w.r.t. OR-split semantics is left to runtime.

To address *Or group* and *Alternative group* feature variability in combination with Multi and Exclusive choice composition patterns (CP7, CP8, and CP9), the aggregation model must consider all possible combinations of features corresponding to the given cardinality $<k-k'>$ over the n features of the feature group specified at design time. Therefore, the resulting feature set (i.e., F_{Sub}) is a subset of $F_{C_k^{k'}}$ and $F_{C_{k'}^n}$ for lower and upper bound quality range values, respectively.

3.4 Quality of Service Range Aggregation

The QoS range values for features in a feature model are computed by hierarchically aggregating the QoS for variability patterns at the level of each composition pattern. Aggregation is performed by gradually collapsing features into a single feature in the

Table 2. Aggregation rules based on *Or* and *Alternative* variability patterns

Or[*] / *Alternative*[**] *Variability Patterns*			QoS Properties	Cost (q_c)	Response Time (q_{rt})
	Seq. Pattern	1	Sequence	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{pr}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{pr}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$
	Parallel Patterns	3	AND-AND	$\left[\min\left(\sum_{f_i \in F_{Stub}} q_{pr}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right), \max\left(\sum_{f_i \in F_{Stub}} q_{pr}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right]$	$\left[\min\left(\max\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		4	AND-DISC		$\left[\min\left(\min\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		5	AND-XOR		
		6	OR-XOR		
		7	OR-OR		
		8	OR-DISC		$\left[\min\left(\max\left(q_{rt}^{LB}(f_i) : \forall F_{Stub} \in F_{C_k^n}\right)\right), \max\left(\max\left(q_{rt}^{UB}(f_i) : \forall F_{Stub} \in F_{C_{k'}^n}^{(*)} \middle| F_{C_k^n}^{(**)}\right)\right)\right]$
		9	XOR-XOR		

[1](*) and (**) represent the feature set combinatorial operators which are applied for *Or* and *Alternative* variability patterns, respectively

feature model, by employing the notion of a *virtual feature*. This approach enables us to perform the aggregation from both micro and macro perspectives. In other words, the quality range values can be computed for each of the given features from a local view and also for the entire feature model from a global view.

Algorithm 1. Aggregate QoS range for feature: AggregateQoSRange(f)

Input: f: given feature of feature model FM
Output: QoS reange- $q^{\mathrm{R}} offeature f$

```
1  begin
       // All direct child features of f;
2      S_f[ ] ⟵ ∀ChildFeatureOf(f) ;
3      if S_f = ∅ then return q^R(f);
4      else
5          for i = 1 to |S_f| do
6              AggregateQoSRange(S_f[i]);
7              PST ⟵ ProcessStructure(S_f[i]);
8              if PST = ∅ then return q^R(f) else q^R(f)= ComputeQoSR(PST)
9
10 end
```

Algorithms 1 and 2 detail the procedure for computing QoS ranges for a feature model. In these algorithms, the feature model is traversed from a given feature node by post-order depth-first traversal, i.e., computing the aggregated QoS ranges from leaves and the right-most nodes up to the root node. For each feature, the business Process Structure Tree (PST) corresponding to a given feature is subsequently created and parsed (Fig.3(b)). For every trivial Single-Entry Single-Exit (SESE) component in PST, the control flow analysis is performed, whose details are omitted from Alg. 2 for the sake of brevity; interested readers are referred to [15].

In order to analyze and identify how features at the same level in the feature model are composed, we decompose the process graph into process components (Fig. 3). A process component is a subgraph of the composition model with a SESE region, which may include individual tasks but also larger subgraphs. We employ the Refined Process Structure Tree (RPST)-based approach proposed in [13,8] to create and parse the process model into a tree of SESE components (line 7 in Alg. 1).

Alg. 2 operates over a PST, and the aggregation is performed at the level of each processes component. Lines 3 to 18 iteratively aggregate the quality for each child component. For individual process components, features are grouped into virtual features corresponding to the variability patterns.

Algorithm 2. Compute QoS range values of business process associated to feature f: ComputeQoSR(C)

Input: C : Process component- node of process structure tree PST
Output: q^{R}: aggregated QoS range of process component

1 **begin**
2 **foreach** $C_i \in ChildOf(C)$ **do** ComputeQoSR(C_i)
3 **forall** $f_k \in C_i$ **do**
`// [⋈]Group features w.r.t. variability patterns and step-wise collapsing features by means of virtual features;`
4 **switch** *feature* f_k.*Type* **do**
5 **case** $f_k \in \overset{\circ}{f} \vee \overset{\bullet}{f}$
6 $f_{V_{mo}}[\,] \longleftarrow f_k$;
7 $q^{\mathrm{R}}(f_{V_{mo}}) = \mathrm{AggQoS}(f_{V_{mo}})$ w.r.t. Formulas in Table 1;
8 **if** $\forall f_k \in f_{V_{mo}} : f_k \in \overset{\circ}{f}$ **then** $f_{V_{mo}}$.*Type*=$\overset{\circ}{f}$ **else** $f_{V_{mo}}$.*Type*=$\overset{\bullet}{f}$;
9 **case** $f_k \in$ *an Or-group*
10 $\overset{\bullet}{f}_{V_{or}}[\,] \longleftarrow f_k$;
11 $q^{\mathrm{R}}(\overset{\bullet}{f}_{V_{or}}) = \mathrm{AggQoS}(\overset{\bullet}{f}_{V_{or}})$ w.r.t. Formulas in Table 2;
12 **case** $f_k \in$ *an Alternative-group*
13 $\overset{\bullet}{f}_{V_{xor}}[\,] \longleftarrow f_k$;
14 $q^{\mathrm{R}}(\overset{\bullet}{f}_{V_{xor}}) = \mathrm{AggQoS}(\overset{\bullet}{f}_{V_{xor}})$ w.r.t. Formulas in Table 2;
15 **end**
16 $f_V[\,] \longleftarrow f_{V_{mo}}, \overset{\bullet}{f}_{V_{or}}, \overset{\bullet}{f}_{V_{xor}}$;
17 $q^{\mathrm{R}}(f_V) = \mathrm{AggQoS}(f_V)$ w.r.t. Formulas given in Tables 1,2;
18 **if** $\exists f_k \in f_V : f_i \in \overset{\bullet}{f}$ **then** f_V.*Type*=$\overset{\bullet}{f}$ **else** f_V.*Type*=$\overset{\circ}{f}$;
19 **end**
20 **return** $q^{\mathrm{R}}(f_V)$
21 **end**

The control flow information is used for identifying the pattern in each of the SESE components. The virtual features, which are denoted as $f_{V_{mo}}$, $f_{V_{or}}$, $f_{V_{xor}}$, represent *Mandatory-Optional*, *Or* and *Alternative* grouped virtual features, respectively. Quality range values of virtual features are computed by an aggregation function (AggQoS) according to the aggregation rules introduced earlier in the paper and shown in Tables 1 and 2. In order to comply further with the aggregation rules, the type of virtual features should also be determined. Hence, the type of the corresponding virtual feature is labeled as optional if all the collapsed features are optional, otherwise it is labeled as mandatory (i.e., Line 8). However, it is noted that according to the given semantic descriptions of feature variability of *Or* and *Alternative groups*, corresponding virtual

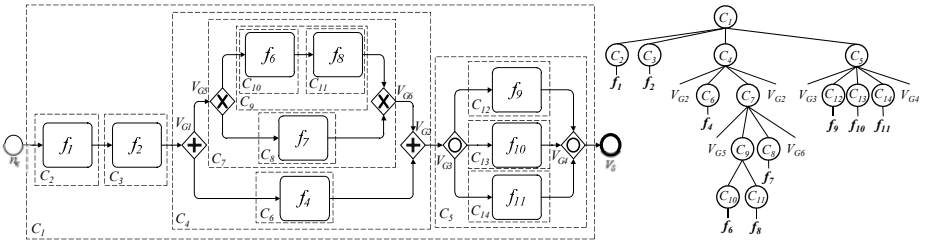


Fig. 3. a) Decomposition of business process model to SESE components b) Business process structure tree

features $f_{V_{or}}$ and $f_{V_{xor}}$ are labeled mandatory. The virtual feature f_V includes all of the collapsed virtual features, which are grouped in each basic composition patterns, and its type is determined such that if there is at least one mandatory feature in a grouped feature, the type of the collapsed virtual features is considered mandatory as well (line 18).

To exemplify the aggregation algorithm described above, we compute the QoS range values of cost (q_{pr}) for the payment feature in the feature model shown in Fig.1. Fig.4 depicts the step-wise transformation of the feature model through gradual hierarchical aggregations. The following is also the step-wise process for computing the QoS range values for the payment feature. To show how each of the transformations is actually performed and how the range values are computed, we refer to the relevant lines of the algorithm that is used besides each step.

$$q^{\mathrm{R}}_{pr}(f) = \left\{ [q^{\mathrm{LB}}_{pr}, q^{\mathrm{UB}}_{pr}] := q^{\mathrm{R}}_{pr}(f_V) | f_V = \mathrm{Agg.} \bigcup_{i=1}^{n} f_i \right\} \quad \text{(Alg.1 lines 2-11)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_1) = [8,20];\ q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_2) = [17,48];\ \ q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_4) = [10,54]; \quad \text{(Alg.1 lines 3)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_6), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_8)\}}_{\overset{1}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_8), \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_8), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_6)\}\right]}_{\text{Tablel-Agg.rule No.1}} = [12,47]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_7)\}}_{\overset{2}{\bowtie}} = \underbrace{\left[\min\left(\{q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_7)\}\right), \max\left(\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_{V_1}), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_7)\}\right)\right]}_{\text{Tablel-Agg.rule No.6}} \quad \text{(Alg.2 lines 5-8)}$$

$$= [12,51];$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_2}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_4)\}}_{\overset{3}{\bowtie}} = \underbrace{\left[q^{\mathrm{LB}}_{pr}(\overset{\bullet}{f}_3), \sum\{q^{\mathrm{UB}}_{pr}(\overset{\bullet}{f}_3), q^{\mathrm{UB}}_{pr}(\overset{\circ}{f}_4)\}\right]}_{\text{Table 1.Agg. rule No. 3}} = [12,105]; \quad \text{(Alg.2 lines 5-8)}$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_5) = \underbrace{\left[\min\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{LB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_2}\right), \max\left(\sum_{f_i \in F_{Sub}} q^{\mathrm{UB}}_{pr}(f_i) : \forall F_{Sub} \in F_{C^3_3}\right)\right]}_{\overset{4}{\bowtie}\ \text{Table 2.Agg. rule No.7}} \quad \text{(Alg.2 lines 9-11)}$$

$$= \left[\min\left(\sum \{\{f_9, f_{10}\}, \{f_9, f_{11}\}, \{f_{10}, f_{11}\}\right), \max\left(\sum\{f_9, f_{10}, f_{11}\}\right)\right] = \left[\min\left(\{18,13,11\}\right), 97\right] = [11,97];$$

$$q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_3}) = \underbrace{\{q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_2), q^{\mathrm{R}}_{pr}(\overset{\circ}{f}_5), q^{\mathrm{R}}_{pr}(\overset{\bullet}{f}_{V_2})\}}_{\overset{5}{\bowtie}} \quad \text{(Alg.2 lines 5-8)}$$

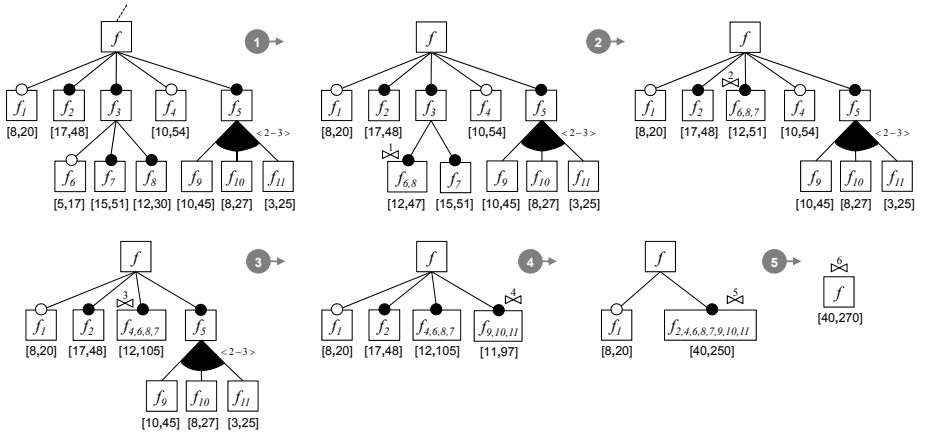


Fig. 4. Step-wise feature model transformation

$$= \underbrace{\left[\sum\{q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_{V_2})\}\ , \sum\{q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{UB}}(\overset{\circ}{f}_5), q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table 1.Agg. rule No. 1}} = [40, 250];$$

$$q_{pr}^{\mathrm{R}}(f) = \underbrace{\{q_{pr}^{\mathrm{R}}(\overset{\circ}{f}_1), q_{pr}^{\mathrm{R}}(\overset{\bullet}{f}_{V_3})\}}_{\bowtie^{6}} = \underbrace{\left[q_{pr}^{\mathrm{LB}}(\overset{\bullet}{f}_{V_3}), \sum\{q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_2), q_{pr}^{\mathrm{UB}}(\overset{\circ}{f}_5), q_{pr}^{\mathrm{UB}}(\overset{\bullet}{f}_{V_2})\}\right]}_{\text{Table 1.Agg. rule No.1}} = [40, 270]$$

It should be noted that the cost represents a QoS property in this example, which follows certain aggregation rules as described formerly; however, different rules are applied for aggregation of range values for other QoS properties, e.g., response time.

4 Discussion

In this section, we analyze the computational complexity of the proposed aggregation method, and then critically discuss its advantages and disadvantages.

4.1 Complexity Evaluation

The proposed QoS aggregation model includes the following three high-level steps: 1) quality range aggregation of features for feature models; 2) process structure tree construction related to each feature and finding canonical process components based on composition patterns; and 3) aggregation of QoS of process components w.r.t. aggregation rules and their propagation over the feature model.

The size of the state-space in a feature model depends primarily on the size of the given feature model graph $G_{FM}(V, E)$. Backtracking of a feature model produces an ordering of the features in which the parent nodes are placed in post-order of their ancestors. The traversal of a feature model requires $O(|V_{FM}|+|E_{FM}|)$, which has linear time complexity. Given the presence of integrity constraints (includes and excludes relations) in a feature model, the size of the resulting graph will be proportional to the number of

constraints. This step could have exponential time complexity if the number of integrity constraints on a feature model is too high. We show below that this is usually not the case.

In the second step, the modular decomposition of business processes and the construction of PST is performed in linear-time proportional to the size of the directed graph of the business process (see [16]). The third step of the algorithm for computing and aggregating quality range values is achieved by parsing the PST of the business process model $G_{BP}(V, E)$ and control flow analysis via alternative post-order depth-first traversals. This step requires $O((|V_{BP}|+|E_{BP}|).(C+S))$, where C and S denote the execution time of control flow analysis for each process component; and computing quality range values according to both the aggregation rules and the size of candidate service lists for each activity. The time required for grouping features based on variability patterns and capturing quality values does not add any computational complexity and can hence be ignored.

As a result, given that the worst-case time complexity of the first step can be in some cases exponential, the time complexity of the entire algorithm can be exponential in the worst-case. However, it is important to note that the algorithm has a linear time complexity when the number of integrity constraints is smaller than the number of features in a feature model, which is usually the case based on our analysis of standard feature models available at `http://www.splot-research.org/`, where the number of integrity constraints to the number of features ratio is approximately 0.18.

4.2 Critical Analysis

As mentioned earlier, we have made some assumptions regarding the topology of the business process graph, i.e., the proposed aggregation method is performed over well-structured business process models. Well-structured business processes have a number of desirable properties, which result in less sophisticated verification mechanisms [7]. However, such an assumption requires the transformation of any ad hoc business process graph into a structural business process model. Substantial amount of work for the transformation of unstructured and arbitrary business process models to structured models already exists [13,17,8,7]. Therefore, our proposed approach can be further adapted and applied to both structured and unstructured business process models (through transformation). The limitation of the presented aggregation model is that it has not considered the integrity constraints and dependencies between features to deliver a more precise aggregation of QoS values. This is left for future work.

Revisiting our original formulated challenge, the goal of QoS range aggregation is to ensure that the required quality levels are achieved for SOSPL architecture for each product in the family. The proposed quality aggregation model considers variability patterns from both structural and behavioral perspectives. The presented approach is a step towards achieving quality-aware product line configuration. Even in this early stage of the development, this approach supports quality aware staged configuration. It can be used to assure that every stage yields a subset of products whose quality ranges satisfy the desired QoS ranges. Finally, this work can be further considered for facilitating the management and customization of multi-tenant cloud applications [18].

5 Related Work

There are several contributions and previous studies addressing QoS aggregation in terms of different process model structures for composite services. The works of Cardoso et al. [11] and Jaeger et al. [12] are the seminal works in the literature, which address the aggregation and estimation of QoS values for Web services composition in well-structured process models. Their approaches are based on (some) workflow patterns in the work by van der Aalst et al. [14]. In [11], the authors propose Stochastic Workflow Reduction (SWR) to compute and estimate the entire workflow QoS values. The SWR algorithm iteratively applies a set of reduction rules for some sequential and parallel patterns over a given structured process graph. Their proposed algorithm for aggregation makes it possible to predict the QoS performance of the entire process by repeatedly performing substitution until the whole process is transformed into one composite service node. In [12], where the aggregation method is the most similar to ours, a QoS aggregation method is proposed for composite Web services by considering workflow patterns and computing upper and lower bounds for QoS values. Authors represent a process model as a graph which is collapsed step-by-step by applying composition patterns. Hwang et al. [19] have proposed a probability-based method where a composite service is represented by a process structure, which is recursively parsed and analyzed to aggregate quality attributes. In a more recent work [13], an aggregation approach employs RPST and supports process models which include unstructured components.

Most of the above studies are related to our proposal. However, existing solutions do not consider the constituent *structural variability* of process models and do not address modeling and managing variation points, which may occur within such an architecture and can significantly impact the proper QoS aggregation. To the best of our knowledge, this is the first work that takes both structural and behavioral variability into account for evaluating QoS dimensions in the context of SOSPL.

6 Conclusion

In this paper, we have provided a systematic approach for QoS range aggregation to support evaluation of quality ranges captured by SOSPL(s), which further helps us for quality-aware product derivation. As the main contribution, we have identified a set of variability patterns which may occur within composition patterns and proposed new aggregation rules for QoS range computation. We also presented an algorithm that analyzes variability and process models to aggregates QoS ranges for an SOSPL. The present work is a continuation of our previous works [3,4], where we have presented approaches for configuration of SOSPLs in the application engineering lifecyle. In those works, we also proposed a method for features prioritization based on stakeholders' objectives and preferences concerning functional and QoS requirements. We also presented how that method can be leveraged by using linear optimization methods for optimal service selection within boundaries of constraints specified by the stakeholders. The approach presented in this paper takes our previous work to next stage where automatic computation of quality ranges in the presence of variability is made possible. As the future work, we also intend to evaluate how our proposed approaches for configuration enabled by the contribution of this work can be applied to real-world scenarios.

References

1. Cohen, S.G., Krut, R.: Managing variation in services in a software product line context. Technical Report SEI-2010-TN-007, Carnegie Mellon University (2010)
2. Lee, J., Kotonya, G.: Combining service-orientation with product line engineering. IEEE Software 27, 35–41 (2010)
3. Mohabbati, B., Hatala, M., Gašević, D., Asadi, M., Bošković, M.: Development and configuration of service-oriented systems families. In: Proceedings of the 2011 ACM Symposium on Applied Computing, SAC 2011, pp. 1606–1613. ACM, New York (2011)
4. Bagheri, E., Asadi, M., Gasevic, D., Soltani, S.: Stratified Analytic Hierarchy Process: Prioritization and Selection of Software Features. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 300–315. Springer, Heidelberg (2010)
5. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
6. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
7. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
8. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. 68, 793–818 (2009)
9. 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, 311–327 (2004)
10. 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)
11. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. 1, 281–308 (2004)
12. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: Qos aggregation for web service composition using workflow patterns. In: Proceedings of the Eighth IEEE International Conference on Enterprise Distributed Object Computing, pp. 149–159. IEEE Computer Society, Washington, DC, USA (2004)
13. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate Quality of Service Computation for Composite Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
14. Van Der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases 14, 5–51 (2003)
15. 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)
16. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Appl. Math. 145, 198–209 (2005)
17. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. 19, 2:1–2:37 (2009)
18. van der Aalst, W.M.P.: Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 8–25. Springer, Heidelberg (2010)
19. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the qos of web-services-based workflows. Inf. Sci. 177, 5484–5503 (2007)

Optimization of Complex QoS-Aware Service Compositions

Dieter Schuller[1], Artem Polyvyanyy[2],
Luciano García-Bañuelos[3], and Stefan Schulte[1]

[1] Technische Universität Darmstadt, Multimedia Communications Lab
{dieter.schuller,stefan.schulte}@kom.tu-darmstadt.de
[2] Hasso Plattner Institute at the University of Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de
[3] Institute of Computer Science, University of Tartu, Estonia
luciano.garcia@ut.ee

Abstract. In Service-oriented Architectures, business processes can be realized by composing loosely coupled services. The problem of QoS-aware service composition is widely recognized in the literature. Existing approaches on computing an optimal solution to this problem tackle structured business processes, i.e., business processes which are composed of XOR-block, AND-block, and repeat loop orchestration components. As of yet, OR-block and unstructured orchestration components have not been sufficiently considered in the context of QoS-aware service composition. The work at hand addresses this shortcoming. An approach for computing an optimal solution to the service composition problem is proposed considering the structured orchestration components, such as AND/XOR/OR-block and repeat loop, as well as unstructured orchestration components.

Keywords: Service composition, Quality of Service, Optimization, Structured and unstructured orchestration components.

1 Introduction

To support and enable agile business processes, the Service-oriented Architecture (SOA) paradigm is often recommended [1]. One of the key features of SOA is that (IT-supported) business processes and, respectively, workflows are realized by composing loosely coupled services – a practice known as service composition. These services autonomously provide a more or less coarse-/fine-grained functionality [2]. Following the vision of the Internet of Services, multiple service providers offer various services at different service marketplaces. If multiple services, which are equally appropriate to accomplish certain tasks, are available at service marketplaces, enterprises can choose to compose those services which meet cost and Quality of Service (QoS) constraints best. This service composition problem (SCP), which forms an optimization problem, is widely recognized in the literature and has been discussed by several authors, e.g. [3,4,5,6,7]. An optimal solution to the SCP constitutes an execution plan, i.e., a set of selected

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 452–466, 2011.

services, which achieves an efficient business process execution with respect to specified cost and QoS requirements.

As real world business processes do not solely consist of structured orchestration components, it is necessary to account for complex structures when composing services. However, existing approaches aiming at computing an optimal solution to the SCP consider complex structures only insufficiently. They usually examine all execution paths resulting from conditional branchings or repeat loops, which leads to significant drawbacks, cf. Section 2. Our approach does not need every possible path of a business process to be examined separately and, thus, does not need all execution paths to be known. We are able to account for OR-blocks. Moreover, our approach goes beyond structured orchestration components by considering unstructured components, viz. unstructured Directed Acyclic Graphs (DAG[1]). To the best of our knowledge, DAGs have not been addressed previously in the context of QoS-aware service composition.

Thus, the work at hand significantly extends the related work in the field of computing optimal solutions to the SCP. We initially formulate the SCP as a non-linear optimization problem and transform it into a linear one. The linear optimization problem is then optimally solved by applying Integer Linear Programming (ILP) techniques from the field of operations research [8,9].

The remainder of the paper is structured as follows: In Section 2, we distinguish our approach from related work. The orchestration models and components considered in the work at hand are introduced in Section 3. We discuss QoS aggregation functions in Section 5 after having presented the applied system model in Section 4. Based on the aggregation functions, the SCP is formulated as an optimization problem in Section 6 and our solution to this problem is evaluated in Section 7. Finally, Section 8 draws conclusions and outlines next steps in our research.

2 Related Work

As already mentioned, the SCP is widely recognized in the literature. A survey of current approaches to the SCP can be found in [7]. The related work in this area can be broadly divided into two groups: heuristic suboptimal SCP methods, e.g., [5,10,11,12,13,14], and optimal SCP methods, e.g., [4,6,15,16].

In [10], the authors present and evaluate (basic) heuristic algorithms, such as greedy and pattern-wise selection. A hill-climbing approach is proposed in [5]. In the same vein, [11,12,13] tackle the optimization problem with genetic algorithms. In all the above cases, the input orchestration is assumed to be structured.

As for optimal SCP methods, the common approach is to analyze every possible execution path. Zeng et al. [15] compute an optimal solution for every execution path. They require a merging step afterwards to account for situations where different services have been selected for the same task in different execution plans. Thus, it is not guaranteed that the merged execution plans still provide the optimal solution as the authors do not consider probabilities of XOR-blocks for the optimization. Anselmi et al. [4] consider all possible execution paths in a single optimization problem; however, they consider probabilities

[1] In the following, we omit explicitly stating that DAG components are unstructured.

for the different execution paths only in the objective function. Thus, they fail to integrate probabilities into the process restrictions, which leads to a worst-case analysis. In other words, the result of ignoring probabilities regarding different branchings for the process restrictions is that, effectively, only the worst path of an XOR-block is considered although all possible execution paths are required to be integrated into the optimization. Our approach allows probabilities for the specification of process restrictions to be considered. This enables an average-case analysis in addition to the worst-case analysis. In [6], Huang et al. consider only one execution path for the optimization – the worst one. Hence, they also fail to consider probabilities of conditional branchings. Ardagna et al. [16] do not consider conditional branchings at all, but they account for repeat loops, cf. Section 3.2. They unfold the cyclic structure and consider all the resulting execution paths. Similar to Anselmi et al., Ardagna et al. integrate all execution paths into a single optimization problem. Thus, the number of constraints grows with the number of considered repeated executions. Furthermore, such an approach does not allow for limiting behavior considerations for a repeat loop.

The work at hand addresses computing an optimal solution to the SCP. In contrast to the related work mentioned above, we do not need to identify all possible execution paths of a given orchestration model. Our approach allows for the consideration of probabilities when specifying constraints for conditional branchings. Thus, we account for all execution possibilities directly. Furthermore, we can perform limiting behavior considerations regarding repeat loops. In addition, our approach accounts for OR-blocks and DAGs. In the next section, we provide a detailed description of all the considered orchestration components.

3 Orchestration Models and Components

3.1 Orchestration Models

An (*orchestration*) *model* is a directed graph consisting of edges (n_1, p, n_2), such that n_1 and n_2 are nodes (the source and target of the edge) and p is the edge probability, i.e., probability of taking the edge assuming that the execution of the orchestration model has reached node n_1. Nodes in an orchestration model are of two types: *tasks* and *gateways*. Tasks represent units of work that are accomplished by atomic services. Gateways encode the routing logic of the orchestration model. Gateways are of three types, cf. [17]: XOR gateways represent conditional branching (XOR-split) or merging of exclusive branches (XOR-join). AND gateways represent parallel forking

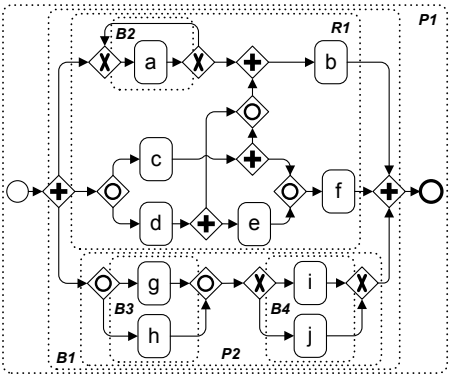


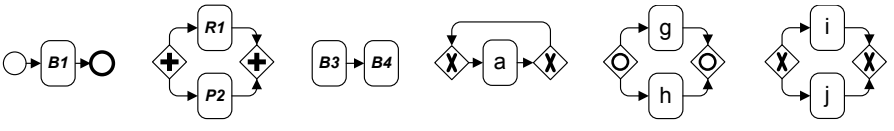
Fig. 1. An orchestration model

(AND-split) or synchronization points (AND-join). OR gateways represent multiple choice (OR-split) or general synchronizing merge (OR-join). A split gateway is the one with a single incoming edge and multiple outgoing edges, while a join gateway is the one with multiple incoming edges and a single outgoing edge.

We expect all orchestration models to be *well-formed*. A well-formed orchestration model meets the following requirements: (i) An orchestration model has a single *source* node, i.e., a node with no incoming edges, and a single *sink* node, i.e., a node with no outgoing edges. (ii) Every node is on a path from the source to the sink. (iii) Every task node has at most one incoming and at most one outgoing edge. (iv) Every gateway is either a split or a join. (v) The sum of the probabilities attached to the outgoing edges of an XOR-split gateway is 1. (vi) The sum of the probabilities attached to the outgoing edges of an OR-split gateway is larger or equal to 1. (vii) An edge whose source is neither an XOR- nor an OR-split gateway has a probability of 1. Fig. 1 shows a well-formed orchestration model using BPMN notation (without edge probabilities).

3.2 Orchestration Components

An orchestration model can be parsed into a hierarchy of (*orchestration*) *components*, each with a single entry and single exit node. Such orchestration components constitute logically independent units of work in the orchestration model. The result of the parsing procedure is a *parse tree*, which is the containment hierarchy of orchestration components of the orchestration model.



(a) P1 (b) B1 (c) P2 (d) B2 (e) B3 (f) B4

Fig. 2. Orchestration components

The *Refined Process Structure Tree* (RPST) is a technique for workflow graph parsing [18,19], i.e., for discovering the structure of a workflow graph. The RPST of an orchestration model is the set of all its canonical orchestration components. An orchestration component is canonical, if it does not overlap (on edges) with any other orchestration component of the orchestration model.

The set of all canonical orchestration components of a model clearly forms a hierarchy that can be represented as a tree. The parent of an orchestration component is the smallest component that contains it. The root of the tree captures the entire orchestration model. A leaf of the tree is an edge of the model. Orchestration components can be classified based on their structure, cf. [19] for details. In

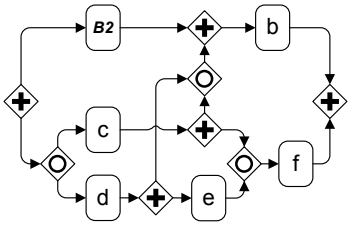


Fig. 3. DAG component R1

the following, we assume that orchestration models are composed of the following structural classes of orchestration components: sequence, AND-block, XOR-block, OR-block, repeat loop, and DAG. Fig. 2 shows all structured orchestration components of the model in Fig. 1. Note that in the following, when referring to a component, we abstract from the internal logic of its child components (see boxes with dotted borderlines in Fig. 1). In the figure, $P1$ and $P2$ are sequences, $B1$ is an AND-block, $B4$ is an XOR-block, $B3$ is an OR-block, and $B2$ is a repeat loop component. Finally, Fig. 3 shows the only DAG component of the orchestration model.

4 System Model

This section describes the system model. We label the set of all tasks with I, $i \in I = \{1, ..., i^{\#}\}$. Referring to Fig. 1, the task numbers i correspond to the identifiers of the tasks, i.e., to "a", "b", etc. The order of the task numbers is not important as long as the respective sets are defined properly. Each task is required to be accomplished by exactly one service $j \in J_i = \{1, ..., j_i^{\#}\}$. Whether or not service j is selected for task i is indicated by the decision-variables $x_{ij} \in \{0, 1\}$. In this work, we take execution time e (duration to execute a service in seconds), reliability r (probability of successful service execution), and throughput d (number of service requests the service is able to serve within a certain time interval) as QoS parameters, as well as cost c (charge for a service invocation in cent considering a pay-per-use pricing model) into account. With these parameters – in fact, even with a subset of these parameters – the aggregation types summation, multiplication, and min/max-operator are covered. Thus, the integration of further QoS parameters into the optimization problem is straightforward. We label bounds for the QoS parameters with b_e, b_r, b_d, b_c.

Regarding branchings, the set L of paths is specified as $L = \{1, ..., l^{\#}\}$. Thereby, $l \in L$ indicate the path numbers within a branching. To give an example, we refer to Fig. 2(b). There are two paths l within the AND-block, thus $L = \{1, 2\}$. In order to distinguish multiple sets of paths from each other, we utilize additional indices, i.e., L_a, L_x, L_o, L_g for AND/XOR/OR-blocks and DAGs, and refer to them as branching L_a, L_x, etc. The set $IW_L \subseteq I$ represents the set of tasks within a branching and $IW_l \subseteq IW_L$ the set of tasks within path $l \in L$. The remaining tasks, which are not located within a branching, are covered in the set $IS = I \setminus (IW_l \mid l \in L)$. The probability of executing a certain path l is indicated by p_l. When it comes to repeat loops, we label the probability that a task i is repeated with ρ_i. To give an example for such a repeat situation, assume a task that tries, e.g., to initialize a resource. The probability that the resource is initialized, is indicated by $1 - \rho_i$. In case we *require* the resource to be initialized, we have to repeat the initialization until we achieve that aim.

The above described system model is used to develop the aggregation functions which are proposed in the next section.

5 Aggregation Functions

In this section, we describe QoS aggregation functions. These functions are required to specify the objective function and the constraints of the SCP. In order

to aggregate the QoS values of the considered candidate services for the whole orchestration model, the regarded orchestration components as well as the respective aggregation type of each QoS parameter have to be taken into account. Table 1 indicates aggregation functions for sequence, AND-block, and XOR-block. In a sequence, the QoS of all services has to be aggregated according to the respective aggregation type. Regarding an AND-block, we have to take the path with the highest aggregated execution time – the *critical path* – into account for execution time. For the other QoS parameters, all services within the AND-block are aggregated. For the XOR-block, we perform an *average-case analysis* by considering possible paths l according to their probabilities p_l in contrast to a worst-case analysis, where the worst of the alternative paths is considered for service selection. Further details on these aggregation functions are available in [20]. For the sake of clarity, we define e_s, e_a, e_x, etc. in Table 1 to represent the respective aggregation functions in Section 6.

Table 1. Aggregation Functions

QoS	Sequence	AND-block	XOR-block
e	$e_s := \sum_{i \in IS} \sum_{j \in J_i} e_{ij} x_{ij}$	$e_a := \max_{l \in L}(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij})$	$e_x := \sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij}$
c	$c_s := \sum_{i \in IS} \sum_{j \in J_i} c_{ij} x_{ij}$	$c_a := \sum_{l \in L} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	$c_x := \sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$
r	$r_s := \prod_{i \in IS} \sum_{j \in J_i} r_{ij} x_{ij}$	$r_a := \prod_{l \in L} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	$r_x := \sum_{l \in L} p_l \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$
d	$d_s := \min_{i \in IS}(\sum_{j \in J_i} d_{ij} x_{ij})$	$d_a := \min_{l \in L}(\min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij}))$	$d_x := \sum_{l \in L} p_l \min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij})$

Regarding repeat loops, we propose to perform limiting behavior considerations taking the mentioned probability ρ_i into account. Thus, we exchange e_{ij} for $e^*_{ij} = \frac{1}{1-\rho_i} e_{ij}$, c_{ij} for $c^*_{ij} = \frac{1}{1-\rho_i} c_{ij}$, and r_{ij} for $r^*_{ij} = \frac{(1-\rho_i) r_{ij}}{1-\rho_i r_{ij}}$, cf. [20] for further explanations. Throughput d_{ij} is not affected by a repeat loop.

The application of these aggregation functions implies a sequential arrangement of the process steps within a split and join [20]. To overcome this shortcoming and in order to account for recursive interlacings of the regarded patterns, we propose to abstract from the interlacing by adding a new service, which replaces the interlacing [21]. This new service as well as the computation of its QoS parameters is then considered for the optimization.

In the following, we propose aggregation functions to account for an OR-block and for a DAG component.

5.1 OR-Block

Regarding the pattern multi-choice (OR-split), not only one (as in XOR-block) and not necessarily all (as in AND-block), but any subset of paths can be executed. In an OR-block, the execution of one, two, three, etc. or even all paths is allowed. In fact, it is also possible that none of the alternative paths are executed. All the selected paths are executed in parallel. When combined with an OR-join (as assumed here), a synchronizing merge is carried out. Note that it

is not known before execution which of the alternative paths will be executed – leading to different aggregation functions when considering the average- or worst-case. In the average-case, only started paths are considered for QoS aggregation, whereas in the worst-case, all paths are executed in parallel. In the latter case, the aggregation functions for an AND-block, cf. Table 1, can be applied.

Regarding the average-case, we have to consider the started paths. Therefore, we are required to take all possible combinations of paths l into account. Let $L = \{1, ..., l^{\#}\}$ be the set of all paths l. If only one path is started, we have $\binom{l^{\#}}{1}$ possibilities to select one of them; if two paths are to be started, there are $\binom{l^{\#}}{2}$ alternative paths. For three paths, it would be $\binom{l^{\#}}{3}$ and so on and so forth. Altogether, we have $h^{\#} \coloneqq \sum_{l \in L} \binom{l^{\#}}{l}$ possible combinations. We define $H = \{1, ..., h^{\#}\}$, $h \in H$, as the set that contains an index number h for every possible path combination. In addition, $L^h = \{l \mid l \in L \wedge h \in H\}$ specifies the set containing the selected paths for path combination h.

To make this clear, we refer to Fig. 2(e). As there are two alternative paths after the OR-split (executing g or h), $l^{\#} = 2$ and $L = \{1, 2\}$. There are $\binom{2}{1} = 2$ possibilities to select exactly one path and $\binom{2}{2} = 1$ possibility to select exactly two paths. Adding these possibilities leads to $h^{\#} \coloneqq \sum_{l \in L} \binom{2}{l} = \binom{2}{1} + \binom{2}{2} = 3$ possible path combinations. H would be $H = \{1, 2, 3\}$, and the sets L^h are the following: $L^1 = \{1\}$, $L^2 = \{2\}$, $L^3 = \{1, 2\}$.

We label the probability that a certain combination h of paths is executed with p_h. Thereby, p_0 represents the probability that none of the paths l is executed. We assume $p_0 + \sum_{h \in H} p_h = 1$. For the sake of simplicity and in order to make sure that the execution does not reach a deadlock situation, we set $p_0 = 0$. As the selected paths are executed in parallel, the aggregation functions are based on the functions for an AND-block. We extend these functions by integrating the selection of the respective paths l. The resulting aggregation functions for an OR-block component are depicted in (1) to (4).

$$e_o \coloneqq \sum_{h \in H} p_h \max_{l \in L^h} \Big(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \Big) \tag{1}$$

$$c_o \coloneqq \sum_{h \in H} p_h \sum_{l \in L^h} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij} \tag{2}$$

$$r_o \coloneqq \sum_{h \in H} p_h \prod_{l \in L^h} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij} \tag{3}$$

$$d_o \coloneqq \sum_{h \in H} p_h \min_{l \in L^h} \Big(\min_{i \in IW_l} \Big(\sum_{j \in J_i} d_{ij} x_{ij} \Big) \Big) \tag{4}$$

5.2 Directed Acyclic Graph

In order to account for DAGs, we firstly identify all possible *runs* of a DAG component. A run is a (potentially concurrent) execution path in the DAG along with the probability of the occurrence of this run. Thus, the original DAG can be rewritten in the form of a XOR-block that combines all possible runs – with their

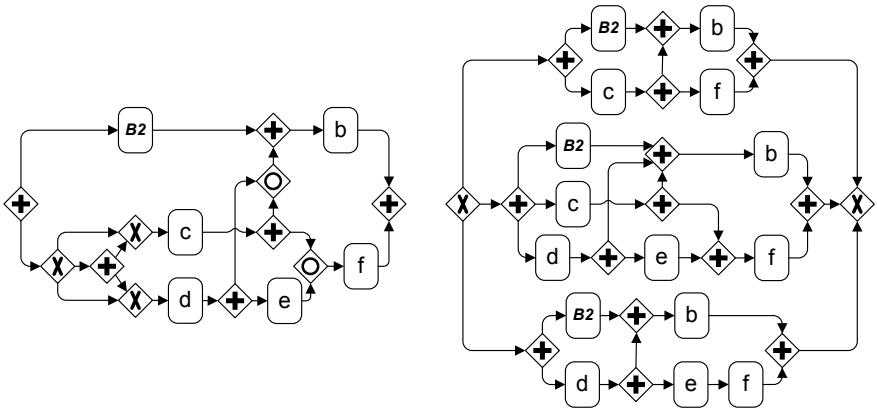


Fig. 4. (a) $R1$ with rewritten OR-splits, and (b) runs of $R1$ combined in a XOR-block

respective aggregated probabilities. An algorithm to compute runs is given in [22] but only considers DAG components composed of XOR and AND gateways and, therefore, has to be extended to account for OR gateways. To this end, every OR-split is transformed into an XOR-split followed by AND-splits according to the path combinations H as described in Section 5.1. To illustrate this, consider the DAG component shown in Fig. 4(a), which is the transformed version of the DAG presented in Fig. 3. Having removed OR-splits, we can now compute runs with a slightly modified version of the algorithm in [22], so as to replace OR-joins by AND-joins where required. Fig. 4(b) presents the XOR-block with the runs computed for $R1$. At this stage, we can apply our aggregation functions for XOR-block from Table 1.

As the tasks in each of the identified runs and the execution paths of the XOR-block respectively, are not arranged sequentially, we again apply our recursive pattern interlacing technique to abstract from the complex N-structure [23] within the runs. This way, we create one new service j_{run} for each path l of the outer XOR-block. To compute the QoS values for each of these new services, we apply the aggregation functions for an AND-block from Table 1 and specify $e_{run}, c_{run}, r_{run}, d_{run}$ in (5)–(8). Please note that we ignore the complexity of the respective N-structures for the QoS parameters c, r, d by applying the respective functions for AND-blocks, as each of the services is executed only once. This is indicated by utilizing L_a instead of L for the set of paths within the N-structure. Regarding the execution time e, the path with the highest aggregated execution time has to be taken into account, as the paths are executed in parallel, cf. [20] for additional explanations.

$$e_{run} := \max_{l \in L} \Big(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \Big) \tag{5}$$

$$c_{run} := \sum_{l \in L_a} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij} \tag{6}$$

$$r_{run} \coloneqq \prod_{l \in L_a} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij} \tag{7}$$

$$d_{run} \coloneqq \min_{l \in L_a} (\min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij})) \tag{8}$$

Having performed this "abstraction" step, we utilize the aggregation functions for XOR-block from Table 1 to aggregate the QoS for each path within the XOR-block according to its respective probability. This is done in (9)–(12) resulting in the aggregation functions for the DAG considered here (indexed with g).

$$e_g \coloneqq \sum_{l \in L} p_l e_{run_l} \tag{9}$$

$$c_g \coloneqq \sum_{l \in L} p_l c_{run_l} \tag{10}$$

$$r_g \coloneqq \sum_{l \in L} p_l r_{run_l} \tag{11}$$

$$d_g \coloneqq \sum_{l \in L} p_l d_{run_l} \tag{12}$$

6 Optimization Problem

As mentioned in the introduction, the SCP describes the problem of selecting and composing those services (from sets of services equally appropriate to accomplish certain tasks) which meet cost and QoS constraints best. In this section, we describe the steps to model the SCP as a linear optimization problem. We, therefore, initially formulate a non-linear optimization problem in Section 6.1 based on the aggregation functions presented in Section 5. In order to obtain the linear optimization problem, in Section 6.2, we conduct adaptations of Model 1 from Section 6.1. Finally, in Section 6.3, we additionally describe a heuristic solution method based on our approach.

6.1 Non-linear Optimization Problem

In order to formulate the non-linear optimization problem in Model 1, we specify an objective function in (13), which is aimed at minimizing the overall cost of the selected services, as well as a set of restrictions for the aggregated QoS values in (14)–(19). We perform an average-case analysis by applying the aggregation functions described in Section 5. For readability reasons, we utilize variables e_a, c_a, r_a, etc. in Model 1 to represent these aggregation functions. Regarding repeat loops, we exchange the QoS parameters e, c, r for the adapted expression e^*, c^*, r^*, as described in Section 5; i.e., if there is a loop at task i, then the respective QoS values of the candidate services j_i appropriate to realize task i are adjusted. Otherwise, the respective QoS values are not modified.

Model 1 depicts the optimization problem in a general form to account for sequences, repeat loops, AND/XOR/OR-blocks, and DAGs that are not interlaced. In (14)–(17), the restrictions for the regarded QoS parameters are specified

Model 1. Generic Service Composition Problem

Objective Function

$$\text{minimize } F(x) = c_s + c_a + c_x + c_o + c_g \tag{13}$$

so that

$$e_s + e_a + e_x + e_o + e_g \leq b_e \tag{14}$$

$$c_s + c_a + c_x + c_o + c_g \leq b_c \tag{15}$$

$$r_s \cdot r_a \cdot r_x \cdot r_o \cdot r_g \geq b_r \tag{16}$$

$$\min(d_s, d_a, d_x, d_o, d_g) \geq b_c \tag{17}$$

$$\sum_{j \in J_i} x_{ij} = 1 \qquad \forall i \in I \tag{18}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J_i \tag{19}$$

by aggregating the considered aggregation functions and restricting them to be lower/greater or equal to the respective bounds b_e, b_c, b_r, b_d for the QoS parameters. For reasons of clarity, we omit defining e_s, e_a, etc. in Model 1, but it has to be noted that their respective equations (from Table 1) are also part of the optimization problem and, therefore, of Model 1. Restriction (18) ensures that every task is accomplished by exactly one service and restriction (19) indicates the integrality restriction.

In order to account for the orchestration model shown in Fig. 1, we utilize the identified orchestration components, cf. Fig. 2 and Fig. 3, and formulate the respective optimization problem in Model 2. Regarding Fig. 2(a) and Fig. 2(b), we apply the aggregation functions for an AND-block. As the orchestration components $R1$ and $P2$ do not belong to the structural class "sequence" (as required for the application of the AND-block formulae), we abstract from their actual structure by creating new services j_{R1} and j_{P2} and use these services for the optimization, i.e., we apply the mentioned AND-block aggregation function for j_{R1}, j_{P2} in (21)–(24).

Model 2. Optimization Problem for Orchestration Model in Fig. 1

Objective Function

$$\text{minimize } F(x) = c_{R1} + c_{P2} \tag{20}$$

$$\max(e_{R1}, e_{P2}) \leq b_e \tag{21}$$

$$c_{R1} + c_{P2} \leq b_c \tag{22}$$

$$r_{R1} \cdot r_{P2} \geq b_r \tag{23}$$

$$\min(d_{R1}, d_{P2}) \geq b_d \tag{24}$$

$$\sum_{j \in J_i} x_{ij} = 1 \qquad \forall i \in I \tag{25}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, \forall j \in J_i \tag{26}$$

In order to calculate the QoS for j_{P2}, we apply the aggregation functions for an OR-block ($B3$) and XOR-block ($B4$) in (27)–(30) with respect to Fig. 2(c).

$$e_o + e_x = e_{P2} \tag{27}$$

$$c_o + c_x = c_{P2} \tag{28}$$

$$r_o \cdot r_x = r_{P2} \tag{29}$$

$$\min(d_o, d_x) = d_{P2} \tag{30}$$

In order to account for DAG $R1$ in Fig. 3, we compute the corresponding choice component as described in Section 5.2 and apply the respective aggregation functions in (5)–(12) to calculate the QoS for j_{R1}. We thereby utilize e^*_{ij}, c^*_{ij}, and r^*_{ij} to account for the repeat loop at $B2$. In analogy to Model 1, the equations for the aggregation functions e_o, e_x, etc. as well as (27)–(30) and (5)–(12) are part of the optimization problem, but are omitted in Model 2 for clarity reasons.

6.2 Linearization of the Non-linear Optimization Problem

In (16), (17), (23), (24), the decision-variables x_{ij} are multiplied and aggregated, respectively, using the min/max-operator, i.e., the decision-variables are aggregated in a non-linear way. As we aim to solve the SCP by applying ILP, we have to adapt these non-linear aggregations.

Regarding the max-operator, e.g., in (21), or in the aggregation function for AND-blocks in Table 1, it has to be noted that if the maximum of a set has to be lower or equal to an upper bound, each element of this set has to fulfill this constraint. Thus, we exchange the term with the max-operator for a new variable, e.g., e_a^{max}, and restrict each element in the max-operator to be lower or equal to e_a^{max}. To make this clear, we exemplify the linearization of e_a, cf. Table 1, in (14) for Model 1. Here, we exchange e_a for e_a^{max} and add restriction (31) to Model 1. To replace the min-operator, we analogously specify variables d^{min} and add appropriate restrictions for each min-operator in (17) to Model 1, cf. [21].

$$\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij} \leq e_a^{max} \qquad \forall l \in L \tag{31}$$

To linearize restrictions (16), (23), where the decision-variables are multiplied with each other, we utilize the approximation in (32) for the aggregation of QoS parameters r, which is very accurate for parameter values z_{ij} that are very close to 1, such as reliability. We thereby avoid having to multiply the reliabilities of the alternative services and allow for summing up the respective approximated reliabilities instead. This way, we avoid the multiplication of decision-variables in all aggregation functions with respect to the QoS parameter r leading to linear aggregation functions and restrictions. For further details, we refer to [20].

$$\prod_{i \in I} \sum_{j \in J_i} z_{ij} x_{ij} \approx 1 - \sum_{i \in I} (1 - \sum_{j \in J_i} z_{ij} x_{ij}) \tag{32}$$

$$\prod_{i \in I} \sum_{j \in J_i} z_{ij} x_{ij} = 1 - \sum_{i \in I} (1 - \sum_{j \in J_i} z_{ij} x_{ij}) + \epsilon \tag{33}$$

Applying the described linearization steps results in a linear optimization problem in Models 1 and 2. An optimal solution can be computed by applying ILP, if a solution exists. But, as we have utilized the approximation in (32) instead multiplying the reliabilities of the alternative services, we inserted an error into the optimization problem. The larger the set of tasks I, the higher is this error. Thus, in order to ensure that our approach actually computes the optimal solution, we compute the value of this error, labeled with ϵ, for the current solution and account for this error by considering its value explicitly in (33). Afterwards, we recompute the optimal solution taking (33) into account. If the resulting execution plan does not change, we obviously found the optimal solution. Otherwise, we recalculate the error and recompute the optimal solution taking this new error into account. This way, we run the optimization at least two times, but in the end, we guarantee that the solution is the optimal one.

6.3 Scalability

Applying our approach, computing the optimal solution to the SCP requires increased computational effort with a growing number of tasks and candidate services per task. To address scalability issues, we propose to relax the integrality restrictions (19) and (26), and to compute a solution by applying mixed integer linear programming (MILP) without considering the error ϵ. This probably results in an invalid solution to the SCP with no explicit indication which service to select for a certain task, as the decision-variables x_{ij} may contain values between 0 and 1 and not exactly 0 or 1. Afterwards, in order to obtain a valid but probably not optimal solution, i.e., $x_{ij} \in \{0, 1\}$, we apply a heuristic selection strategy. Based on the values of the decision variables x_{ij}, we randomly select services which satisfy the constraints. The performance of this heuristic solution method compared to the optimal solution is depicted in Fig. 5 and Fig. 6.

Alternatively, we could interpret the x_{ij} values as probabilities to select respective services for the accomplishment of a certain task. This way, if a business process is executed not only once but multiple times (as is assumed to be the normal case), the business process execution can be seen as realization of a random experiment – selecting respective services based on their probabilities – with minimal average cost satisfying the constraints *in average*.

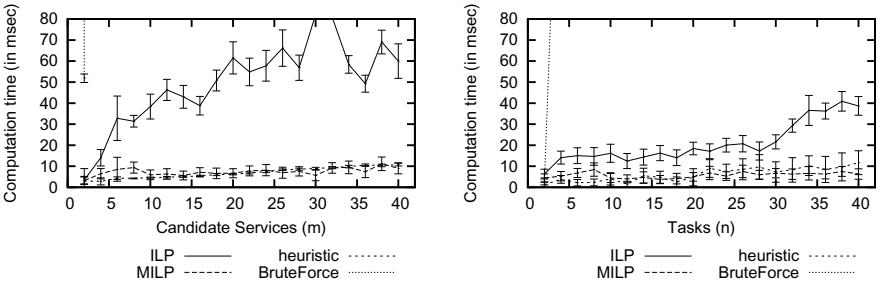
7 Evaluation

As a proof of concept, we implemented our approach to the SCP using the linear programming solver CPLEX[2]. In order to evaluate the efficiency and the solution quality of our approach, i.e., the time for computing the execution plan and its

[2] `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

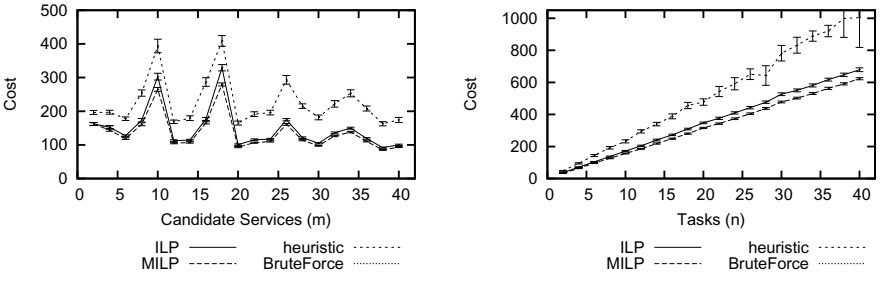
cost, we conducted a series of experiments to compare our approach, which we label with ILP, to the heuristic solution method mentioned in the previous section and to a BruteForce algorithm, which iterates through all possible service combinations. Thus, BruteForce computes the optimal solution *per definitionem*. Our proposition to interpret the values of the decision variables as execution probabilities corresponds to the label MILP in Fig. 5 and Fig. 6. The experiments were performed on an Intel Core 2 Quad processor at 2.66 GHz, 4 GB RAM, running Microsoft Windows 7.

In order to evaluate the influence of the number m of candidate services j_i per task i, we varied m in Fig. 5(a) and Fig. 6(a) from 2 to 40 with step 2 for the orchestration model in Fig. 1, which is composed of the orchestration components sequence, AND-block, XOR-block, OR-block, repeat loop, and DAG. Regarding the influence of the number n of tasks, we varied n from 2 to 40 with step 2 considering 10 candidate services per task. The resulting orchestration models thereby only comprise of sequences.



(a) Impact of candidate services (b) Impact of tasks

Fig. 5. Evaluation of computation time



(a) Impact of candidate services (b) Impact of tasks

Fig. 6. Evaluation of cost

Regarding the efficiency of our ILP approach, we observe that the computation time increases with the number of tasks and candidate services. However, the computation time remains lower than 100 msec. For the heuristic approach as well as for MILP, the computation times increase only slightly. As indicated in Fig. 5(a) and Fig. 5(b), the computation time using the BruteForce algorithm is

greatly increasing. Regarding Fig. 5(a), BruteForce requires 63,883.92 msec for $m = 4$, which is not displayed in Fig. 5(a). For $n = 4$ and $n = 6$, which would be the next plots for BruteForce in Fig. 5(b), the algorithm requires 209.62 msec and 29,261.48 msec, respectively.

With respect to the solution quality, the ILP approach computes an optimal solution, which is indicated in Fig. 6(a) and Fig. 6(b) by comparing the cost of the ILP solution to the cost of BruteForce. As the MILP algorithm does not create integer values for the decision variables, it must not be compared to ILP regarding cost.

The evaluation results show that ILP requires more time than the heuristic method for computing a solution, but the heuristic does not achieve the solution quality of ILP, i.e., the cost for execution plans computed by the heuristic are always higher than cost for execution plans computed by ILP. Compared to the BruteForce algorithm, which also computes the optimal solution, ILP's computation time is rather small.

8 Conclusion

The problem of selecting services based on their QoS – the QoS-aware SCP – is widely recognized in the literature and has been discussed recently by several authors. In the work at hand, we addressed the SCP for orchestration models composed of sequences, AND-blocks, XOR-blocks, OR-block, repeat loops, and DAGs, which has, to date, been insufficiently considered in the literature, cf. Section 2. We thereby aim to compute an optimal solution to the SCP. In our future work, we will focus on considering further structural classes of orchestration components such as loops with multiple entry and/or multiple exit points. We further aim to consider stochastic QoS values for the SCP.

Acknowledgment. This work is supported in part by E-Finance Lab e. V., Frankfurt am Main, Germany (`http://www.efinancelab.com`).

References

1. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Web Information Systems Engineering (WISE), pp. 3–12. IEEE Computer Society (2003)
2. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River (2004)
3. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for QoS-aware web service composition. In: International Conference on Web Services (ICWS), pp. 72–82. IEEE Computer Society (2006)
4. Anselmi, J., Ardagna, D., Cremonesi, P.: A QoS-based selection approach of autonomic grid services. In: Service-Oriented Computing Performance (SOCP), pp. 1–8. ACM (2007)
5. Menascé, D.A., Casalicchio, E., Dubey, V.K.: A heuristic approach to optimal service selection in service oriented architectures. In: Workshop on Software and Performance (WOSP), pp. 13–24. ACM (2008)

6. Huang, A.F.M., Lan, C.W., Yang, S.J.H.: An optimal QoS-based web service selection scheme. Information Sciences (ISCI) 179(19), 3309–3322 (2009)
7. Strunk, A.: QoS-aware service composition: A survey. In: European Conference on Web Services (ECOWS), pp. 67–74. IEEE Computer Society (2010)
8. Hillier, F., Lieberman, G.: Introduction to Operations Research, 8th edn. Mc Graw Hill, Boston (2005)
9. Taha, H.: Operations Research – An Introduction, 8th edn. Pearson Prentice Hall, London (2007)
10. Jaeger, M.C., Mühl, G., Golze, S.: QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 646–661. Springer, Heidelberg (2005)
11. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 1069–1075. ACM (2005)
12. Gao, C., Cai, M., Chen, H.: QoS-aware service composition based on tree-coded genetic algorithm. In: International Computer Software and Applications Conference (COMPSAC), pp. 361–367. IEEE Computer Society (2007)
13. Lécué, F.: Optimizing QoS-Aware Semantic Web Service Composition. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 375–391. Springer, Heidelberg (2009)
14. Mabrouk, N.B., Georgantas, N., Issarny, V.: A semantic end-to-end QoS model for dynamic service oriented environments. In: Workshop on Principles of Engineering Service-oriented Systems (PESOS), pp. 34–41. IEEE Computer Society (2009)
15. 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 (TSE) 30(5), 311–327 (2004)
16. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering (TSE) 33(6), 369–384 (2007)
17. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases (DPD) 14(1), 5–51 (2003)
18. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering (DKE) 68(9), 793–818 (2009)
19. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified Computation and Generalization of the Refined Process Structure Tree. In: Bravetti, M. (ed.) WS-FM 2010. LNCS, vol. 6551, pp. 25–41. Springer, Heidelberg (2011)
20. Schuller, D., Eckert, J., Miede, A., Schulte, S., Steinmetz, R.: QoS-aware service composition for complex workflows. In: International Conference on Internet and Web Applications and Services (ICIW), pp. 333–338. IEEE Computer Society (2010)
21. Schuller, D., Miede, A., Eckert, J., Lampe, U., Papageorgiou, A., Steinmetz, R.: QoS-Based Optimization of Service Compositions for Complex Workflows. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 641–648. Springer, Heidelberg (2010)
22. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate Quality of Service Computation for Composite Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
23. 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)

Goal-Driven Business Process Derivation*

Aditya K. Ghose[1], Nanjangud C. Narendra[2], Karthikeyan Ponnalagu[2], Anurag Panda[3], and Atul Gohad[3]

[1] University of Wollongong, Wollongong, Australia
{aditya.ghose}@gmail.com
[2] IBM Research India, Bangalore, India
{narendra,karthikeyan.ponnalagu}@in.ibm.com
[3] IBM India Software Lab, Bangalore, India
{panurag,agohad}@in.ibm.com

Abstract. Solutions to the problem of deriving business processes from goals are critical in addressing a variety of challenges facing the services and business process management community, and in particular, the challenge of quickly generating large numbers of effective process designs (often a bottleneck in industry-scale deployment of BPM). The problem is similar to the planning problem that has been extensively studied in the artificial intelligence (AI) community. However, the direct application of AI planning techniques places an onerous burden on the analyst, and has proven to be difficult in practice. We propose a practical yet rigorous (semi-automated) algorithm for business process derivation from goals. Our approach relies on being able to decompose process goals to a more refined collection of sub-goals whose ontology is aligned with that of the effects of available tasks which can be used to construct the business process. Once process goals are refined to this level, we are able to generate a process design using a procedure that leverages our earlier work on semantic effect annotation of process designs. We illustrate our ideas throughout this paper with a real-life running example, and also present a proof-of-concept prototype implementation.

Keywords: business process, goals, tasks, capabilities.

1 Introduction

One of the most crucial (and difficult) tasks in enterprises today is the derivation of business processes to meet stated business goals. Poor process derivation could result in wasteful and/or wrong tasks, and would require significant and costly rework to ensure that business process executions are able to adhere to their goals. Additionally, the requirement of business process compliance [1–3] - over and above the basic business goals - adds further complexity and difficulty.

Traditional approaches towards business process derivation from goals have focused on modeling this problem as an artificial intelligence (AI) planning problem [4], e.g., citations such as [5–7]. While such methods undoubtedly produce accurate solutions, they require specialist knowledge of planning and formal knowledge representation techniques that business analysts typically do not possess.

* Thanks to GR Gangadharan for his feedback.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 467–476, 2011.

Therefore, in this paper, we take a different approach. We assume the following inputs: (a) a set of process goals represented as a collection of boolean conditions in conjunctive normal form (CNF); (b) a capability library of existing tasks that can be used to satisfy the goals, with each annotated via its *effects* [8]; (c) a set of *domain constraints* that impose restrictions on how task execution in the business process should be sequenced. Given these inputs, the salient contribution of our paper is an algorithm for deriving a business process design from these inputs.

Our algorithm works as follows. First, the goals are successively refined using goal refinement strategies leveraged from the KAOS methodology [9, 10]. This refinement continues until there is an ontological match with the effects of the tasks in the capability library. Second, using the capability library, tasks are identified for each leaf-level goal. Third, precedence constraints among the tasks are derived from the given domain constraints. Finally, the business process design is generated from the goals and precedence constraints. Our approach also does not require the use of preconditions, which we show can be encoded via domain constraints.

2 Running Example

Our running example is a simplified version of an incident management process. Incidents are customer initiated calls based on service issues. The mission of incident management process is to handle all requests for problem solving and support in a consistent, timely and cost-effective manner. Typically, the process begins with a request from a client or with a problem statement highlighting the concerns of the client. It concludes with the client being satisfied with the response and the solution provided to solve the problem.

The goals and derived sub-goals of this process are depicted in Fig. 1. An AND link in Fig. 1 specifies that all sub-goals of a goal need to be satisfied for the goal to be satisfied; an XOR link specifies that the sub-goals are mutually exclusive, and only one is needed to satisfy the goal. For example, the goal of Incident and Problem management fulfills the goals Fix Problem, Detect Problem and Verify Problem, viz., a case of AND relationship. If we consider the goals Isolate Problem or Escalate Problem they share an XOR Relationship as in any given situation only one of the goals can be fulfilled and they are mutually exclusive in nature.

Some of the applicable domain constraints for this business process are: whether to escalate the problem to the next level, and whether a new incident should be linked to a previous incident in order to enhance reuse of earlier solutions.

3 Background

A *business process* is a sequence of *tasks*, with each task producing an *effect*. The accumulated effects of all task executions is the overall effect of the business process. The effect is the result of an activity executed by a cause or agent. Effects can be viewed as both: *normative* - as they state required outcomes (i.e., goals); and *descriptive* in that they describe the normal, and predicted, subset of all possible outcomes. We formally represent effect annotations using first-order logic. For simplicity, our business process

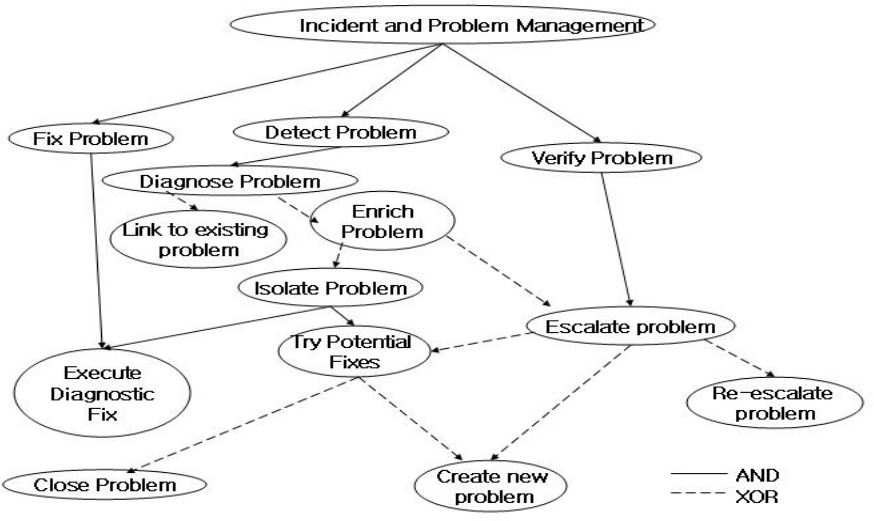


Fig. 1. Goals & Sub-Goals

does not contain loops; they can be incorporated into our business process model by abstracting them into single tasks.

We define a process for *pair-wise effect accumulation* [8, 11], which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. The procedure serves as an easily understandable yet rigorous methodology for analysts to follow. We assume that the effect annotations have been represented in conjunctive normal form (CNF) [12]. Simple techniques (e.g., [12]) exist for translating arbitrary sentences into CNF.

Let $< t_i, t_j >$ be the ordered pair of tasks, and let e_i and e_j be the corresponding pair of effect annotations. Let $e_i = \{c_{i1}, c_{i2}, ..., c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, ..., c_{jn}\}$ (we can view CNF sentences as sets of clauses, without loss of generality). If $e_i \cup e_j$ is consistent, then the resulting cumulative effect is $e_i \cup e_j$. Else, we define $e'_i = \{c_k | c_k \in e_i \; and \; \{c_k\} \cup e_j \; is \; consistent\}$ and the resulting cumulative effect to be $e'_i \cup e_j$. In other words, the cumulative effect of the two tasks consists of the effects of the second task plus as many of the effects of the first task as can be consistently included. We remove those clauses in the effect annotation of the first task that contradict the effects of the second task. The remaining clauses are undone, i.e., these effects are overridden by the second task. In the following, we shall use $acc(e_1, e_2)$ to denote the result of pair-wise effect accumulation of two contiguous tasks t_1 and t_2 with effects e_1 and e_2.

In addition to the effect annotation of each task, we annotate each task t with a cumulative effect E_t. E_t is defined as a set $\{es_1, es_2, ..., es_p\}$ of alternative effect scenarios. Alternative effect scenarios are introduced by OR-joins or XOR-joins, as we shall see below. Cumulative effect annotation involves a left-to-right pass through a sequence of tasks. Tasks which are not connected to any preceding task via a control flow link are annotated with the cumulative effect $\{e\}$ where e is the immediate effect of the task

in question. We accumulate effects through a left-to-right pass of a sequence, applying the pair-wise effect accumulation procedure on contiguous pairs of tasks. The process continues without modification over splits. Joins require special consideration. In the following, we describe the procedure to be followed in the case of 2-way joins only, for brevity. The procedure generalizes in a straightforward manner for n-way joins.

AND-joins: Let t_1 and t_2 be the two tasks immediately preceding an AND-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec1m\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively (where ec_{sc} denotes an effect clause within an effect scenario). Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the AND-join. We define $E = \{acc(ec_{1i}, e) \cup acc(ec_{2j}, e) | ec_{1i} \in E_1 and ec_{2j} \in E2\}$. Note that we do not consider the possibility of a pair of effect scenarios ec_{1i} and ec_{2j} being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models. The result of effect accumulation in the setting described here is denoted by $ANDacc(E_1, E_2, e)$.

XOR-joins: Let t_1 and t_2 be the two tasks immediately preceding an XOR-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec_{1m}\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the XOR-join. We define $E = \{acc(ec_i, e) | ec_i \in E1 or eci \in E2\}$. The result of effect accumulation in the setting described here is denoted by $XORacc(E_1, E_2, e)$.

OR-joins: Let t_1 and t_2 be the two tasks immediately preceding an OR-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec_{1m}\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the OR-join. The result of effect accumulation in the setting described here is denoted by $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \wedge XORacc(E_1, E_2, e)$. Henceforth in our paper, for simplicity, we consider that OR-joins can be represented via XOR-joins themselves.

Pair-wise effect accumulation as described above will form the basis of our business process derivation algorithm, and will enable our algorithm to verify whether the derived business process design does meet the stated goals.

4 Goal Refinement and Constraint Specification

4.1 Goal Refinement

We define the *goals* of a business process as a combination $G_1 \wedge G_2 \wedge \ldots \wedge G_n$ of boolean conditions in CNF, all of which need to be satisfied at the end of the process execution. For example, the goal 'Link to Existing problem' of our running example in Fig. 1 can be represented as follows: *Goal: Achieve[LinkIncidentToProblemTicket] (∀ i: incident, pt: problem ticket, p: problem, it: incident ticket) IsCausedBy(p,i) ⇒ link(it, pt).*

Each boolean condition G_i can itself be broken down into a (conjunctive as well as disjunctive) combination of clauses, each of which is a sub-goal of G_i. The case of conjunction is best illustrated by the goal 'Incident and Problem Management' as it is

a combination of sub goals 'Fix Problem', 'Diagnose Problem' and 'Verify Problem' and we expect all these sub goals to be realized in conjunction. Similarly the disjunctive case is illustrated again by the goal 'Diagnose Problem' as realizing this goal can satisfy one of the goals 'Link to existing problem' or 'Enrich Problem'.

Our goal refinement procedure is based on the KAOS methodology [9]. For a goal G_i, let it be expressed as $G_{i1} \wedge G_{i2} \ldots \wedge G_{im}$, where each sub-goal G_{ij} is of the form $G_{ij1} \vee G_{ij2} \ldots \vee G_{ijl}$. That is, each clause G_{ij} is a purely disjunctive clause. In accordance with [9], we say that the sub-goals *refine* G_i if the following hold:

1. $G_{i1} \wedge G_{i2} \ldots \wedge G_{im} \vdash G_i$ (entailment)
2. $\forall i : \wedge_{j \neq i} G_{ij} \nvdash G_i$ (minimality)
3. $G_{i1} \wedge G_{i2} \ldots \wedge G_{im} \nvdash false$ (consistency)

That is, the set of sub-goals for a goal will achieve the goal (entailment); it will be the smallest set of sub-goals to achieve the goal (minimality); and it will never be incorrect (consistency).

In its turn, each disjunctive clause can itself be refined into a collection of one or more conjunctive clauses, each of which could themselves possess a collection of two or more disjunctive clauses, and so on. Indeed, the presence of a disjunctive clause signifies a set of mutually exclusive options by which the particular sub-goal is to be satisfied. Later in Section 5, we will show how these clauses can be used to design XOR-splits and joins.

Our goal refinement procedure, therefore, refines the overall goals of a business process alternatively using conjunctive and disjunctive clauses, until all sub-goals have been completely specified to the user's satisfaction. The goal model that we presented in Fig. 1, is the outcome of such an exercise.

We define a *singleton* clause in a (refined) goal specification as a clause that is a single literal. In contrast, a non-singleton clause is a disjunctive combination $L_1 \vee L_2 \ldots \vee L_n$ of literals. The relevance of this distinction will become clear in Section 5.1, when we present our business process derivation algorithm.

4.2 Domain Constraint Specification

As we have seen, a goal is merely a collection of boolean conditions, without any specific ordering on how they have to be fulfilled in the business process. In case the analyst desires to impose an ordering, he/she can specify them in the form of what we call *domain constraints*, which are restrictions on the way in which the goal conditions are to be achieved. Business compliance constraints [1–3] can also be specified using our domain constraint formalism.

Formally, we define a domain constraint as a tuple $< \mathcal{C}_i, \mathcal{C}_j, rel >$; where $\mathcal{C}_i$ and $\mathcal{C}_j$ are boolean conditions; and rel is one of the following relations - IMM standing for *immediately*, EVE standing for *eventually*. This is to be interpreted as: the condition $\mathcal{C}_i$ has to be realized in the business process before the condition $\mathcal{C}_j$ can be realized. The operator rel qualifies this constraint by specifying how soon $\mathcal{C}_j$ should be realized after $\mathcal{C}_i$. Please note that our domain constraints are at a higher level of abstraction than task precedence constraints expressible in languages such as concurrent transaction logic (see [7] and the citations contained therein); indeed, later in Section 5.1 we will

show how these constraints are used to create precedence constraints among the derived business process steps.

From the above formulation of domain constraints, the following proposition can be stated.

Proposition 1. *Any precondition can be represented via a domain constraint.*

Proof: If a task T_i has no predecessors, then it is the starting task of a business process, and its precondition can be represented via the domain constraint $< precondition(T_i), effect(T_i), IMM >$. If T_i has at least one predecessor, then its precondition can be represented as a boolean condition $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, where each C_k is an effect of a predecessor. If the effect of T_i is $effect(T_i)$, then the precondition of T_i can be represented via the set of domain constraints $\{< C_k, effect(T_i), IMM >\}, 0 \leq k \leq n$. **QED**

In our running example, considering the goal 'Try Potential Fixes', we can construct the domain constraint as: $< IsProblemIsolated(it) \wedge AreKnownFixesAvaliable(it), TryPotentialFixes(it), EVE >$, where *it* denotes the incident ticket, $IsProblemIsolated(it)$ and $AreKnownFixesAvailable(it)$ as boolean conditions share the relation EVE with $TryPotentialFixes(it)$, again another boolean condition. Similarly the condition 'CanCreateNewProblem' has to be realized after the condition 'CannotEscalate' in realizing the goal 'CreateNewProblem' and they share the relation IMM.

5 Process Derivation from Goals

For deriving a business process design from goals and constraints, we assume the following inputs: a set of effect-annotated tasks in a capability library; a set of goals and sub-goals, refined until the level of an ontological match with the effects of the tasks in the capability library; and a set of domain constraints.

5.1 Process Derivation Algorithm

Our process derivation algorithm takes as input the refined (i.e., ontologically matching with effects) goals G & domain constraints DC, and effect-annotated tasks in the capability library, and produces a set of effect-annotated process steps PS and a set of *precedence constraints* PREC among the process steps. A precedence constraint among two process tasks $T_i \stackrel{rel}{\rightarrow} T_j$ specifies the order in which each task should execute vis-a-vis the other, and where $rel \in \{IMM, EVE\}$, with IMM standing for $immediately$ and EVE standing for $eventually$. The former type of precedence specifies that T_j must execute immediately after T_i has executed, whereas the latter specifies that T_j can executed any time after T_i has executed.

Our algorithm consists of the following steps. First, it distinguishes between singleton and non-singleton clauses; for each singleton clause, it identifies the appropriate task in the capability library whose effect entails the clause, and adds that task to the set of process steps PS. For each non-singleton clause, however, the algorithm determines a collection of tasks whose collective effects entail the clause. It then adds the tasks to PS,

along with an appropriate XOR gateway. Second, the algorithm generates precedence constraints from the domain constraints. Third, the algorithm evaluates the generated precedence constraints for inconsistencies and alerts the user in case it discovers any, so that the user can resolve the inconsistencies. Finally, the algorithm generates a business process design from the (user-resolved) precedence constraints.

Precedence Constraint Generation: For now, we consider the sub-case when both $\mathcal{C}_i$ and $\mathcal{C}_j$ can be fulfilled by single tasks; the sub-case when either $\mathcal{C}_i$ or $\mathcal{C}_j$ is fulfilled by a disjunctive combination of tasks, is dealt with under XOR gateways.

Hence our algorithm for generating precedence constraints from the domain constraint $< \mathcal{C}_i, \mathcal{C}_j, rel >$, with each condition represented by a single task, works as follows. First, each condition $\mathcal{C}_i$ and $\mathcal{C}_j$ is analyzed, and the appropriate process tasks that fulfil the condition, are identified. Second, for each pair of tasks T_i, T_j, with T_i (resp. T_j) pertaining to $\mathcal{C}_i$ (resp. $\mathcal{C}_j$), the precedence constraint $T_i \overset{rel}{\rightarrow} T_j$ is generated, where rel is represented by EVE or IMM.

XOR Gateway Generation: We represent a disjunctive clause via an XOR gateway. In addition, we also need to accommodate domain constraints of the form $< \mathcal{C}_i, \mathcal{C}_j, rel >$, where either $\mathcal{C}_i$ or $\mathcal{C}_j$ is a disjunctive clause, and where one of either $\mathcal{C}_i$ or $\mathcal{C}_j$ is a non-singleton clause. This is needed in order to generate the appropriate precedence constraints from these domain constraints. Hence if such a domain constraint exists, we have three sub-cases:

1. Only $\mathcal{C}_i$ is a disjunctive clause: $\mathcal{C}_i$ would be represented via an XOR gateway $T_{i1} \vee T_{i2} \ldots T_{im}$, by tasks $T_{i1} \ldots T_{im}$ that collectively fulfill condition $\mathcal{C}_i$; and $\mathcal{C}_j$ by the single task T_j that fulfills condition $\mathcal{C}_j$. For this sub-case, we create a "dummy" XOR-join node $T_{i,m+1}$, whose effect is $\mathcal{C}_i$; and we create the following precedence constraints: $T_{ik} \overset{IMM}{\rightarrow} T_{i,m+1}, k = 1, \ldots, m$, and $T_{i,m+1} \overset{rel}{\rightarrow} T_j$.
2. Only $\mathcal{C}_j$ is a disjunctive clause: this is the reverse of the above sub-case; if $\mathcal{C}_j$ is represented via the XOR gateway $T_{j1} \vee T_{j2} \ldots T_{jm}$, and $\mathcal{C}_i$ by the task T_i, then the precedence constraints, $T_i \overset{rel}{\rightarrow} T_{jk}, k = 1, \ldots, m$, are generated.
3. Both $\mathcal{C}_i$ and $\mathcal{C}_j$ are disjunctive clauses: Let $\mathcal{C}_i$ be represented by an XOR gateway with p_i paths, and $\mathcal{C}_j$ be represented by an XOR gateway with p_j paths. Then, as in the first sub-case above, a "dummy" XOR-join node $T_{i,m+1}$, is first generated, whose effect is $\mathcal{C}_i$. Next, for each node $T_{ik}, 0 \leq k \leq j$ on the XOR gateway whose effect is $\mathcal{C}_j$, the following precedence constraint is generated: $T_{i,m+1} \overset{rel}{\rightarrow} T_{ik}$, $0 \leq k \leq j$.

For instance, our running example shows different types of problems based on their escalation support; escalation/Non-escalation cases would therefore differ. Hence there are 4 possible branches in the above scenario just based on the support for escalation at a given level. The user can later tweak the XOR gateway by pruning the variables, and thereby, the number of branches. However, that is beyond the scope of this algorithm.

Inconsistency Resolution & Business Process Design Generation: Once the precedence constraints are generated, inconsistencies could arise. For any pair of tasks T_i and T_j, we define an *inconsistency* as the existence of two precedence constraints that

are mutually conflicting. That is, if there are two precedence constraints $T_i \stackrel{rel}{\rightarrow} T_j$ and $T_j \stackrel{rel}{\rightarrow} T_i$, where $rel \in \{IMM, EVE\}$, then this is an inconsistency. For each precedence constraint $T_i \stackrel{rel}{\rightarrow} T_j$, our inconsistency detection procedure checks whether there exists a (direct or transitively obtained) constraint $T_j \stackrel{rel}{\rightarrow} T_i$. Inconsistencies are flagged to the user, who will then need to resolve them manually. (We will be investigating automated inconsistency resolution for future work.)

The actual generation of the business process design, assumes that all inconsistencies have been resolved by the user. It basically consists of adding edges between tasks T_i and T_j based on the derived precedence constraints, whether t_i is supposed to *immediately* or *eventually* precede T_j. For the former case, the edge between two tasks is added right away. For the latter, on the other hand, we first check whether a chain of *immediately*-type constraints already exists on a path between the tasks. If so, then the last task on this chain is made the predecessor of T_j. If not, then T_i itself is made T_j's predecessor. While doing so, the algorithm also uses the effect accumulation procedure described in Section 3 to verify the compatibility of the business process under generation with the refined goals.

6 Prototype Implementation

Our prototype implementation is built as a plugin on IBM's Rational Software Architect (RSA) tool, and is depicted in Fig. 2. It was tested on a PC with 3.2 GHz processor speed and 3 GB of RAM. For our running example, once the user recorded the goals and domain constraints, the business process was generated within one minute.

The plugin provides the business analyst with various views that help define the inputs to business process derivation. The Goal Modeling view provides options to define the goals and sub-goals. Expected effect outcomes can be added as annotations for the respective goals and sub-goals that are represented in the AND/XOR logic. The domain constraints are defined in the Constraint Modeling view, which also helps generate the precedence constraints. The Capability Modeling view helps add various capabilities that can be used in order to fulfill the goals. Based on the capability availability and matching of capability effects with that of the specific goal/sub-goal in question, the goal-capability matching is arrived at. This helps in derivation of the incident management business process in BPMN format, which is also depicted in Fig. 2.

7 Related Work

Planning-based Business Process Derivation: The work reported in this paper is inspired in part by planning techniques from AI [4], in particular, partial-order planning. However, as we have already shown, such techniques require non-trivial adjustments in order to make them usable for business process derivation. Our business process derivation technique is also inspired in part by our earlier work [13], where we proposed a technique to map user's process goals into scenario descriptions described in the form of sequence diagrams. Appropriate composition of the sequence diagrams yields the final business process design.

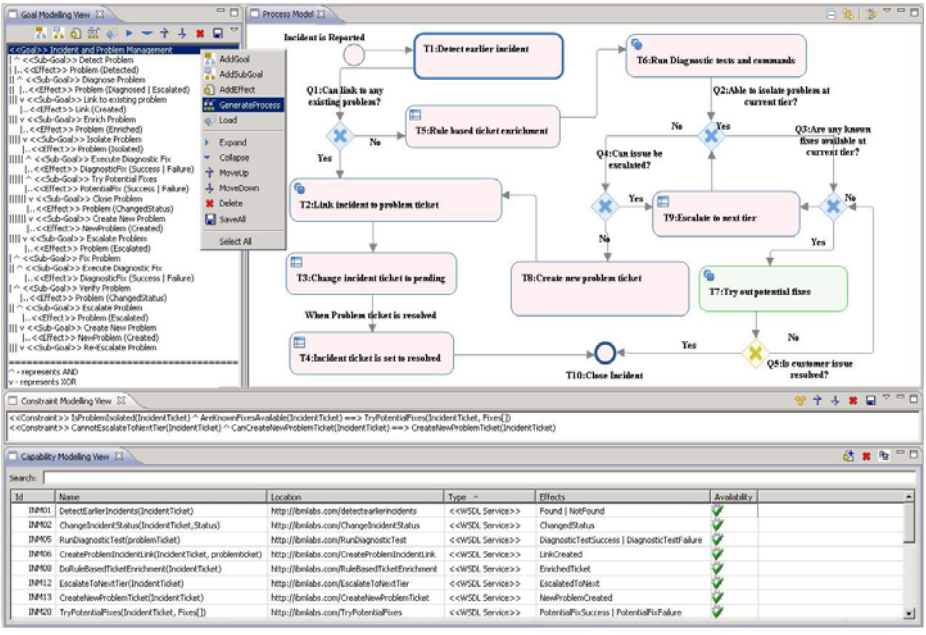


Fig. 2. Prototype Implementation

The citations [14–16] describe techniques for semantic annoations of business processes using mining techniques, with applications such as adaptation and token analysis to identify components in business processes. While undoubtedly powerful, these techniques lack the simplicity of our semantic annotation approach. However, we will be investigating the adaptation application from [15] for future work.

Goal Modeling and Decomposition: The primary goal decomposition methodology that we have leveraged in this paper is KAOS [9], which provides a language and method for goal-driven requirements elaboration.

Business Process Compliance: Business process compliance management [2, 3] involves several aspects, of which the following are relevant for this paper, viz., verifying process compliance against compliance requirements, and business process derivation to ensure adherence to compliance requirements. For the former aspect, various frameworks [2, 11] have been developed to manage and check the violation of compliance policies by a given business process at design time, in order to minimize the cost of non-compliance. The citation [3] presents a semi-automated approach to synthesize business process templates out of compliance requirements expressed in linear temporal logic; however, that paper only focuses on compliance requirements, whereas our approach also considers functional requirements expressed as goals.

8 Future Work

Future work will involve testing our approach on larger case studies, and incorporating automated inconsistency resolution, process adaptation and incremental redesign.

References

1. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: APCCM, pp. 3–12 (2010)
2. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance checking between business processes and business contracts. In: EDOC, pp. 221–232 (2006)
3. Awad, A., Goré, R., Thomson, J., Weidlich, M.: An Iterative Approach for Business Process Template Synthesis from Compliance Rules. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 406–421. Springer, Heidelberg (2011)
4. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall (2009)
5. Henneberger, M., Heinrich, B., Lautenbacher, F., Bauer, B.: Semantic-based planning of process models. In: Multikonferenz Wirtschaftsinformatik (2008)
6. Heinrich, B., Bolsinger, M., Bewernik, M.: Automated planning of process models: The construction of exclusive choices. In: ICIS, paper 184 (2009)
7. Mukherjee, S., Davulcu, H., Kifer, M., Senkul, P., Yang, G.: Logic based approaches to workflow modeling and verification (2003)
8. Hinge, K., Ghose, A.K., Koliadis, G.: Process seer: A tool for semantic effect annotation of business process models. In: EDOC, pp. 54–63 (2009)
9. Darimont, R., van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. SIGSOFT Software Engineering Notes 21, 179–190 (1996)
10. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20(1-2), 3–50 (1993)
11. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
12. Carbonell, J., et al.: Context-based machine translation. In: Proceedings of the 7th Conference of the Association for Machine Translation in the Americas, pp. 19–28 (2006)
13. Narendra, N.: A goal-based and risk-based approach to creating adaptive workflow processes. In: AAAI Spring Symposium on Bringing Knowledge to Business Processes (2000)
14. Lautenbacher, F., Bauer, B., Forg, S.: Process mining for semantic business process modeling. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 45–53 (2009)
15. Lautenbacher, F., Eisenbarth, T., Bauer, B.: Process model adaptation using semantic technologies. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 301–309 (2009)
16. Gotz, M., Roser, S., Lautenbacher, F., Bauer, B.: Token analysis of graph-oriented process models. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 15 –24 (2009)

Defining and Analysing Resource Assignments in Business Processes with RAL*

Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés

Universidad de Sevilla, Spain
{cristinacabanillas,resinas,aruiz}@us.es

Abstract. Business process (BP) modelling notations tend to stray their attention from (human) resource management, unlike other aspects such as control flow or even data flow. They not only offer little intuitive languages to assign resources to BP activities, but neither link BPs with the structure of the organization where they are used, so BP models can easily contain errors such as the assignment of resources that do not belong to the organizational model. In this paper we address this problem and define RAL (Resource Assignment Language), a domain-specific language explicitly developed to assign resources to the activities of a BP model. RAL makes BPs aware of organizational structures. Besides, RAL semantics is based on an OWL-DL ontology, which enables the automatic analysis of resource assignment expressions, thus allowing the extraction of information from the resource assignments, and the detection of inconsistencies and assignment conflicts.

Keywords: resource-aware business process model, RAL, workflow resource pattern, organizational model, OWL, description logics.

1 Introduction

In this paper we face human-resource[1] management in BP models. Specifically, we deal with resource assignment to the activities of a BP, aiming at easing and improving the way resources can be associated with BP activities. Unfortunately, unlike other aspects such as control flow, resources have received much less attention. However, the participation of people in BPs is of utmost importance, both to supervise the execution of automatic activities and to carry out software-aided and/or manual activities, so they should be considered when designing and modelling the BPs used in an organization.

Furthermore, the alignment of the BPs of an organization with its organizational structure enables the automation of work in different directions. On the one hand, it makes it possible to infer interesting information, such as: (i) the potential performers of each BP activity, i.e., the set of people that fulfill the

* Partially supported by the European Commission (FEDER), Spanish Government under project SETI (TIN2009-07366); and projects THEOS (TIC-5906) and ISABEL (P07-TIC-2533) funded by the Andalusian Local Government.

[1] From now on we will use the term *resource* to refer to human resources.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 477–486, 2011.

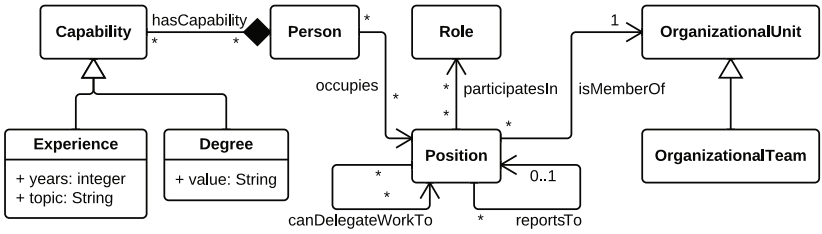


Fig. 1. Excerpt of the organizational metamodel described by Russell et al. [1]

resource-related constraints imposed in the model; or (ii) the potential set of activities each person of an organization can be assigned at runtime. This kind of information may be beneficial for an organization in several ways. For instance, in the previous case: the former benefits the person in charge of resource allocation, and the latter provides an employee-oriented vision, informing about the possible workload of each employee and, hence, allowing reacting in time to avoid having people overburderned with work. On the other hand, it enables the detection of inconsistencies between the resource assignments associated to activities of a BP and the structure of the organization where it is used, e.g. non-existent roles or persons.

The main contribution of this paper is the description and formalization of RAL (Resource Assignment Language), a DSL (Domain Specific Language) to express resource assignments in the activities of a BP in terms of the concepts used in the organizational metamodel proposed by Russell et al. [1]. This formal description is provided by means of a semantic mapping between RAL and description logics (DLs), which is a logical formalism widely used by the semantic web community. A semantic mapping is a way to provide semantics to a model, RAL, by mapping its concepts into a target domain whose semantics has been formally defined [2]. An important advantage of our approach is that one can capitalize on existing efficient DLs algorithms for inferring the aforementioned interesting information from RAL expressions, instead of having to invent a corresponding ad-hoc algorithm for each problem. Furthermore, a prototype has been developed to show the use of RAL and the benefits of its DL-based semantics.

After introducing RAL is Section 2, we describe the semantic mapping in Section 3. Then, we detail how we can leverage DLs to analyse resource assignments in Section 4. Related work can be found in Section 5, and a set of conclusions and future work are discussed in Section 6.

2 Introduction to RAL. Definition and Application

RAL is a DSL that allows the assignment of resources to BP activities in terms of the concepts used in organizational models such as persons, roles, positions, capabilities, or organizational units. Specifically, the concepts used in RAL (cf. Figure 1) are a subset of those included in the organizational metamodel described by Russell et al. [1]. This metamodel was used by the authors as basis to

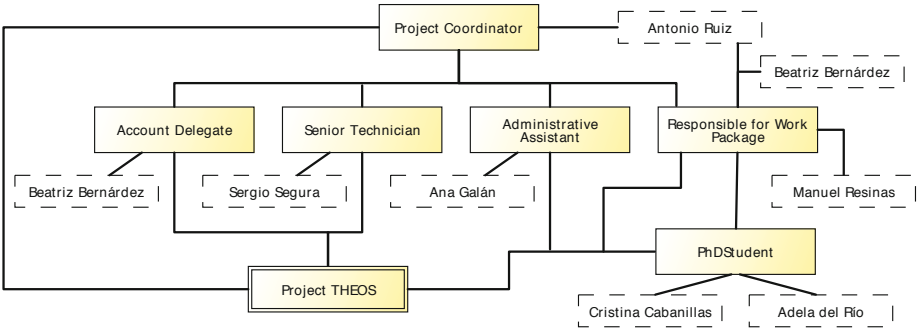
Language 1. RAL's EBNF definition

```
Expression := IS PersonConstraint
    | HAS GroupResourceType GroupResourceConstraint
    | SHARES Amount GroupResourceType WITH PersonConstraint
    | HAS CAPABILITY CapabilityConstraint
    | IS ASSIGNMENT IN ACTIVITY activityName
    | RelationshipExpression
    | CompoundExpression

RelationshipExpression := ReportExpression
          | DelegateExpression

ReportExpression := REPORTS TO PositionConstraint Depth
         | IS Depth REPORTED BY PositionConstraint

DelegateExpression := CAN DELEGATE WORK TO PositionConstraint
          | CAN HAVE WORK DELEGATED BY PositionConstraint

CompoundExpression := NOT (Expression)
           | (Expression) OR (Expression)
           | (Expression) AND (Expression)
           | (Expression) AND IF POSSIBLE (Expression)

PersonConstraint := personName
        | PERSON IN DATA FIELD dataObject.fieldName
        | PERSON WHO DID ACTIVITY activityName

GroupResourceConstraint := groupResourceName
            | IN DATA FIELD dataObject.fieldName

CapabilityConstraint := capabilityName
             | CapabilityRestriction

PositionConstraint := POSITION namePosition
           | POSITION OF PersonConstraint

Amount := SOME          GroupResourceType := POSITION
    | ALL                              | ROLE
                                | UNIT
Depth := DIRECTLY
       | λ
```

define a set of *workflow resource patterns*. These patterns have already been used by other authors as framework to extend BPMN regarding resource management [3], so we believe it is reasonable to use the same metamodel.

Building on this metamodel, RAL allows formulating expressions that define who can perform an activity in the BP. The concrete syntax of RAL is specified in Language 1, whereas its abstract representation can be found at `http://www.isa.us.es/cristal`. In short, RAL allows expressing that an activity has to be performed by, e.g.: *a*) a concrete person, the person who did another activity, or the person indicated in a data field; *b*) someone with a specific group resource[2]; *c*) a person that has a group resource in common with other person; *d*) someone with certain capability; and *e*) someone that reports to or can delegate work to a given position. The language also allows stating that an activity has the same RAL expression as another activity (no matter which

[2] We use the term *group resource* when referring to concepts that represent groups of persons, i.e., positions, roles and organizational units.



Position	Role
Project Coordinator	Responsible
	Account Administrator
	Resource Manager
	Doctoral Thesis Advisor
	Researcher
Responsible for Work Package	Responsible
	Researcher
	Doctoral Thesis Advisor

Position	Role
PhD Student	Research Assistant
Senior Technician	Responsible
Account Delegate	Account Administrator
Administrative Assistant	Clerk

Fig. 2. Excerpt of ISA group's organizational model from a project perspective

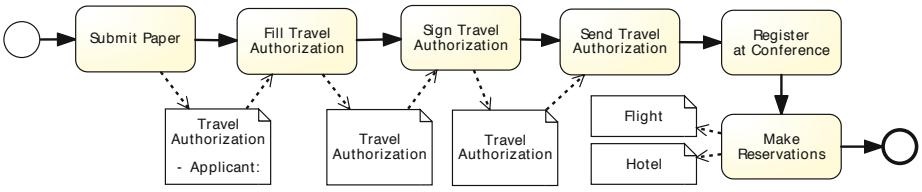
it is), and formulating negative and compound assignments with conjunctions `NOT`, `AND`, `OR` and `AND IF POSSIBLE`. The last one helps indicate preferences. For a more detailed description, we refer the reader to [4].

Figure 2 depicts a possible instantiation of the organizational metamodel shown in Figure 1. This instance is an excerpt of the *ISA Research Group* of the University of Seville from a research project perspective. There are six positions that are members of one organizational unit (Project THEOS), and seven persons occupying these positions. Each position of the model can delegate work to any inferior position and reports work to its immediately upper position. Relationship *participatesIn* is summarized in a table. A table with relationship *hasCapability* is also required, but it is omitted here for space limitations.

Based on that organizational model, one can use RAL to assign resources to the activities of a BP. For instance, Figure 3 shows resource assignments for some activities of the example BP, along with the corresponding DL queries, which will be explained in Section 3.

3 RAL Semantics

In this section, we provide a precise definition of RAL by means of a *semantic mapping* into DLs. Knowledge representation systems based on DLs consist of two components: TBox and ABox. The TBox describes terminology, i.e., the ontology in the form of *concepts* and *roles* (relations between the concepts) definitions and their relationships, while the ABox contains *assertions* about individuals (instances of concepts) using the terms from the ontology (see [5] for more details about DLs and their syntax). We have chosen DLs because of



Submit Paper: Only *Researchers* and *Research Assistants* are authorized to execute this task, and they must have a degree.
RAL: `((HAS ROLE Researcher) OR (HAS ROLE ResearchAssistant)) AND (HAS CAPABILITY Degree)`
DL: $AssignmentSubmitPaper \equiv ((\exists occupies.(\exists participatesIn.\{Researcher\})) \sqcup (\exists occupies.(\exists participatesIn.\{ResearchAssistant\}))) \sqcap (\exists hasCapability.\{Degree\})$

Sign Travel Authorization: This task must be undertaken by someone that is reported by (the position of) the person that undertook task *Submit Paper*.
RAL: `(IS REPORTED BY POSITION OF PERSON WHO DID ACTIVITY SubmitPaper)`
DL: $AssignmentSignTravelAuth \equiv \exists occupies.(\exists isExtendedReportedBy.(\exists isOccupiedBy.\{AssignmentSubmit\}))$

Make Reservations: Antonio cannot execute this task but the performer must either have some role in common with Antonio.
RAL: `(NOT (IS Antonio)) AND (SHARES SOME ROLE WITH Antonio)`
DL: $AssignmentMakeReservations \equiv (\neg\{Antonio\}) \sqcap (\exists occupies.(\exists participatesIn.(\exists developedIn.(\exists isOccupiedBy.\{Antonio\}))))$

Fig. 3. Resource assignments to activities of a process for Conference Travel

two reasons. First, DLs provide a very natural way to describe an organizational structure. Second, there is a plethora of DLs reasoners that can be used to automatically analyse RAL expressions efficiently and, hence, to automatically infer information from them.

Note that, since RAL builds on an organizational metamodel, it is necessary to provide a mapping not only to RAL expressions but also to the organizational metamodel and its instances. Next, we detail all those mappings. A full version can be found at `http://www.isa.us.es/cristal`, in which we use the W3C recommendation OWL 2 [6] to define the ontologies[3].

3.1 Mapping the Organizational Structure into DLs

Mapping the organizational metamodel into an ontology: It is quite straightforward since the elements used in both domains are quite similar. Each class of the metamodel is mapped to one *concept* and the hierarchies are mapped using the *subclassOf* relationship. The remaining relationships (i.e., *hasCapability*, *occupies*, *canDelegateWorkTo*, *reportsTo*, *participatesIn* and *isMemberOf*)

[3] Sometimes OWL terms *classes*, *properties* and *objects* will be used to refer to DL terms *concepts*, *roles* and *individuals*, respectively.

are mapped into *properties* together with their corresponding *cardinality restrictions*. In addition, properties *isOccupiedBy*, *isReportedBy*, *developedIn* and *formedBy* have been defined as the inverse properties of *occupies*, *reportsTo*, *participatesIn* and *isMemberOf*, respectively, to make it easier the formulation of some RAL expressions. Properties *hasDegree* and *hasExperience* have been added to represent the existing specific capabilities. Furthermore, in order to be able to transitively refer to upper positions in the organizational model, a *transitive* super-property *extendedReportsTo* has been added. For the same reason, property *extendedCanDelegateWorkTo* has been added as well.

Instantiating the Organizational Ontology: The structure of a concrete organization, such as that in Figure 2, is mapped as *individual assertions* in the ABox (e.g., $Role(Researcher)$) and the relationships between the individuals are stated as *property assertions* (e.g., $participatesIn(ProjectCoordinator, Researcher)$). Besides, an additional individual assertion has been made for each individual to state that each individual has exactly the properties stated and no more (e.g. Position *Project Coordinator* has exactly five *participatesIn* relationships: $(= 5\ participatesIn)(ProjectCoordinator)$). This is a technical detail that is necessary to be able to express the negation included in RAL because of the open world assumption of DLs. The *open world assumption* means that DLs assume that the knowledge may be incomplete and, hence, the absence of a property assertion stating that $participatesIn(ProjectCoordinator, Clerk)$ does not mean that a *Project Coordinator* does not have role *Clerk*.

3.2 Mapping RAL Expressions into DLs

Each RAL expression can be seen as a definition of a subset of all the people in the organization who can do an activity, e.g. a RAL expression stating that certain activity can only be done by someone occupying position *Project Coordinator* reduces the set of potential owners to the persons that occupy that position. In terms of DL, a RAL expression can be seen as a new concept that characterises the individuals that belongs to it amongst all the individuals of type *Person* that there are in the ABox. Therefore, the concept that defines the resource assignment of a certain BP activity a whose RAL expression is $expr_a$ is defined as: $AssignmentA \equiv map(expr_a)$, where $map(expr)$ is a mapping from a RAL expresion into DL as summarised in Table 1. Figure 3 shows the DL queries for some activities of the BP model.

PersonConstraints provide ways to refer to a concrete person. However, in the last two cases the concrete person is unknown until runtime, in which case an approximation is made. The approximation is either all persons in the organization, in case the concrete person is defined in a data field, because we cannot figure out who might be; or all the persons who can do a certain activity in the BP, in case the concrete person is defined as the person who did that activity.

IS PersonConstraint is defined as the `PersonConstraint` mapping it uses.

Table 1. Mapping of some RAL expressions into DLs *concepts*

PersonConstraints (pConst)	**DL Mapping** ($map_p(pConst)$)
personName	$\{personName\}$
PERSON IN DATA FIELD d.field	$Person$
PERSON WHO DID ACTIVITY name	$AssignmentActivityName$
RAL expression (expr)	**DL Mapping** ($map(expr)$)
IS pConst	$map_p(pConst)$
HAS POSITION posName	$\exists occupies.\{posName\}$
HAS ROLE roleName	$\exists occupies.(\exists participatesIn.\{roleName\})$
HAS UNIT unitName	$\exists occupies.(\exists isMemberOf.\{unitName\})$
HAS GroupResourceType IN DATA FIELD d.field	$Person$
SHARES SOME POSIT WITH pConst	$\exists occupies.(\exists isOccupiedBy.map_p(pConst))$
SHARES SOME ROLE WITH pConst	$\exists occupies.(\exists participatesIn.(\exists developedIn.(\exists isOccupiedBy.map_p(pConst))$
SHARES ALL UNIT WITH pConst	$\exists occupies.(\exists isMemberOf.(\forall formedBy.(\exists isOccupiedBy.map_p(pConst))$
HAS CAPABILITY name	$\exists hasCapability.\{name\}$
HAS CAPABILITY name.attr=val	$\exists hasCapability.(name \sqcap \forall attr.\{val\})$
IS ASSIGNMENT IN ACTIVITY name	$AssignmentActivityName$
REPORTS TO POSITION posName	$\exists occupies.(\exists extendedReportsTo.\{posName\})$
REPORTS TO (POSITION OF) pConst	$\exists occupies.(\exists extendedReportsTo.(\exists isOccupiedBy.map_p(pConst))$
(expr1) AND (expr2)	$map(expr1) \sqcap map(expr2)$
(expr1) OR (expr2)	$map(expr1) \sqcup map(expr2)$
NOT (expr)	$\neg map(expr)$
(expr1) AND IF POSSIBLE (expr2)	$map(expr1)$

HAS GroupResourceType GroupResourceConstraint is defined either as the persons that occupy a given position, or as the persons that occupy a given position that *participatesIn* or *isMemberOf* a certain *roleName* or *unitName*, respectively. When the specific resource name is given in a data field, the mapping is generalized to any person.

SHARES Amount GroupResourceType WITH PersonConstraint assigns persons that share some or all positions, roles or organizational units with the given person. Expressions with group resource types `ROLE` or `UNIT` apply the same idea but changing it accordingly for each group resource type.

HAS CAPABILITY CapabilityConstraint is defined as those persons who have the given capability and/or persons who have a capability with certain value in some of its attributes. Table 1 shows the case of *equal* operator. Other operator could be used provided it can be mapped to DLs.

IS ASSIGNMENT IN ACTIVITY activityName is defined by making it equivalent to the concept defined for the assignment of the given activity.

Table 2. Some possible analyses of RAL expressions

Question	DL operations
Who are the people that can do activity A?	*individuals(AssignmentA)*
Who are the activities that can do person p?	*realization(p)* and, then, select all those concepts that are assignments
Is there any person that can do all of the activities of the BP?	*individuals(AssignmentA* $\sqcap$ ... $\sqcap$ *AssignmentX)*, where *AssignmentA ... AssignmentX* are all of the assignments of the BP
Are the people that can do activity B a subset of those that can do activity A?	*subsumes(A,B)*
Can the same people do activities A and B?	*subsumes(A,B)* $\wedge$ *subsumes(B,A)*

REPORTS TO PositionConstraint Depth is defined as the persons who occupy a position that has a *reportsTo* or *extendedReportsTo* relationship with a given position name depending on whether it is `DIRECTLY` reported or not, respectively. Also, the positions of a given person can be used instead of a concrete position name. The other *relationship expressions* are like this one, but changing the property accordingly, e.g., changing *extendedReportsTo* for *extendedCanDelegateWorkTo*. In *delegate expressions* no direct delegations are allowed.

CompoundExpression has a quite direct mapping except for expressions `AND IF POSSIBLE` and `NOT`. The former expresses a preference for allocation, but it is not mandatory. Thus, in order to ensure the actual potential owner of the activity is within the result, the right side of the expression is ignored in the mapping. The latter must be generalized to any person if *expr* contains runtime information, e.g. the person who did an activity.

4 DL-Based Analysis of Resource Assignments

The definition of the semantics of RAL expressions in terms of DLs makes it possible to automate their analysis by means of a DL reasoner. DL reasoners are software tools that implement several operations on the ontologies in an efficient manner by using several heuristics and techniques. Some of these operations are:

- $satisfiability(C)$: Determine whether concept C is not contradictory.
- $subsumes(A, B)$: Determine whether concept A subsumes concept B, i.e., whether description of A is more general than description of B.
- $individuals(C)$: Find all individuals that are instances of concept C.
- $realization(i)$: Find all concepts which the individual i belongs to.

By using these operations, we can analyse the assignment of resources made to a BP model in order to extract information from it and answer questions such as "*Who are the activities that can do person P?*" Table 2 depicts some of these questions and how they can be written on the basis of DL operations. These operations allow us to detect problem situations, such as that in which there is an activity that cannot be allocated to any person given the RAL expression of the activity and the organizational model of the company.

5 Related Work

The need of including organizational aspects in BP design can be seen in [7], where Künzle et al. present a set of challenges that should be addressed to make BPs both data-aware and resource-aware. Bertino et al. have defined a language to express constraints in role-based and user-based assignments to the tasks of a WF [8]. They get to check whether the configured assignments are possible at runtime and to plan possible resource allocation based on the assignments. They consider also dynamic aspects for these checkings. However, the language is more complex and hard to use than RAL because its goal is wider.

Russell et al. defined the *workflow resource patterns* with the aim of explaining the requirements for resource management in workflow (WF) environments [1]. They analysed the support provided by some WF tools, but they did not provide a specific way to assign resources to WF activities. These patterns were used by other authors as a reference framework to analyse the ability of BPMN to deal with resources and to propose solutions to improve BPMN [3]. The *creation patterns* have been used to assess the expressiveness of RAL in [4].

Strembeck et al. presented a formal metamodel for process-related role-based access control models and they defined a runtime engine to enforce the different policies and constraints in a software system [9]. However, the resource assignments that can be made with their metamodel is less expressive than RAL. Besides, they have to use ad-hoc algorithms instead of reusing those already implemented for DLs. An optimal approach to allocate the most proficient set of employees for a whole BP from event logs based on Hidden Markov Models is introduced in [10] and Nakatumba et al. proposed a way to analyse and characterise resource behaviour after BP execution from event logs using process mining [11].

Finally, some extensions to enhance resource management in BP execution environments have been recently released, e.g., BPEL4People and WS-HumanTask are two extension proposals for BPEL (both can be found at `http://www.oasis-open.org/committees/bpel4people/`). However, there is limited support to express and manage resource allocation on higher level modelling languages such as BPMN.

6 Conclusions and Future Work

The result of this work lets us conclude that defining and automatically analysing new languages to describe resource assignments in BP models is possible. RAL, our proposal, allows not only precisely defining the assignments required to cover most of the *creation patterns* proposed by Russell et al. [1] and more expressive assignments, but also automatically reasoning about the resource assignments configured. To this end, RAL semantics has been described in an OWL-DL ontology and we have shown how DL reasoners can be used to extract information from them. However, it is important to notice that RAL currently addresses only expressions involving a *single instance* of a BP, i.e., the history of individual resources and past executions are not considered for now.

We have developed a prototype that analyses RAL expressions associated to BPMN activities and returns the potential owners of a selected activity. It has

been implemented as a plugin for Oryx [12] and it can be tested following the instructions described at `http://www.isa.us.es/cristal`.

We believe the present work settles the basis towards the spread of the use of resource assignments in BP models, something we consider vital to be able to incorporate business environments (organizations) currently limited, due to their inability to link the organizational structure with BPs, in a efficient and standardized way, and extracting information from them.

In the near future we intend to refine the mapping to obtain more precise information about the potential owners of the activities of a process, according to the execution state of a process instance. In addition, we plan to develop a visual notation for RAL and to define a specific catalogue of analysis operations for the extraction of interesting resource-related information.

References

1. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005)
2. Rivera, J.E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing rule-based behavioral semantics of visual modeling languages with maude. In: Gašević, D., Lämmel, R., Van Wyk, E. (eds.) SLE 2008. LNCS, vol. 5452, pp. 54–73. Springer, Heidelberg (2009)
3. Awad, A., Grosskopf, A., Meyer, A., Weske, M.: Enabling resource assignment constraints in BPMN, tech. rep., BPT at Hasso Plattner Institut (2009)
4. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: RAL: A high-level user-oriented resource assignment language for business processes. In: BPM Workshops, BPD 2011 (in press, 2011)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logics Handbook: Theory, Implementations, and Applications. Cambridge University Press (2003)
6. Motik, B., Patel-Schneider, P.F., Grau, B.C.: OWL 2 Web Ontology Language Direct Semantics (2009)
7. Künzle, V., Reichert, M.: Integrating Users in Object-Aware Process Management Systems: Issues and Challenges. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 29–41. Springer, Heidelberg (2010), `http://www.informatik.uni-trier.de/~ley/db/conf/bpm/bpmw2009.html`
8. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. 2, 65–104 (1999)
9. Strembeck, M., Mendling, J.: Modeling process-related RBAC models with extended UML activity models. Inf. Softw. Technol. 53(5), 456–483 (2011)
10. Yang, H., Wang, C., Liu, Y., Wang, J.: An Optimal Approach for Workflow Staff Assignment Based on Hidden Markov Models. In: Meersman, R., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 24–26. Springer, Heidelberg (2008)
11. Nakatumba, J., van der Aalst, W.M.P.: Analyzing Resource Behavior Using Process Mining. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 69–80. Springer, Heidelberg (2010), `http://www.informatik.uni-trier.de/~ley/db/conf/bpm/bpmw2009.html`
12. Decker, G., Overdick, H., Weske, M.: Oryx - An Open Modeling Platform for the BPM Community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008)

Stochastic Optimization for Adaptive Labor Staffing in Service Systems

L.A. Prashanth[1], H.L. Prasad[1], Nirmit Desai[2],
Shalabh Bhatnagar[1], and Gargi Dasgupta[2]

[1] Indian Institute of Science, Bangalore, India
{prashanth,hlprasu,shalabh}@csa.iisc.ernet.in
[2] IBM Research, Bangalore, India
{nirmdesa,gaargidasgupta}@in.ibm.com

Abstract. Service systems are labor intensive. Further, the workload tends to vary greatly with time. Adapting the staffing levels to the workloads in such systems is nontrivial due to a large number of parameters and operational variations, but crucial for business objectives such as minimal labor inventory. One of the central challenges is to optimize the staffing while maintaining system steady-state and compliance to aggregate SLA constraints. We formulate this problem as a parametrized constrained Markov process and propose a novel stochastic optimization algorithm for solving it. Our algorithm is a multi-timescale stochastic approximation scheme that incorporates a SPSA based algorithm for 'primal descent' and couples it with a 'dual ascent' scheme for the Lagrange multipliers. We validate this optimization scheme on five real-life service systems and compare it with a state-of-the-art optimization tool-kit OptQuest. Being two orders of magnitude faster than OptQuest, our scheme is particularly suitable for adaptive labor staffing. Also, we observe that it guarantees convergence and finds better solutions than OptQuest in many cases.

Keywords: Service systems, labor optimization, constrained stochastic optimization.

1 Introduction

In service-based economies, the clients and service providers exchange value through service interactions and reach service outcomes. Service requests of a client can vary greatly in the skills required to fulfill the request, expected turn-around time, and the context of the client's business needs. As a result, service delivery is a labor-intensive business and it is crucial to optimize labor costs. A *Service System (SS)* is an organization composed of (i) the resources that support, and (ii) the processes that drive service interactions so that the outcomes meet customer expectations [1]. The contributions of this paper are focused on data-center management services but can be extended to all service domains. The service providers manage the data-centers from remote locations called *delivery centers* where groups of *service workers (SW)* skilled in specific technology areas support corresponding *service requests (SR)*. In each group, the processes, the people and the customers that drive the operation of center constitute a SS. A delivery center is a system of multiple SS. A central component in these operational models

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 487–494, 2011.

is the policy for assigning SRs to SWs, called the *dispatching policy*. The fundamental challenges here are: (i) given an SS with its operational characteristics, the staffing across skill levels and shifts needs to be optimized while maintaining steady-state and compliance to Service Level Agreement (SLA) constraints. (ii) the SS characteristics such as work patterns, technologies and customers supported change frequently, and hence the optimization needs to be adaptive.

This paper presents a novel stochastic optimization algorithm *SASOC (Staff Allocation using Stochastic Optimization with Constraints)* to address the above challenges. SASOC is a three timescale stochastic approximation scheme that uses SPSA-based [2] estimates for performing gradient descent in the primal, while having a dual ascent scheme for the Lagrange multipliers. In the evaluation step of SASOC, we leverage the simulation-based operational models developed in [3] for two of the dispatching policies, namely, PRIO-PULL (basic priority scheme) and EDF (earliest deadline first). We evaluate our algorithm on data from five real-life SS in the data-center management domain. In comparison with the state-of-the-art OptQuest optimization toolkit [4], we find that (a) SASOC is two orders of magnitude faster than OptQuest, (b) it finds solutions of comparable quality to OptQuest, and (c) it guarantees convergence where OptQuest does not find feasibility even with 5000 iterations. Precisely due to the guaranteed convergence and by returning good solutions quickly, SASOC is well suited to better address the above two challenges, especially with respect to adaptivity. By comparing SASOC results on two independent operational models corresponding to PRIO-PULL and EDF dispatching policies, we show that SASOC's performance is independent of the operational model of SS.

We now review relevant literature in service systems and stochastic optimization. In [5], a two step mixed-integer program is formulated for the problem of dispatching SRs within service systems. While their goal is similar, their formulation does not model the stochastic variations of arrivals or processing times. Further, unlike our framework, the SLA constraints in their formulation cannot be aggregates. In [6], the authors propose a scheme for shift-scheduling in the context of third-level IT support systems. Unlike this paper, they do not validate their method against data from real-life third-level IT support. In [3], a simulation framework for evaluating dispatching policies is proposed. A scatter search technique is used to search over the space of SS configurations and optimize the staff there. While we share their simulation model, the goal in this paper is to propose a fundamentally new algorithm that is based on stochastic optimization. In general, none of the above papers propose an optimization algorithm that is geared for SS and that leverages simulation to adapt optimization search parameters.

A popular and highly efficient simulation based local optimization scheme for gradient estimation is Simultaneous Perturbation Stochastic Approximation (SPSA) proposed by [7]. SPSA is based on the idea of randomly perturbing the parameter vector using i.i.d., symmetric, zero-mean random variables that has the critical advantage that it needs only two samples of the objective function for any N-dimensional parameter. Usage of deterministic perturbations instead of randomized was proposed in [2]. The deterministic perturbations there were based either on lexicographic or Hadamard matrix based sequences and were found to perform better than their randomized perturbation counterparts. In [8], several simulation based algorithms for constrained optimization

have been proposed. Two of the algorithms proposed there use SPSA for estimating the gradient, after applying Lagrangian relaxation procedure to the constrained optimization problem.

2 Problem Formulation

We consider the problem of finding the optimal number of workers for each shift and of each skill level in a SS, while adhering to a set of SLA constraints. We formulate this as a constrained optimization problem with the objective of minimizing the labor cost in the long run average sense, with the constraints being on SLA attainments. The underlying dispatching policy (that maps the service requests to the workers) is fixed and is in fact, parametrized by the set of workers. In essence, the problem is to find the 'best' parameter (set of workers) for a given dispatching policy.

In a typical SS, the arrival as well as service time of SRs are probabilistic. Thus, the system can be modeled to be evolving probabilistically over states, where each state transition incurs a cost. The objective is to minimize the long run average sum of this single stage cost, while adhering to a set of SLA constraints. The state is the vector of the utilization of workers for each shift and skill level, and the current SLA attainments for each customer and each SR priority. Any arriving SR has a customer identifier and a priority identifier. Let A be the set of shifts of the workers, B be the set of skill levels of the workers, C be the set of all customers and P be the set of all possible priorities in the SS under consideration. The state X_n at time n is given by

$$X_n = (u_{1,1}(n), ..., u_{|A|,|B|}(n), \gamma'_{1,1}(n), ..., \gamma'_{|C|,|P|}(n), q(n)),$$

where $0 \leq u_{i,j} \leq 1$ is the per-unit utilization of the workers in shift i and skill level j. $0 \leq \gamma'_{i,j} \leq 1$ denotes the SLA attainment level for customer i and priority j. q is a Boolean variable that denotes the queue feasibility status of the system at instant n. In other words, q is false if the growth rate of the SR queues (for each complexity) is beyond a threshold and true otherwise. We need q to ensure system steady-state which is independent of SLA attainments because SLA attainments are computed only on the SRs that were completed and not on SRs queued up in the system. The action a_n at instant n specifies the number of workers of each skill level in each shift.

The single stage cost function is designed so as to minimize the under-utilization of workers as well as over-achievement/under-achievement of SLAs. Here, under-utilization of workers is the complement of utilization and in essence, this is equivalent to maximizing the worker utilizations. The over-achievement/under-achievement of SLAs is the distance between attained SLAs and the contractual SLAs. Hence, the cost function is designed to balance between two conflicting objectives and has the form:

$$c(X_n) = r \times \left(1 - \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \alpha_{i,j} \times u_{i,j}(n)\right) + s \times \left(\sum_{i=1}^{|C|} \sum_{j=1}^{|P|} \left|\gamma'_{i,j}(n) - \gamma_{i,j}\right|\right), \quad (1)$$

where $r, s \geq 0$ and $r + s = 1$. $0 \leq \gamma_{i,j} \leq 1$ denotes the contractual SLA for customer i and priority j. Note that the first factor uses a weighted sum of utilizations over workers

from each shift and across each skill level. The weights $\alpha_{i,j}$ are derived from the workload distribution across shifts and skill levels over a month long period. These weights satisfy $0 \leq \alpha_{i,j} \leq 1, \quad \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \alpha_{i,j} = 1$. This prioritization of workers helps in optimizing the worker set based on the workload.

The constraints are on the SLA attainments and are given by:

$$g_{i,j}(X_n) = \gamma_{i,j} - \gamma'_{i,j}(n) \leq 0, \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|, \tag{2}$$

$$h(X_n) = 1 - q(n) \leq 0, \tag{3}$$

where constraints (2) specify that the attained SLA levels should be equal to or above the contractual SLA levels for each pair of customer and priority. The constraint (3) ensures that the SR queues for each complexity in the system stay bounded.

Considering that the state is a vector of utilization levels and the current SLA attainments, it is easy to see that $\{X_n, n \geq 1\}$ is a constrained Markov process for any given dispatch policy. Further, $\{X_n, n \geq 1\}$ is parametrized with

$$\theta = (W_{1,1}, ..., W_{|A|,|B|})^T \in \mathcal{R}^{|A| \times |B|},$$

where $W_{i,j}$ indicates the number of service workers whose skill level is j and whose shift index is i. We want to find an optimal value for the parameter vector θ that minimizes the long-run average sum of single-stage cost $c(X_n)$ while maintaining queue stability $h(X_n)$ and compliance to contractual SLAs $g_{i,j}(X_n)$, $\forall i = 1, \ldots, |C|, j = 1, \ldots, |P|$.

We let the parameter vector $\theta \in \mathcal{R}^{|A| \times |B|}$ take values in a compact set $M \stackrel{\triangle}{=} [0, W_{\max}]^{|A| \times |B|}$ through the projection operator Π defined by $\Pi(\theta) \stackrel{\triangle}{=} (\pi(W_{1,1}), \ldots, \pi(W_{|A|,|B|}))^T$, $\theta \in \mathcal{R}^{|A| \times |B|}$. Here $\pi(x) \stackrel{\triangle}{=} \min(\max(0, x), W_{\max})$. In essence, the projection operator Π keeps each $W_{i,j}$ bounded between 0 and $W_{\max}$ and this is necessary for ensuring the convergence of θ. Our aim is to find a θ that minimizes the long run average cost,

$$\begin{aligned} &J(\theta) \stackrel{\triangle}{=} \lim_{n \to \infty} \tfrac{1}{n} \sum_{m=0}^{n-1} E[c(X_m)] \\ &\text{subject to} \\ &G_{i,j}(\theta) \stackrel{\triangle}{=} \lim_{n \to \infty} \tfrac{1}{n} \sum_{m=0}^{n-1} E[g_{i,j}(X_m)] \leq 0 \quad \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|, \\ &H(\theta) \stackrel{\triangle}{=} \lim_{n \to \infty} \tfrac{1}{n} \sum_{m=0}^{n-1} E[h(X_m)] \leq 0 \end{aligned} \tag{4}$$

Here each step from n to $n+1$ indicates a state transition from X_n to X_{n+1}, incurring a cost $c(X_n)$. The parameter θ decides what cost is incurred and whether the constraints are met. The actions a_j are assumed to be governed by the underlying dispatching policy of the SS. We make a standard assumption that the Markov process $\{X_n, n \geq 1\}$ is ergodic for the given dispatching policy, which is true in general. Thus, the limits in the above optimization problem are well defined. While it is desirable to find the optimum $\theta^* \in M$ i.e.,

$$\theta^* = \text{argmin} \left\{ J(\theta) \text{ s.t. } \theta \in M, G_{i,j}(\theta) \leq 0, i = 1, \ldots, |C|, j = 1, \ldots, |P|, H(\theta) \leq 0 \right\},$$

it is in general very difficult to achieve a global minimum. We apply the Lagrangian relaxation procedure to the above problem and then use a local optimization scheme based on SPSA for finding the optimum parameter θ^*.

3 Our Algorithm (SASOC)

The constrained optimization problem (4) can be expressed using the standard Lagrange multiplier theory as an unconstrained optimization problem given below.

$$\max_{\lambda} \min_{\theta} L(\theta, \lambda) \stackrel{\triangle}{=} \lim_{n \to \infty} \frac{1}{n} \sum_{m=0}^{n-1} E \left\{ c(X_m) + \sum_{i=1}^{|C|} \sum_{j=1}^{|P|} \lambda_{i,j} g_{i,j}(X_m) + \lambda_f h(X_m) \right\} \tag{5}$$

where $\lambda_{i,j} \geq 0, \quad \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|$ represent the Lagrange multipliers corresponding to constraints $g_{i,j}(\cdot)$ and λ_f represents the Lagrange multiplier for the constraint $h(\cdot)$, in the optimization problem (4). The function $L(\theta, \lambda)$ is commonly referred to as the Lagrangian. An optimal (θ^*, λ^*) is a saddle point in the Lagrangian i.e. $L(\theta, \lambda^*) \geq L(\theta^*, \lambda^*) \geq L(\theta^*, \lambda)$. Thus, it is necessary to design an algorithm which descends in θ and ascends in λ to find the optimum point. The simplest iterative procedure for this purpose would use the gradient of the Lagrangian with respect to θ and λ to descend and ascend respectively. However, for the given system the computation of gradient with respect to θ would be intractable due to lack of a closed form expression of the Lagrangian. Thus, a simulation based algorithm is required. We employ an SPSA technique [7,2] for obtaining a stochastic approximation descent procedure for θ. For $\lambda_{i,j}$ and λ_f, values of $g_{i,j}(\cdot)$ and $h(\cdot)$ respectively can be seen to provide a stochastic ascent direction.

The above explanation suggests that an algorithm would need three stages in each of its iterations. (i) The inner-most stage which performs one or more simulations over several time steps; (ii) The next outer stage which computes a gradient estimate using simulation results of the inner most stage and then updates θ along descent direction. This stage would perform several iterations for a given λ and find out the best θ; and (iii) The outer-most stage which computes the long-run average value of each constraint using the iterations in the inner two stages and updates the Lagrange multipliers λ for using that as the ascent direction. The above three steps need to be performed iteratively till the solution converges to a saddle point described previously. However, this approach suffers from a serious drawback of requiring to perform several simulations as a whole as one outer stage update happens for one full run of inner stages at both levels. This issue gets addressed by using simultaneous updates to all three stages but with different time-steps, the outer-most having the smallest while the inner-most having the largest time-steps. This comes under the realm of multiple time-scale stochastic approximation [9, Chapter 6]. We develop a three time-scale stochastic approximation algorithm that does primal descent using an SPSA based actor-critic algorithm while performing dual ascent on the Lagrange multipliers. The update rule for SASOC is given below:

$$
\left.
\begin{aligned}
&W_{i,j}(n+1) = \Pi\left(W_{i,j}(n) + b(n)\left(\frac{\bar{L}(nK)-\bar{L}'(nK)}{\delta\triangle_{i,j}(n)}\right)\right), \\
&\qquad\qquad \forall i = 1,2,\ldots,|A|, j = 1,2,\ldots,|B|, \\
&\text{where for } m = 0,1,\ldots,K-1, \\
&\bar{L}(nK+m+1) = \bar{L}(nK+m)+ \\
&d(n)(c(X_{nK+m}) + \sum_{i=1}^{|C|}\sum_{j=1}^{|P|}\lambda_{i,j}(nK)g_{i,j}(X_{nK+m}) + \lambda_f h(X_{nK+m}) - \bar{L}(nK+m)), \\
&\bar{L}'(nK+m+1) = \bar{L}'(nK+m)+ \\
&d(n)(c(\hat{X}_{nK+m}) + \sum_{i=1}^{|C|}\sum_{j=1}^{|P|}\lambda_{i,j}(nK)g_{i,j}(\hat{X}_{nK+m}) + \lambda_f h(\hat{X}_{nK+m}) - \bar{L}'(nK+m)), \\
&\lambda_{i,j}(n+1) = (\lambda_{i,j}(n) + a(n)g_{i,j}(X_n))^+, \forall i = 1,2,\ldots,|C|, j = 1,2,\ldots,|P|, \\
&\lambda_f(n+1) = (\lambda_f(n) + a(n)h(X_n))^+,
\end{aligned}
\right\} \quad (6)
$$

where

- $\hat{X}_m$ represents the state at iteration m from the simulation run with perturbed parameter $\theta_{[\frac{n}{K}]} + \delta\Delta_{[\frac{n}{K}]}$. Here $[\frac{n}{K}]$ denotes the integer portion of $\frac{n}{K}$. For simplicity, hereafter we use $\theta + \delta\Delta$ to denote $\theta_{[\frac{n}{K}]} + \delta\Delta_{[\frac{n}{K}]}$;
- $\delta > 0$ is a fixed perturbation control parameter while Δ represents a deterministic perturbation sequence chosen according to an associated Hadamard matrix, explained later in this section;
- The operator $\Pi(\cdot)$ ensures that the updated value for θ stays within the chosen compact space T;
- $\bar{L}$ and $\bar{L}'$ represent Lagrangian estimates for θ and $\theta + \delta\Delta$ respectively. Thus, for each iteration two simulations are carried out, one with θ parameter and the other with the perturbed parameter $\theta + \delta\Delta$, the result of which is used to update $\bar{L}$ and $\bar{L}'$; and
- $K \geq 1$ is a fixed parameter which controls the rate of update of θ in relation to that of $\bar{L}$ and $\bar{L}'$. This parameter allows for accumulation of updates to $\bar{L}$ and $\bar{L}'$ for K iterations in between two successive θ updates.

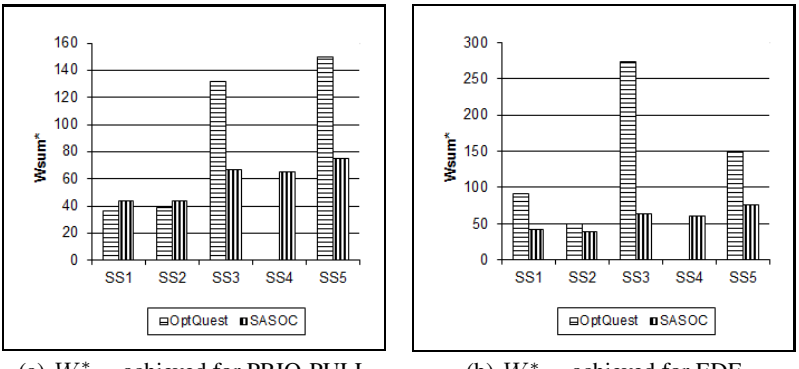
The step-sizes $\{a(n)\}$, $\{b(n)\}$ and $\{d(n)\}$ satisfy $\sum_n a(n) = \sum_n b(n) = \sum_n d(n) = \infty; \sum_n (a^2(n) + b^2(n) + d^2(n)) < \infty, \frac{b(n)}{d(n)}, \frac{a(n)}{b(n)} \rightarrow 0$ as $n \rightarrow \infty$.. The above choice of step-size ensure separation of time-scales between the recursions of $W_{i,j}, \bar{L}, \bar{L}'$ and λ. The perturbation sequence $\{\triangle(n)\}$ is constructed using Hadamard matrices and the reader is referred to Lemma 3.3 of [2] for details of the construction.

4 Simulation Experiments

We use the simulation framework developed in [3] and focus on the PRIO-PULL and EDF dispatching policies. However, SASOC algorithm is agnostic to the dispatching policies. In PRIO-PULL policy, SRs are queued in the complexity queues based directly

on the priority assigned to them by the customers. On the other hand, in the EDF policy the time left to SLA target deadline is used to assign the SRs to the SWs i.e., the SW works on the SR that has the earliest deadline. We implemented our SASOC algorithm as well as an algorithm for staff allocation using the state-of-the-art optimization toolkit OptQuest. OptQuest is a well-established tool for solving simulation optimization problems [4].

For the SASOC algorithm, we set the weights in the single-stage cost function $c(X_m)$, see (1), as $r = s = 0.5$. We thus give equal weightage to both the worker utilization and the SLA over-achievement components. The feasibility Boolean variable q used in the constraint (3) was set to false (i.e., infeasible) if the queues were found to grow by 1000% over a two-week period. On each SS, we compare our SASOC algorithm with the OptQuest algorithm using W_{sum} as the performance metric. Here $W_{sum} \triangleq \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} W_{i,j}$ is the sum of workers across shifts and skill levels. We observe that simulation run-times are proportional to the number of SS simulations and hence, an order of magnitude higher for OptQuest as compared to SASOC.



(a) W^*_{sum} achieved for PRIO-PULL (b) W^*_{sum} achieved for EDF

Fig. 1. Performance of OptQuest and SASOC for two different dispatching policies on five real SS (*Note:* OptQuest is infeasible over SS4)

Figs 1(a) and 1(b) compare the W^*_{sum} achieved for OptQuest and SASOC algorithms using PRIO-PULL on five real life SS. Here W^*_{sum} denotes the value obtained upon convergence of W_{sum}. On three SS pools, namely SS3, SS4 and SS5, respectively, we observe that our SASOC algorithm finds a significantly better value of W^*_{sum} as compared to OptQuest. Note in particular that the performance difference between SASOC, and OptQuest on SS3 and SS5 is nearly 100%. Further, on SS4, OptQuest is seen to be infeasible whereas SASOC obtains a feasible good allocation. On the other two pools, SS1 and SS2, OptQuest is seen to be slightly better than SASOC. Further, the SASOC algorithm requires 500 iterations, with each iteration having 20 replications of the SS - 10 each with unperturbed parameter θ and perturbed parameter $\theta + \delta\Delta$ respectively, whereas OptQuest requires 5000 iterations with each iteration of 100 replications. This implies a two orders of magnitude improvement while searching for the optimal SS configuration in SASOC as compared to OptQuest. Fig 1(b) presents similar results for

the case of EDF dispatching policy. The behavior of OptQuest and SASOC algorithms was found to be similar to that of PRIO-PULL and SASOC shows significant performance improvements over OptQuest here as well. We observe that SASOC is a robust algorithm that gives a reliably good performance, is computationally efficient and is provably convergent, unlike OptQuest that does not possess these features.

5 Conclusions

We presented an efficient algorithm SASOC for optimizing staff allocation in the context of SS. We formulated the problem as a constrained optimization problem where both the objective and constraint functions were long run averages of a state dependent single-stage cost function. A novel single stage cost that balanced the conflicting objectives of maximizing worker utilizations and minimizing the over-achievement of SLA was employed. Numerical experiments were performed to evaluate SASOC against prior work in the context of a real-life service system. SASOC showed much superior performance compared to the state-of-the-art simulation optimization toolkit OptQuest, as it (a) was an order of magnitude faster than OptQuest, (b) found solutions of quality comparable to those found by OptQuest even in scenarios where OptQuest did not find feasibility even after 5000 iterations. By comparing SASOC results on two independent operational models, we showed that SASOCs performance is independent of the operational model of SS. We are in the process of developing and applying a second-order Newton based scheme with SPSA estimates for this problem. It would be of interest to compare the performance of that scheme with the one proposed here. It would also be of interest to prove the theoretical convergence of these algorithms.

References

1. Alter, S.: Service system fundamentals: Work system, value chain, and life cycle. IBM Systems Journal 47(1), 71–85 (2008)
2. Bhatnagar, S., Fu, M., Marcus, S., Wang, I.: Two-timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences. ACM Transactions on Modeling and Computer Simulation (TOMACS) 13(2), 180–209 (2003)
3. Banerjee, D., Desai, N., Dasgupta, G.: Simulation-based evaluation of dispatching policies in service systems. In: Winter Simulation Conference (2011) (under review)
4. Laguna, M.: Optimization of complex systems with optquest. OptQuest for Crystal Ball User Manual, Decisioneering (1998)
5. Verma, A., Desai, N., Bhamidipaty, A., Jain, A., Nallacherry, J., Roy, S., Barnes, S.: Automated optimal dispatching of service requests. In: SRII Global Conference (2011)
6. Wasserkrug, S., Taub, S., Zeltyn, S., Gilat, D., Lipets, V., Feldman, Z., Mandelbaum, A.: Creating operational shift schedules for third-level it support: challenges, models and case study. International Journal of Services Operations and Informatics 3(3), 242–257 (2008)
7. Spall, J.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control 37(3), 332–341 (1992)
8. Bhatnagar, S., Hemachandra, N., Mishra, V.: Adaptive multivariate three-timescale stochastic approximation algorithms for simulation based optimization. ACM Transactions on Modeling and Computer Simulation (TOMACS) 21(3) (2011)
9. Borkar, V.S.: Stochastic approximation: a dynamical systems viewpoint. Cambridge Univ. Pr. (2008)

Declarative Enhancement Framework for Business Processes*

Heerko Groefsema, Pavel Bulanov, and Marco Aiello

Distributed Systems Group, Johann Bernoulli Institute, University of Groningen,
Nijenborgh 9, 9747 AG Groningen, The Netherlands
{h.groefsema,p.bulanov,m.aiello}@rug.nl
http://www.cs.rug.nl/ds/

Abstract. While Business Process Management (BPM) was designed to support rigid production processes, nowadays it is also at the core of more flexible business applications and has established itself firmly in the service world. Such a shift calls for new techniques. In this paper, we introduce a variability framework for BPM which utilizes temporal logic formalisms to represent the essence of a process, leaving other choices open for later customization or adaption. The goal is to solve two major issues of BPM: enhancing reusability and flexibility. Furthermore, by enriching the process modelling environment with graphical elements, the complications of temporal logic are hidden from the user.

Keywords: BPM, Variability, Temporal Logic, e-Government.

1 Introduction

The world of Business Process Management (BPM) has gone through some major changes [4] due, among other things, to the advent of Web services and Service-orientation; providing opportunities as well as challenges [2]. Variability is an abstraction and management method that addresses a number of the open issues. In the domain of software engineering, variability refers to the possibility of changes in software products and models [13]. When this is introduced to the BPM domain, it indicates that parts of a business process remain either open to change, or not fully defined, in order to support different versions of the same process depending on the intended use or execution context, see for instance our survey [3]. Since BPM is moving into more fields of business and rely on autonomous remote building blocks, a need for flexible processes has arisen. Today, when a number of closely related processes are in existence, they are either described in different process models or in one large model using intricate branching routes, resulting in redundancy issues in case of the former and maintainability and readability issues in the case of the latter [14,3]. When applied to process models, variability introduces solutions to both issues by offering support for reusability and flexibility.

* The research is supported by the NWO SaS-LeG project, http://www.sas-leg.net, contract No. 638.001.207.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 495–504, 2011.

Considering the two features introduced by variability, we immediately notice how the two tend to coincide with design– and run–time aspects. The reusability attribute is usually introduced at design–time, and the flexibility attribute at run–time. Generally two approaches to variability are considered, imperative and declarative ones [12]. While imperative approaches focus on how a task is performed, declarative approaches focus on what tasks are performed. When mapping these to the two areas where variability would operate, design– and run–time, we notice four possible directions regarding variability in BPM. Most research currently focuses on the areas of imperative/design–time and declarative/run–time, while in this paper we shall focus on an approach which is able to capture both imperative and declarative methods at design–time [3]. The common problem with the frameworks of the imperative/design–time area is that in many cases they introduce unnecessary restrictions for process designers [1]. The source of such restrictions lies in the fact that imperative approaches require all variations from the main process to be predefined, limiting variations to only those added explicitly. On the other hand, declarative run–time frameworks lie on the far side of the semantic gap between the traditional and well–understood way of imperative process specification and the unintuitive way of declarative specification. We intend to solve these issues through a design–time declarative framework, in which the principles of process designing are similar to the ones of traditional imperative–based process modelling. This is achieved via introducing a set of visual modelling elements, which are internally transposed into a set of declarative constraints.

In this paper, we start by introducing the Process Variability - Declarative'n'Imperative (PVDI) framework which enhances process models with variability management through the introduction of temporal logics which capture the essence of a process. In doing so, we provide both the BPM and service composition domains with a formal way to model processes such that they can be used as a template for either the modelling of process variants or automatic service composition. In addition, by enriching the traditional process modelling environment with graphical elements, the complications of the underlying temporal logic is hidden from the user. We then show the expressive power of these graphical elements by utilizing them on an e-Government related case–study. Then, we evaluate the strengths and weaknesses of the framework by looking at the requirements for variability management in service-oriented systems introduced in [3]. Finally, we conclude with related work, and some final remarks.

2 The PVDI Framework

A PVDI *process* is defined as a directed graph and can serve as a frame for a modal logic of processes, including computational tree logic$^+$(CTL^+)[6]. Using CTL^+, we can introduce *constraints* for processes, which allow us to capture the basic meaning of a process in such a way that we can control changes within the process while keeping its essence, its intended use, intact as long as none of these constraints are violated. It is then possible to allow anyone to design a variant from such a template process without compromising its intended use.

Definition 1 (Process). *A process P is a tuple $\langle A, G, T\rangle$ where:*

- *A is a finite set of activities, with selected start $\odot$ and final $\otimes$ activities;*
- *$G = G_a \cup G_o \cup G_x$ is a set of gateways, consisting of and, or, and xor gates as defined by BPMN, respectively;*
- *$S = A \cup G$ is a set of states;*
- *$T = T_a \cup T_g$, where:*
- *$T_a : (A \backslash \{\otimes\}) \rightarrow S$ is a finite set of transitions, which assign a next state for each activity;*
- *$T_g : G \rightarrow 2^S$ is a finite set of transitions, which assign a nonempty set of next states for each gateway.*

In order to use a process as a model, we introduce a set of variables and a valuation function. We use the so–called **natural** valuation, that is, for each state (i.e., for each activity or gateway) we introduce its dedicated variable, and this variable is valuated to TRUE on this state only. Additionally, under the natural valuation we can use the same letter to represent both activity and its corresponding variable.

Definition 2 (Constraint). *A* **constraint** *over the process P is a computation tree logic$^+$ (CTL^+) formula. A constraint is* **valid** *for a process P iff it is valuated to TRUE in each state of the process under the natural valuation. More formally, let ϕ be a constraint, $\mathcal{M}$ be a model built on the process P using the natural valuation, and S be the set of states of the process P. Then ϕ is valid iff $\mathcal{M}, x, \phi \models TRUE\ \forall x \in S$.*

Notice how, now that we introduced constraints, a template process can be defined without strictly defining the precise structure of all activities and their respective ordering. The mapping of T can therefore be a partial one. As a result, a template may range from being a list of tasks to being a fully specified process. On the other hand, templates are enriched with a set of constraints, which outline the general shape of a process or set of processes.

Definition 3 (Template). *A template T is a tuple $\langle A, G, T, \Phi\rangle$ where:*

- *A, G, S, and T are from Definition 1;*
- *$T_a : (A \backslash \{\otimes\}) \rightarrow S$ is a finite set of transitions, which assign a next state for* **some** *activities;*
- *$T_g : G \rightarrow 2^S$ is a finite set of transitions, which assign a nonempty set of next states for* **some** *gateways.*
- *Φ is a finite set of constraints.*

In order to facilitate template design we introduce a number of graphic–elements for the design process. Constraints as embedded within templates are then generated from these graphical–elements. For every element contained in the template, we generate one or more CTL^+ formulas. Of course, this process differs greatly per element and even per situation. Some of the simpler elements directly resemble simple CTL^+ formulas, whereas other more complex structures resemble

a set of CTL^+ formulas. A potentially large number of graphic–elements can be considered for the template design, ranging from simple flows to complex groupings of tasks. We discuss only those elements needed in order to introduce the wide variety of options available trough PVDI.

2.1 Flow Constraints

Flow Constraints state that all elements from one set are followed by at least one element from another set in either a single or all paths. With a path being a series of consecutive transitions.

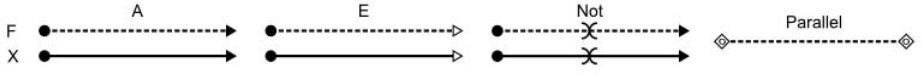


Fig. 1. Flow elements

Definition 4 (Flow Constraint). *A Flow Constraint F is a tuple $\langle s, t, \Omega, \Pi, N\rangle$ where:*

- *s is a finite set of process elements;*
- *t is a finite set of process elements; $s \bigcap t = \emptyset$;*
- *$\Omega \in \{A, E\}$ is a state quantifier from CTL^+;*
- *$\Pi \in \{X, F, G\}$ is a path quantifier from CTL^+;*
- *$N \in \{TRUE, FALSE\}$ being a negation.*

The graphical–elements related to Flow Constraints are shown in Figure 1. These graphical–elements describe flow relations over two dimensions; path and distance. The rows of Figure 1 relate to the temporal dimensions; F (Finally) and X (neXt), which require the linked elements to either follow each other eventually or immediately. The first two columns relate to the paths; E (there Exists a path) and A (for All paths), which require the linked elements to follow each other in either a path or all paths respectively. The third column represents a negation of two of these flows. These flows are related to simple CTL^+ formulas of the form $\mathcal{M}, p \models AXq$ (At p, in All paths, the neXt task is q), $\mathcal{M}, p \models EFq$ (At p, there Exists a path, where Finally a task is q), and their negations.

The CTL^+ constraints are generated from the graphical–elements according to the steps below. Remember that we reuse the same symbol to represent both the state and the variable representing that state.

1. Let $s_1, \ldots s_n$ be elements of *source*, and $t_1, \ldots t_m$ be elements of *target*.
2. If $N = TRUE$, then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega\Pi(t_1 \vee t_2 \vee \ldots t_m)$.
3. If $N = FALSE$, then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega\Pi\neg(t_1 \vee t_2 \vee \ldots t_m)$.
4. If $N = FALSE$ and $\Pi = F$, then create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow \Omega G\neg(t_1 \vee t_2 \vee \ldots t_m)$.

2.2 Parallel Constraints

Parallel Constraints (Figure 1), enforce that two sets of states do not appear in the same path. Meaning that any series of consecutive transitions taken from any state in either set may never lead to any element from the other set.

Definition 5 (Parallel Constraint). *A Parallel Constraint P is a tuple $\langle S, T\rangle$ where:*

- *S and T are sets of states; $S \bigcap T = \emptyset$;*

The CTL^+ constraints are generated from the graphical–elements according to the steps below. This construction enforces that all states from each set or branch, S and T, can never be followed by any state from the other set. Further constraints between states in the branches S and T, and the specification of a specific preceding gate should be added through other means, i.e. flow constraints.

1. Let $s_1, \ldots s_n$ be elements of S, and $t_1, \ldots t_m$ be elements of T.
2. Create a formula $(s_1 \vee s_2 \vee \ldots s_n) \Rightarrow AG\neg(t_1 \vee t_2 \vee \ldots t_m)$
3. Create a formula $(t_1 \vee t_2 \vee \ldots t_m) \Rightarrow AG\neg(s_1 \vee s_2 \vee \ldots s_n)$

2.3 Frozen Groups

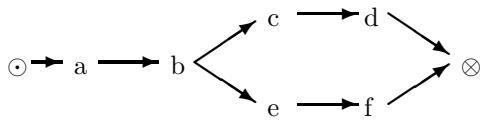
Frozen Groups are sub–processes which cannot be modified. Such a restriction is achieved by generating a set of CTL^+ formulas which constrain all elements inside of a frozen group. Figure 2 includes an example of such a group.

Definition 6 (Frozen group). *A frozen group is a pair $\langle P, M\rangle$, where P is a process (Definition 1), and $M \subseteq P_A$ is a set of mandatory activities.*

The group itself is a process but can be part of a bigger process or template, which is why we refer to groups as sub–processes. Since this element is not limited to one set of simple sources and targets like the previous ones, its transformation to CTL^+ is more complicated, and yields a set of CTL^+ formulas instead of a single one. A Frozen Group will be constrained in such a way that all paths, being a series of consecutive transitions, within it, must be kept intact. The CTL^+ constraints are generated from the graphical–elements (Figure 2) according to the steps below.

1. For each state a let the chain of states $P = \langle a_1, \ldots a_k\rangle$ be a path between a and $\otimes$. If the path P is not empty, then do the following steps:
2. If the path P is empty (i.e., the next step is the final one), then create a CTL^+ formula of type $a \Rightarrow AX\otimes$.
3. If the path P is the only path from a to $\otimes$, then create a CTL^+ formula of type $a \Rightarrow A((a_1 \vee a_2 \vee \ldots \vee a_k)U\otimes)$. A and U are quantifiers of CTL^+.
4. If there are several paths which lead from a to $\otimes$, let say paths $P_1, \ldots P_t$ lead from a to $\otimes$. Then, the formula of step 3 becomes more complicated: $a \Rightarrow A[P_1^T \vee \ldots \vee P_t^T]$, where $P_i^T = (a_1^i \vee a_2^i \vee \ldots \vee a_k^i)U\otimes$, with $a_1^i \ldots a_k^i$ being the steps of the path P_i.

Example 1 (Frozen Group).



$$
\begin{aligned}
a &\Rightarrow A[(b \vee c \vee d)U\otimes] \vee \\
&\quad [(b \vee e \vee f)U\otimes] \\
b &\Rightarrow A[(c \vee d)U\otimes] \vee [(e \vee f)U\otimes] \\
c &\Rightarrow A(dU\otimes) \\
e &\Rightarrow A(fU\otimes) \\
d &\Rightarrow AX\otimes \\
f &\Rightarrow AX\otimes
\end{aligned}
$$

In this example, the process illustrated above is encoded as a set of CTL^+ formulas. To do that, we take each step one by one and generate a formula according to the algorithm presented above. The first two formulas are the most complicated, because there are two possible paths from a to $\otimes$ (and from b to $\otimes$ as well). Therefore, according to the step 4 of the algorithm, the formula splits into two pieces, one per each path. In the case of activity b, one path contains activities c and d, and the other one contains e and f.

2.4 Semi–frozen Group

Semi–frozen groups are Frozen Groups with less strict constrains, allowing for removal or replacement, addition, or moving of activities. The advantage of this group representation is that for example any activity inside a frozen group can be made optional and can therefore be removed during the customization process.

Optional activities will not affect the consistency of the group, since the CTL^+ formulas remain valid as long as at least no new activity is added into the group. Actually, if an activity a is removed, then all formulas of kind $a \Rightarrow \ldots$ become automatically valid, and formulas of kind $b \Rightarrow aU\otimes$ are valid as long as there is either activity a or nothing between b and $\otimes$. The same is true for more complicated cases like $b \Rightarrow a \vee \ldots U\otimes$. In other words, any activity can be removed from a frozen group but not replaced by another one.

Weaken a link between two states thus allowing to insert a new activity into a specific place(s) in a group. To do that, we have to modify the base algorithm.

1. Let the chain of states $P = \langle a_1, \ldots a_k \rangle$ be a path between a and $\otimes$. For example, the link between a_i and a_{i+1} is "weak". Then the appropriate CTL^+ formula is $a \Rightarrow A[(a_1 \vee \ldots a_i)UAFA[(a_{i+1} \vee \ldots a_n)U\otimes]]$.
2. If there are several "weak" links in a path, for example, links $a_i \rightarrow a_{i+1}$ and $a_j \rightarrow a_{j+1}, i < j$ are weak. In this case, build a formula $a_j \Rightarrow \phi$ according to p.1, and the final formula is $a \Rightarrow A[(a_1 \vee \ldots a_i)UA[(a_{i+1} \vee \ldots a_j)U\phi]]$, where ϕ is retrieved in the previous step.
3. The same recursive rule applies in the case of three or more "weak" links.

Making an activity floating thus allowing to swap two activities or drag an activity into another place in the group. The algorithm is as follows:

1. Create the set of constraints for the group as described above;
2. To make an activity a floating, remove all constraints of kind $a \Rightarrow \phi$.

Each move of an activity can be split into two atomic operations: (i) remove the activity (ii) insert this activity into another place. As it has already been shown above, no special correction needed in order to remove the activity. The next step, however, is only possible when there is a "weak" link in the process – otherwise we can only put the activity back into its original place.

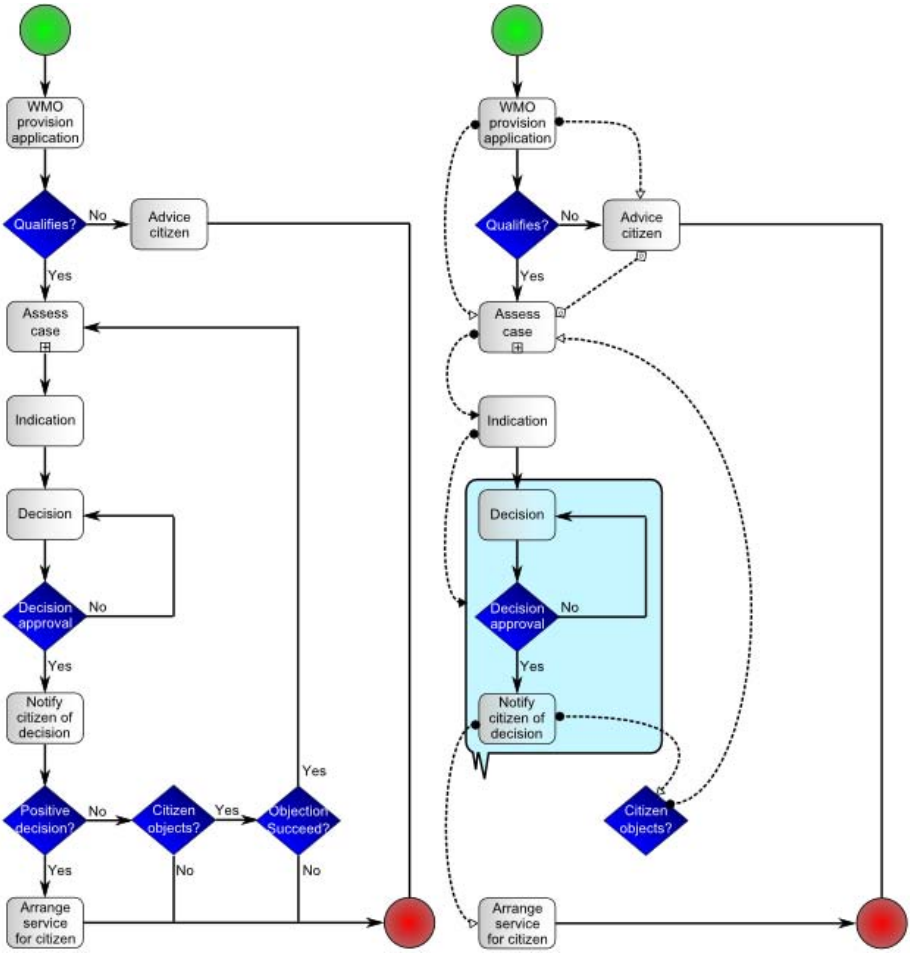


Fig. 2. Simplified WMO process (Left) and template (Right)

3 Case–Study: Variability in Local eGovernment

The Netherlands consists of 418 municipalities which all differ greatly. Because of this, each municipality is allowed to operate independently according to their local requirements. However, all the municipalities have to provide the same services and execute the same laws. An example of such a law which is heavily subjected to local needs is the WMO (Wet maatschappelijke ondersteuning, Social Support Act, 2006), a law providing needing citizens with support ranging from wheelchairs, help at home, home improvement and homeless sheltering. Figure 2 illustrates a simplified version of a WMO process found at one of the Dutch municipalities of the Northern region of the Netherlands (Left), and an example of how one transforms this process into a template usable by all municipalities (Right). Variant processes can then be obtained via customization of the template process. The flexibility of a process, bounded with such constraints, can vary from zero (a frozen block over the whole process) to unlimited, when there are no constraints at all. On examination of the figure, we notice how constraints are not evaluated for templates but only for processes resulting from templates. Using a frozen group we restrict the decision making process, which because of this must be kept intact at all times. The rest of the template is captured using simple flow constraints and one parallel constraint. Therefore, extra activities could be easily included at most places except the frozen group, and certain parts could be moved around without affecting the correctness of the process.

4 Related Work

Existing tools and frameworks for variability management in BPM concentrate on a single view on variability. Most focus either on imperative design–time or on declarative run–time solutions, while we combine imperative and declarative techniques. In addition, most research disregards services entirely and focuses solely on the BPM aspects. One framework which does look at services specifically is the Variability extension to Business Process Execution Language (VxBPEL) [15] that we proposed previously. This BPEL extension introduces a number of new keywords allowing for the inclusion of variation points, variations, and realization relations into BPEL. Other imperative frameworks focus solely on BPM, most notably ADEPT [5], Process Variants by Options(Provop) [8], and configurable workflow models [7]. These imperative frameworks however require that all variability options must be included into the template process directly, leading to maintainability and readability issues. On the other hand, our framework does not focus on what can be done, but on what should be done, and leaves any other options open; resulting in a much higher degree of flexibility. Declarative frameworks focus mostly on run–time solutions to flexibility issues, of which most notable are the DECLARE framework [10], and Business Process Constraint Network (BPCN) and Process Variant Repository(PVR) [9,11]. Our framework on the contrary hides this complexity by the introduction of simple graphical elements which can be directly incorporated into business process models.

Table 1. Evaluation on Requirements

	Requirement	Support
	Structural variations	
(a)	*Insert Process Fragment*	Achieved via introducing a "weak" link in a frozen group.
(b)	*Delete Process Fragment*	Achieved via making an activity optional.
(c)	*Move Process Fragment*	Achieved via an floating activity at frozen groups.
(d)	*Replace Process Fragment*	Achieved through a combination of an optional activity and a weak link.
(e)	*Swap Process Fragment*	Special case of moving a process fragment, therefore it is also supported.
	Constraint expressions	
(a)	*Mandatory selection*	Achieved with PVDI through a flow constraint emerging from $\odot$ and targeting the activity.
(b)	*Prohibitive selection*	Achieved with PVDI through a negated flow constraint emerging from $\odot$ and targeting the prohibited activity.
(h)	*Mandatory execution*	Achieved with PVDI through a flow constraint of the type "for All paths" emerging from $\odot$.
(i)	*Order of execution*	Achieved through flow constraints.
(j)	*Parallel execution*	Achieved through a combination of flow constraints emerging from a gate and a parallel constraint.
(k)	*Exclusive execution*	Achieved through a combination of flow constraints emerging from a gate and a parallel constraint.

5 Conclusion

We have shown how a variability framework for process modelling adds a large amount of functionality to both the area of BPM, as well as service composition. By enriching the process modelling environment with graphical elements, we provided an easy way to hide the complications of temporal logic from the end user. Using a case–study from the area of e–Government, we then explained how the high amount of reusability and flexibility enriches templates in such a way that variants become easily maintainable and templates easily readable.

In order to evaluate the flexibility of PVDI, we consider the expressive power requirements proposed in [3]. Two types are discussed: structural variations related to imperative techniques, and constraint expressions related to declarative techniques. In Table 1 the requirements which are directly supported by our framework are enlisted along with the description of the support. Due to the approach taken with PVDI, both declarative and imperative techniques are being considered. In cases of imperative techniques we consider a semi–frozen block, which keeps the process structure intact while allowing some modification.

Many items are open for further investigation, among which the support of data flows, dependencies between constraints, template publication, and automated composition from templates as a constraint satisfaction problem.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
2. van der Aalst, W., Jablonski, S.: Dealing with workflow change: Identification of issues and solutions. International Journal of Computer Systems, Science, and Engineering 15(5), 267–276 (2000)
3. Aiello, M., Bulanov, P., Groefsema, H.: Requirements and tools for variability management. In: IEEE Workshop on Requirement Engineering for Services (REFS 2010) at IEEE COMPSAC (2010)
4. Bandara, W., Indulska, M., Sadiq, S., Chong, S.: Major issues in business process management: an expert perspective. In: European Conference on Information Systems, ECIS (2007)
5. Dadam, P., Reichert, M.: The adept project: a decade of research and development for robust and flexible process support. Computer Science - R&D 23(2), 81–97 (2009)
6. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 169–180 (1982)
7. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., Rosa, M.L.: Configurable workflow models. Int. J. Cooperative Inf. Syst. 17(2), 177–221 (2008)
8. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process life cycle. In: ICEIS, vol. (3-2), pp. 154–161 (2008)
9. Lu, R., Sadiq, S., Governatori, G.: On managing business processes variants. Data Knowl. Eng. 68(7), 642–664 (2009)
10. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: Meersman, R. (ed.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
11. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. 30(5), 349–378 (2005)
12. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process flexibility: A survey of contemporary approaches. In: Dietz, J.L.G., Albani, A., Barjis, J. (eds.) CIAO! / EOMAS. LNBIP, vol. 10, pp. 16–30. Springer, Heidelberg (2008)
13. Sinnema, M., Deelstra, S., Hoekstra, P.: The COVAMOF Derivation Process. In: Morisio, M. (ed.) ICSR 2006. LNCS, vol. 4039, pp. 101–114. Springer, Heidelberg (2006)
14. Sun, C., Rossing, R., Sinnema, M., Bulanov, P., Aiello, M.: Modelling and managing the variability of web service-based systems. Journal of Systems and Software 83, 502–516 (2010)
15. Sun, C.-a., Aiello, M.: Towards Variable Service Compositions Using VxBPEL. In: Mei, H. (ed.) ICSR 2008. LNCS, vol. 5030, pp. 257–261. Springer, Heidelberg (2008)

RSCMap: Resiliency Planning in Storage Clouds

Vimmi Jaiswal[2], Aritra Sen[1], and Akshat Verma[1]

[1] IBM Research, India
[2] JIMS, India

Abstract. Clouds use economies of scale to host data for diverse enterprises. However, enterprises differ in the requirements for their data. In this work, we investigate the problem of resiliency or disaster recovery (DR) planning in a cloud. The resiliency requirements vary greatly between different enterprises and also between different datasets for the same enterprise. We present in this paper Resilient Storage Cloud Map (RSCMap), a generic cost-minimizing optimization framework for disaster recovery planning, where the cost function may be tailored to meet diverse objectives. We present fast algorithms that come up with a minimum cost DR plan, while meeting all the DR requirements associated with all the datasets hosted on the storage cloud. Our algorithms have strong theoretical properties: 2 factor approximation for bandwidth minimization and fixed parameter constant approximation for the general cost minimization problem. We perform a comprehensive experimental evaluation of RSCMap using models for a wide variety of replication solutions and show that RSCMap outperforms existing resiliency planning approaches.

1 Introduction

Infrastructure as a Service (IaaS) clouds have rapidly emerged as a popular IT delivery model to reduce costs and improve resource utilization in data centers. Data governance and Reliability have emerged as the key challenges in the adoption of clouds by enterprises. The always-online 24/7 business model of enterprises today has led to a situation, where downtime leads to direct business loss and customer erosion. Further, enterprises must comply with many regulations that require data governance. By moving the data into the cloud, end users lose the ability to govern their own data set and rely on the service providers to guarantee the safety of their data.

The unique selling point of cloud computing has been a low cost delivery model, which necessitates multi-tenancy or sharing of resources between end users and standardized offerings. Multi-tenancy implies that end users with varying data governance or reliability needs are co-hosted on the cloud. Cloud providers would increasingly need to provide disaster recovery support for individual customers in line with their governance and reliability requirements.

The crux of a disaster recovery plan is data replication [4,5]; where application data is replicated on same or a different site. Data replication technologies differ in (a) the time taken to restore data (RTO) (b) the amount of updates lost before disaster in terms of seconds or minutes (RPO) (c) the impact on application

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 505–512, 2011.

performance due to data replication and (d) the maximum distance that they can replicate the data. Resilient data replication technologies with quick recovery are very expensive, and in some cases require expensive network infrastructure as well [1,2,11]. Hence, applying a common replication technology for all data in the cloud is infeasible and a replication technology needs to be selected as per user requirements. Further, the licensing models of various replication technologies are complicated; differing based on the amount of data being replicated, the amount of unique updates being made by the applications, the number of users, and even the number of servers. The diversity amongst the capabilities and costs of replication technologies makes disaster recovery planning a complex problem.

The easy way out that enterprises take today is to select one replication technology to mirror all its data in an ad-hoc manner, independent of its criticality. Such an approach is infeasible in a cloud, which has the goal to provide IT services at a low price point. Hence, a formal approach to resiliency planning is a necessity for emerging clouds.

1.1 Contribtution

The contribution of our work is two-fold. First, we present a formal framework to study the DR plan composition problem, using realistic models for high performance replication technologies and their costs. To the best of our knowledge, this is the first attempt to rigorously study the DR plan composition problem. Secondly, we present an efficient algorithm for the DR plan composition problem that provides a constant factor approximation for an important sub-class of the problem. Further, we present extensions of the algorithm for the general case with an approximation guarantee bounded by the number of parameters in the cost function, which is typically a constant. Our model and algorithms have been developed in the context of existing DR Planning tools [8,3,10,12].

2 Model and Problem Formulation

We now present the model for the *Plan Composition Problem*. We start with some definitions.

Definition 1. *Disaster Recovery Service Class (DRSC): A Disaster Recovery Service Class denotes a specific class of service in terms of meeting Disaster Recovery requirements. The DRSCs are named as Gold, Silver, Bronze where the classification is based on attributes like RTO, RPO, Application Impact, distance etc.*

Definition 2. *Data Container: A Data Container D_i is any set of logically grouped data that has identical DR requirements. (e.g., all files of a particular user (/home/akshat etc) or of a particular type (temp files, mpeg files) that have the same criticality). For a data container D_i, s_i denotes the space (in terms of GBytes) and w_i denotes the write workload seen by the data container.*

Definition 3. *Replication Solution: A Replication Solution R_j is any technology that can provides a specific DR Protection for a given failure. Hence, any Replication Solution has tuples of failure class and DRSC parameters. To take an example, the IBM Global Mirror solution provides an RTO of 30 mins and an RPO of 3 seconds for site failure.*

Disaster recovery planning essentially consists of two phases [10]. The first phase is what we term as the matching phase. In the matching phase, we match the requirements of a data container to replication solutions. Hence, for each data container D_i, we create a set of solutions RS_i that meet the DR requirements of D_i. In the second phase called the *plan composition* phase, we select one replication solution for each data container such that the overall cost of deploying the solutions is minimized. This is followed by actual deployment of the computed plan. The cost-minimizing plan composition is the focus of this work. For more details on the matching problem and the plan deployment problem, the reader is referred to [10].

2.1 The DR Cost Minimization Framework

Consider the problem of designing a storage provisioning plan for a set of data containers, each of which has certain service requirements. Every data container belongs to a storage service class (SSC), where each SSC has performance, availability and Disaster Recovery Service Class(DRSC) with it. We focus only on the Disaster Recovery Service Class in this work. Every data container is associated with multiple replication solutions that meet its DR requirements (as a result of the matching step) and its workload information (space s_i, read throughput r_i and write throughput w_i). Every replication solution also has an associated cost metric, which is a function of the space of data protected by the replication solution and its characteristics (write rate, read rate, users etc). We use the terms s_i, r_i, w_i to denote the space, read throughput and write throughput of data container D_i and the terms s_j, r_j, w_j to denote the total space, read throughput and write throughput of all data containers protected by replication solution R_j. Further, each replication solution transforms the traffic of the data containers protected by it and this is captured using a bandwidth transformation factor $B_{i,j}$. The output of the problem is a detailed provisioning plan, which maps a data container to exactly one of its eligible solution, and strives to minimize the cost of the overall solution. A formal mathematical description of the problem is available in an extended report [6].

2.2 Disaster Recovery Service Class (DRSC) Model

The Disaster Recovery Service Class notion captures the important attributes of disaster recovery. DRSC is used both to denote the DR requirements of a data container as well as the DR capabilities of a replication solution. A DRSC consists of the following attributes

- Recovery Time Objective (RTO): Recovery Time Objective is defined as the maximum time that will be taken to recover data after a disaster.

- Recovery Point Objective (RPO): The Recovery Point objective denotes the amount of time by which the recovered data lags behind the lost data.
- Application Impact:Application Impact denotes the latency impact that occurs as a result of deploying the replication solution.
- Resource Impact:A replication solution takes away system resources (CPU cycles, Disk bandwidth), which is captured by resource impact.
- Distance:This represents the maximum supported distance between the copies of a replication solution.

2.3 Replication Solution Model

A Disaster Recovery requirement is met by deploying a replication technology. A replication technology is characterized by the RTO, RPO, impact and distance parameters provided by the technology. In addition, deploying a replication technology leads to license cost, labour cost, and network costs. The sum of all these costs is captured as total cost, which we aim to minimize in this work. The network costs of replication are determined by the amount of network traffic generated due to the technology. A synchronous replication solution generates network traffic equal to the total amount of writes on the data container. Hence, the network bandwidth provisioned equals the peak write rate (or burst rate) to the data container. On the other hand, an asynchronous replication technology only needs network bandwidth equal to the average write rate. Further, some technologies like IBM Global Mirror use techniques like write-coalescing to reduce the number of writes and need to provision only for unique writes. We use the bandwidth transformation function $B_{i,j}$ is to capture the real network bandwidth required by the replication solution.

3 Model Assumptions

We now use insights from the practical setting of the *Plan Composition* problem to simplify the problem. The reader may observe that all the restricted versions of the problem we consider that take into account the practical constraints of the problem are also NP-hard. We first make the simplifying assumption that the amount of data to be protected by any DR Service Class is more than the space taken by a single instance of any replication solution. One may observe that this is true for public as well as private clouds that protect a large amount of data. We now investigate more intricate properties about the replication solution sets RS_i that are typically true in real deployments.

3.1 Pure Subset Replication Set Property

We observe that a replication solution that can meet the *Gold* service class would also meet *Silver* or *Bronze* service class. Hence, any data container that requires a low level DR protection can use all the replication solutions that provide protection for that or any higher class for the given failure type. We capture this real-life constraint in the *Pure Subset Replication Set Property* defined next.

Definition 4. *Pure Subset Replication Set: A Plan Composition problem is said to satisfy the Pure Subset Replication Set Property if*

$$\forall D_i, D_j \quad RS_i \cap RS_j \neq \phi \quad \Rightarrow \quad RS_i \subseteq RS_j \quad or \quad RS_j \subseteq RS_i \tag{1}$$

3.2 Traffic Independent Bandwidth Transformation

The replication bandwidth generated by a replication solution typically depends entirely on the replication technology being used and not on the properties of the data container. We use this fact to simplify the bandwidth transformation function ($B_{i,j}$) from being a function of both the replication technology R_j and data container D_i to being a function B_j of only the replication technology R_j. This greatly simplifies the problem since the the cost of a replication solution only depends on additive properties of data containers (space, burst rate, write rate etc for multiple data containers can be added to obtain the parameters for the replication solution whereas $B_{i,j}$ can not be added).

4 RSCMap Algorithms

We now present fast algorithms that solve the plan composition problem and prove approximation guarantees for the algorithms. We first present algorithms for a simplified version of the problem, where cost is only a function of the size of data being protected, i.e., Cost of a replication technology is a function of only the total space taken by all the data containers protected by the technology. This one-dimensional cost problem, as noted earlier, is also NP-hard and is of independent interest, since it captures the bandwidth minimization variant of the plan composition problem. Moreover, we will later enhance the algorithm to work with multiple parameters or multi-dimensional cost functions.

4.1 Algorithms for the One-Dimensional Cost Problem

In the One-dimensional variant of the problem, cost of a replication technology is dependent on only one parameter. We assume that cost is a function of space, i.e., $Cost_j = C(s_j)$, while noting that the same formulation and results hold for any other dimension as well.

A Plan Composition algorithm makes two decisions: (i) Pick a [cost,space] point for each selected replication technology and (ii) map data containers to a selected replication technology for DR protection. In order to solve the first problem, we use the greedy strategy of filling up as much space as possible at the least cost. Hence, we favour replication solution corner points $\min\{A_j\}$ that can provide protection to data at the least cost per unit amount of data protected. (From now on, we use replication solution R_j to indicate R_j at the least slope corner point) This greedy strategy may lead to incompletely filled replication solutions R_j (or equivalently incompletely filled replication solution corner points $\min\{A_j\}$) and we bound this fragmentation cost by separately listing these partially filled replication solutions, to add to the least slope solutions later. For

the second problem, we pick the data container D_i to be protected first that have the minimum number of eligible replication technologies (smallest $|RS_i|$). The LeastSlopeFirst (LSF) algorithm greedily picks replication technologies and bounds the cost due to fragmentation. The *selectMostConstrainedDC* method captures the data container selection technology. The procedures used by *LSF* are detailed next.

Definition 5. *selectMostConstrainedDC Selection: Given a set of data containers D_i, a replication solution R_j, and a space constraint s_j^*, selectMostConstrainedDC sorts the data containers by the arity of their list of feasible replication solutions. It then keeps on adding data containers from the sorted list to its selection, till the accumulated space of the selected data containers equals s_j^* .*

Definition 6. *PickBest Procedure: Given a set of k partial matches PM_k and a match M, the PickBest procedure returns the minimum cost subset of PM_k and M that covers all the data containers.*

We proved the optimality of the *selectMostConstrainedDC* data container selection process and then use it to prove approximation guarantees on *LSF*. For lack of space, all proofs have been omitted and are available in [6].

Lemma 1. *selectMostConstrainedDC is a Maximum Matching Selection.*

Theorem 1. *The LeastSlopeFirst algorithm returns a plan P such that the cost C_P of the plan is no more than twice the cost C_O of the optimal solution O.*

4.2 Algorithms for General Cost Functions

We now consider the k-dimensional version of the Plan Composition problem, where the cost of a replication solution depends on k parameters/dimensions, from d^1 to d^k. Since any traffic transformation (e.g., bandwidth transformation) is already captured in the cost function (Sec. 3.2), the value of a replication solution R_j across any dimension d^l is summation of the value along the dimension d^l all the data containers protected by R_j.

$$\forall l \in [1, d], \forall j \in [1, m], \quad d_j^l = \sum_{i=1}^{N} x_{i,j} d_i^l \tag{2}$$

Our strategy remains the same as in *LSF*. We pick replication solutions that can protect data containers at the least cost per unit amount of data protected. However, since data containers have more than one relevant dimension now, we need to make a slight modification. In our first version, we order dimensions by their relative cumulative sizes ($d_a^l = \sum_{i=1}^{N} d_i^l$). We then pick the dimension d^{max} with the greatest accumulated sum ($\max_{l=1}^{k} d_a^l$) and order replication solution corner points by $\frac{\delta C_j}{\delta d_j^{max}}$. We follow the LSF procedure with the dimension d^{max} replacing the space parameter s. Once, we use up the dimension d^{max}, we use the next largest dimension and continue till we have protected all data containers ($currDC = \phi$).

We also make the following enhancement to *selectMostConstrainedDC* to capture the affinity of data containers with specific dimensions. When two or more data containers have the same arity, we use a tie-breaker method to choose an ordering among them. For tie breaking, we consider the solid angle the data container's requirement vector makes with the dominant dimension axis. We first select containers whose requirement runs only along the dominant dimension, in decreasing order of magnitude. We next select containers in decreasing order of their solid angle computed in the previous step. We add data containers from this sorted list to its selection, till the accumulated value in any dimension of the selected data containers equals the capacity of the replication solution in that dimension. To illustrate the strategy, consider a hyper-cuboid, where we move along the dominant dimension until we hit its wall. We then pick another dimension to move along. We have proved the following result for the algorithm.

Theorem 2. *Multi-dimensional LSF returns a plan that has a cost no more than* $2k$ *times the cost of the optimal solution.*

We now propose a variant of Multi-dimensional LSF called $LSAF$ that also has an approximation guarantee of $2k$. However, it satisfies additional theoretical properties, which may lead to a better approximation guarantee. The LeastSolidAngleFirst ($LSAF$) algorithm differs from multi-dimensional LSF by ordering the replication solutions by $\frac{\delta C}{\delta\sqrt{\sum_{l=1}^{k}(d^l)^2}}$. Hence, $LSAF$ uses the derivative of the cost (or the solid angle) to pick the replication solutions. If at any given time, it consumes any particular dimension than it removes that dimension from the derivative calculation. Hence, the method has a list of active dimensions, which is initialized with all the dimensions and pruned as one or more dimensions get exhausted. To take a geometric view, if the process hits any walls of the hyper-cuboid, it takes out that dimension from any future calculations (i.e. the set of active dimensions). Along the lines of theorem 2, one can show that $LSAF$ provides an approximation guarantee of $2k$. However, note that $LSAF$ always incurs less cost per unit distance traveled in the protected space of active dimensions than the optimal. On the other hand, the optimal can travel along the diagonal whereas $LSAF$ can be forced to travel along the sides of the hypercuboid. We have proved the following results.

Lemma 2. *The total distance traveled by LSAF is no more than* $\sqrt(k)$ *times the distance traveled by optimal.*

5 Related Work and Conclusion

Disaster Recovery in data centers has attracted a lot of attention in recent times with work on improved replication mechanisms [7], modeling dependability of a system [9], planning papers [8,3,10], plan deployment papers [10] and recovery papers [12]. Nayak *et al.* present an end-to-end disaster recovery planning and deployment tool whose focus is on the match-making phase and plan deployment [10]; our work is geared towards providing the optimization engine for

the *plan composition* phase. Keeton *et al.* [8] tackle the problem of creating a cost-effective disaster recovery plan for a single application site and extend it in [3] to include shared applications. They use a linear cost model, which does not take into account price-bundling, which makes the cost functions incapable of capturing many typical planning scenarios. Further, they do not propose any optimization algorithms and suggests using off-the-shelf algorithms. The optimization algorithms used (including the new one in [3]) are based on mixed-integer programming and expensive, making them unsuitable for use in highly interactive disaster recovery planning tools [10]. Hence, earlier planning work [8,10,3] only provide a framework for DR planning without efficient algorithms, and this is exactly the deficiency that we address. In this way, our work complements earlier work; by using the match-making methodologies of [10] and the cost-functions proposed in [8,3] for the optimization framework. Our proposed algorithms provide the last missing piece required for integrated DR Planning.

References

1. Datamonitor Computer Wire Article, `http://www.computerwire.com/industries/research/?pid=1CEC81FD-5FDA-41D8-8FFC-79A959A87FD7`
2. Synchronous Optical Network, `http://www.iec.org/online/tutorials/acrobat/sonet.pdf`
3. Gaonkar, S., Keeton, K., Merchant, A., Sanders, W.H.: Designing Dependable Storage Solutions for Shared Application Environments. In: Proc. DSN (2006)
4. IBM TotalStorage Solutions for Disaster Recovery. In: IBM Redbook, `http://www.redbooks.ibm.com`
5. IBM TotalStorage Business Continuity Solutions Overview. In: IBM Redbook, `http://www.redbooks.ibm.com`
6. Jaiswal, V., Sen, A., Verma, A.: RSCMap: Resiliency Planning in Storage Clouds. In: IBM Technical Report RI11012 (2011)
7. Ji, M., Veitch, A., Wilkes, J.: Seneca: remote mirroring done write. In: Proc. USENIX Annual Technical Conference (2003)
8. Keeton, K., Santos, C., Beyer, D., Chase, J., Wilkes, J.: Designing for Disasters. In: Proc. USENIX FAST (March 2004)
9. Keeton, K., Merchant, A.: A framework for evaluating storage system dependability. In: Proc. DSN (2004)
10. Nayak, T., Routray, R., Singh, A., Uttamchandani, S., Verma, A.: End-to-end Disaster Recovery Planning: From Art to Science. In: IEEE NOMS (2010)
11. Oracle License Prices, `http://www.pro-dba.com/pricing.html`
12. Verma, A., Voruganti, K., Routray, R., Jain, R.: SWEEPER: An Efficient Disaster Recovery Point Identification Mechanism. In: Usenix FAST (2008)

Dynamically Selecting Composition Algorithms for Economical Composition as a Service

Immanuel Trummer and Boi Faltings

Artificial Intelligence Laboratory
Ecole Polytechnique Fédérale de Lausanne
{immanuel.trummer,boi.faltings}@epfl.ch

Abstract. Various algorithms have been proposed for the problem of quality-driven service composition. They differ by the quality of the resulting executable processes and by their processing costs. In this paper, we study the problem of service composition from an economical point of view and adopt the perspective of a Composition as a Service provider. Our goal is to minimize composition costs while delivering executable workflows of a specified average quality. We propose to dynamically select different composition algorithms for different workflow templates based upon template structure and workflow priority. For evaluating our selection algorithm, we consider two classic approaches to quality-driven composition, genetic algorithms and integer linear programming with different parameter settings. An extensive experimental evaluation shows significant gains in efficiency when dynamically selecting between different composition algorithms instead of using only one algorithm.

Keywords: Quality-Driven Service Composition, Composition as a Service, Dynamic Algorithm Selection.

1 Introduction

Over the last years, large scale, public registries for Web services have been emerging. These include domain-specific registries (e.g. biology[1], geospatial Web services[2]) as well as general purpose registries such as Seekda![3] which currently advertises over 28.000 Web services. Due to the large number of available services, a common situation is that several services are able to fulfill the same functionality. In order to select between them, non-functional properties such as service availability and response time can be taken into account. This issue is at the heart of quality-driven service composition [12] (QDSC). In QDSC, tasks of an abstract workflow are associated with sets of functionally equivalent services which differ in their non-functional properties. The goal is to select one service for every task such that the aggregated quality properties of the workflow are optimized while certain minimum requirements are fulfilled. Various algorithms have

[1] http://www.biocatalogue.org/
[2] http://services.eoportal.org/
[3] http://webservices.seekda.com/

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 513–522, 2011.

been proposed for QDSC. Some of them produce optimal executable workflows but have high resource requirements, others sacrifice optimality for efficiency. In this paper, we propose to select different composition algorithms for different workflow templates in order to maximize the overall performance. The selection should consider structural properties of the template as well as workflow priority. Classifying workflow templates according to structural properties allows to predict the behavior of composition methods more accurately. Considering workflow priority allows to select high-quality composition algorithms for high-priority workflows and high-efficiency algorithms for low-priority workflows.

The original scientific contributions of this paper are *i)* an algorithm that maps workflow templates to composition algorithms, minimizing the overall processing costs for a specified average target quality, and *ii)* an extensive experimental evaluation of our algorithm in comparison to naive approaches. The remainder of the paper is organized as follows. In Sect. 2, we present a motivating scenario, in Sect. 3 the corresponding formal model. We review related literature in Sect. 4. In Sect. 5, we describe our approach in detail, followed by the experimental evaluation in Sect. 6. We conclude with Sect. 7.

2 Motivating Scenario

We adopt the perspective of a fictive Composition as a Service provider as described and motivated by Rosenberg et al. [10] and Blake et al. [4]. Fig. 1 shows an overview of the corresponding architecture. Clients are companies with a portfolio of business processes corresponding to different products and services (presumably more than one). Clients submit their whole portfolio as set of composition requests to the composition service. Every request is associated with a specific workflow template, minimum requirements on the QoS of the executable process, and a utility function weighting between different QoS of the executable process. Clients subscribe and pay for regularly receiving executable processes corresponding to their requests. It is necessary to repeat the composition regularly since the set of available services may change. We consider the processing cost for the provider to be proportional to the running time of the used composition algorithms (this is the case if an Infrastructure as a Service offer like Amazon EC2 [1] is used). The 80/20 rule predicts strong variations in the relative importance of different products and services in industry [8]. It is plausible that this translates to different priorities of the workflows within the portfolio. We will use the number of workflow executions per time unit as priority measure while different measures could be applied as well. Clients specify the expected number for every workflow (eventually using a rough estimate first and refining it later). The composition provider can exploit this information and select computationally cheap composition algorithms for less frequently executed workflows. These cost savings can in part be passed on to the clients. We assume that the composition provider has set a target average quality for the resulting compositions and assigns requests to algorithms in order to minimize the processing cost while guaranteeing this average quality.

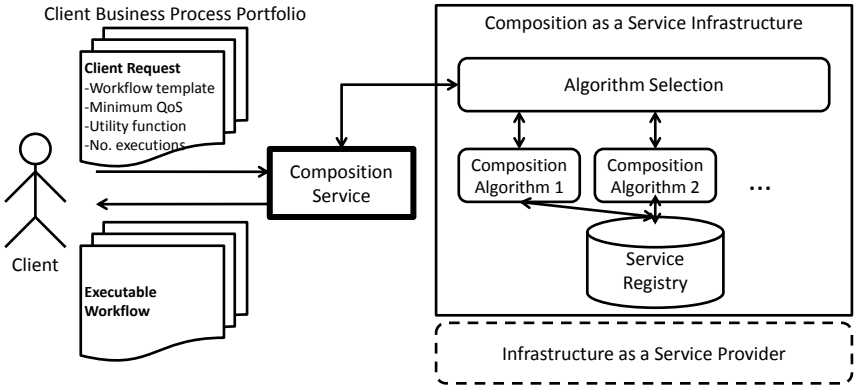


Fig. 1. Architectural overview

3 Formal Model

Our model is similar to the one presented by Zeng et al. [12] and makes the same fundamental assumptions. QDSC starts from an abstract workflow W. Every task of W is associated with a set of services which fulfill the required functionality. Those services expose different non-functional properties. We denote the set of relevant quality properties by A and by $QoS(s, a)$ the value of attribute $a \in A$ for service s. A binding is a function that maps every workflow task to one service in its associated set. The selected binding will determine the aggregated quality properties of the workflow as a whole. We denote by $QoS(W, B, a)$ the value of attribute a for workflow W and binding B. Zeng et al. [12] have shown how to map QoS values to the interval $[0, 1]$ such that 1 corresponds to best quality. Hence, we have $QoS(W, B, a) \in [0, 1]$. Depending on the attribute type, different aggregation functions must be used. Dumas et al. [7] classify QoS attributes into additive, multiplicative, and attributes whose value is aggregated over the critical path. We consider attributes where the value is aggregated as sum (e.g. response time) or as product (e.g. reliability) over the tasks on the (previously known) critical path.

In QDSC, the goal is to find a binding for a workflow such that *(i)* certain minimum requirements on the quality of the composite workflow are respected, and *(ii)* a user-defined measure of optimality on the quality attributes of the workflow is optimized. Assuming that the attributes in A are ordered ($A = \{a_1, a_2, \ldots, a_n\}$), we can express the quality requirements on the composite workflow as vector $\overrightarrow{r} = (r_1, \ldots, r_n)$. A valid binding B must satisfy (1).

$$\forall i \in \{1, \ldots, n\} : QoS(W, B, a_i) \geq r_i \tag{1}$$

The ranking between different admissible bindings depends on the preferences of the user. Some users will prefer having a lower response time even if this does mean additional invocation costs, for other users it may be the inverse. Users

specify their preferences via a vector of weights $\overrightarrow{w} = (w_1, \ldots, w_n)$ where the sum over all components is 1: $|\overrightarrow{w}| = 1$. We define the utility of a binding B:

$$Utility(W, B, \overrightarrow{w}) = \sum_{i \in \{1,\ldots,n\}} w_i QoS(W, B, a_i) \tag{2}$$

For a fixed set of available services, requirements and preferences, we define the relative quality of a binding B by comparison with the optimal binding B_{opt}:

$$relQuality(B, W, \overrightarrow{r}, \overrightarrow{w}) = \frac{Utility(W, B, \overrightarrow{w})}{Utility(W, B_{opt}, \overrightarrow{w})} \tag{3}$$

We assume that workflows are associated with an expected number of executions (during a specific time period) $nExec$. In our model, the number of executions determines the relative importance between workflows in the same set. In summary, a composition request CR is defined by the tuple $CR = (W, \overrightarrow{r}, \overrightarrow{w}, nExec)$. Clients submit sets of composition requests $crSet = \{CR_i\}$ and obtain a set of pairs $resultSet = \{(CR_i, B_i)\}$ with corresponding bindings for every request. The relative quality of a result set is the weighted average over the relative quality of all included bindings weighted by the number of executions:

$$relQuality(resultSet) = \frac{\sum_i relQuality(B_i, W_i, \overrightarrow{r}_i, \overrightarrow{w}_i) \cdot nExec_i}{\sum_i nExec_i} \tag{4}$$

4 Related Work

Among the most popular approaches for QDSC are integer linear programming and genetic algorithms. We will use these two approaches in different configurations for evaluating our dynamic selection algorithm. An **Integer Linear Program (ILP)** consists of a set of variables, a set of linear constraints and a linear objective function. After having translated the QDSC problem into this formalism, specific solver software such as CPLEX [2] can be used. Examples for this approach include the work by Zeng et al. [12] and Ardagna et al. [3]. Canfora et al. [5] introduced **Genetic Algorithms (GA)** for QDSC. Individuals of the population correspond to different bindings, their genes to the workflow tasks and the possible gene values to the available services. While GAs do not guarantee to find the optimal solution, they can be more efficient than ILP-based methods (which have exponential worst-case time complexity). By tuning parameters like the number of iterations, the probability of finding a close-to-optimal solution can be improved. Various other approaches have been applied to QDSC. Many of them offer specific parameters for trading result quality for lower running time (e.g. [6,11]). Such parameters can be leveraged by our selection algorithm for reducing the composition effort for low-priority workflows.

5 Approach for Selecting Composition Algorithms

In this section we will present an algorithm that maps composition requests to composition methods. The goal is to guarantee an average quality for the result

set while minimizing the composition cost. In Sect. 5.1, we describe a preliminary filtering for composition methods, in Sect. 5.2 the selection algorithm.

By $Methods$ we designate the set of composition methods. Every method refers to a specific algorithm with a specific parameter setting (e.g. genetic algorithm with population size 50 chromosomes and 100 generations). For selecting between different methods, we characterize them by the delivered average relative quality (see (3)) and invocation cost. However, the behavior of composition methods depends on the properties of the composition request (e.g. the running time of an ILP method correlates with the number of workflow tasks). We assume that requests can be classified such that the behavior does not vary too much for requests within the same class. $RequestClasses$ designates the set of classes, $class(cr)$ the class of a request cr. We characterize methods for specific classes using the functions $Ecost$ (expected cost) and $ErelQ$ (expected relative quality)—data can be gained by experiments with representative request sets:

$$Ecost : Methods \times RequestClasses \longrightarrow \mathbb{N} \tag{5}$$

$$ErelQ : Methods \times RequestClasses \longrightarrow [0, 1] \tag{6}$$

5.1 Initialization: Filtering Composition Methods

During initialization, each request class is assigned to a set of recommended composition methods. The result is the function

$$efficientMethods : RequestClasses \rightarrow \mathcal{P}(Methods) \tag{7}$$

Initially, all methods are considered efficient for all request classes. Then, two filtering steps are performed for each request class separately based upon the experimental data. First, composition methods have to be filtered out that risk to produce workflows of too low quality. We only consider average quality during our dynamic selection, therefore this step is important—having single bindings of very bad quality within the result set may dissatisfy clients even if the average quality is good. Further, methods can be filtered out for certain request classes if they are *dominated* by other methods, meaning that they have higher cost and deliver lower average quality. Filtering out dominated methods diminishes the search space for the dynamic selection and improves therefore the efficiency.

5.2 Mapping Composition Requests to Composition Methods

Every time that a new set of composition requests is submitted by the client, the requests in the set have to be mapped to composition algorithms based upon their relative importance and request class. This mapping has to be done efficiently since the mapping time adds as overhead to the total processing time. Our goal is to minimize the processing cost of the request set while the minimum requirements on the average quality must be met. We will show how our mapping problem can be reformulated as multi-choice 0-1 knapsack problem (MCKP) [9].

Algorithm 1. Select and execute composition methods for request set

```
1: function TreatRequestSet(crSet, efficientMethods, Ecost, ErelQ, tQ)
2:     // Transform selection problem into multi-choice 0-1 knapsack
3:     weightLimit ← 0
4:     for all cr = (W, r⃗, w⃗, nExec) ∈ crSet do
5:         optM(cr) ← argmax_{m∈efficientMethods(class(cr))}(ErelQ(m, class(cr)))
6:         mckItems(cr) ← ∅
7:         for all m ∈ efficientMethods(class(cr)) \ {optM(cr)} do
8:             costSavings ← Ecost(optM(cr), class(cr)) − Ecost(m, class(cr))
9:             qualityLoss ← ErelQ(optM(cr), class(cr)) − ErelQ(m, class(cr))
10:            newItem ← (m, qualityLoss · nExec, costSavings)
11:            mckItems(cr) ← mckItems(cr) ∪ {newItem}
12:        end for
13:        weightLimit ← weightLimit + nExec · (ErelQ(optM(cr), class(cr)) − tQ)
14:    end for
15:    mckSelected ← approximateKnapsack(crSet, mckItems, weightLimit, ϵ)
16:    // Use approximated solution and call corresponding composition methods
17:    resultSet ← ∅
18:    for all cr ∈ crSet do
19:        if mckSelected(cr) =⊥ then
20:            binding ← Execute(optM(cr), cr)
21:        else
22:            (m, qualityLoss, costSavings) ← mckSelected(cr)
23:            binding ← Execute(m, cr)
24:        end if
25:        resultSet ← resultSet ∪ {(cr, binding)}
26:    end for
27:    return resultSet
28: end function
```

This problem is NP-hard but can be approximated efficiently using a *fully polynomial time approximation scheme (FPTAS)*. Such an approximation scheme guarantees polynomial running time and a close-to-optimal solution. If the optimal utility value for a given problem instance is P_{opt}, then the approximation scheme finds a solution with utility value at least P such that $P_{opt} - P \leq \epsilon \cdot P_{opt}$ where ϵ can be chosen. The running time grows polynomial in $\frac{1}{\epsilon}$ and in the size of the problem. We use the MCKP FPTAS by Lawler [9] for our implementation.

Alg. 1 is executed every time a client submits a set of composition requests. It takes as input the submitted request set $crSet$, the set of recommended methods for every request class $efficientMethods$, the characteristics of the available methods $ErelQ$, $Ecost$, and the targeted relative quality of the result set tQ. In a first phase, the algorithm reformulates the problem of selecting optimal methods for every composition request as MCKP. The classes correspond to the different requests that have to be treated. Items within a specific class are associated with composition methods. Selecting an item for a class symbolizes the choice of the associated composition method for treating the request corresponding to that class. The item weight corresponds to the *quality loss* in comparison with the

optimal method, weighted by the number of executions. The total weight limit is proportional to the total number of executions of all workflow templates in the request set. It integrates the distance between target quality $tQ \in [0, 1]$ and the expected quality of the best method for every request. We want to minimize the processing cost, a solution to the MCKP is optimal once it maximizes the aggregated profit. Therefore, item profits correspond to *cost savings* that can be realized by choosing the associated method instead of the optimal one.

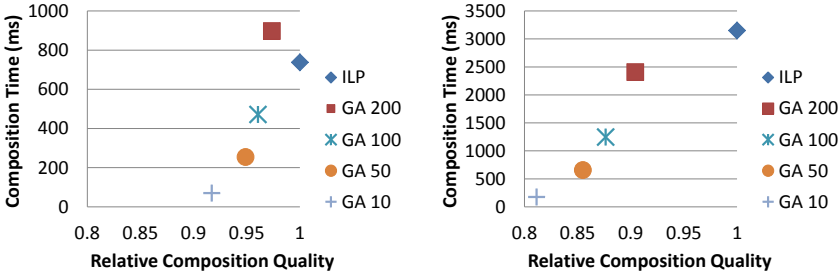
The item associated with the optimal method has weight and profit 0. This is equivalent to selecting no element in the class. Therefore, we do not integrate these items and interpret an empty selection for a class as selection of the optimal method for the corresponding request. The algorithm uses the auxiliary function approximateKnapsack which implements the FPTAS proposed by Lawler [9]. The functions takes as input the set of item classes, the sets of items for every class, the weight limit and the accuracy ϵ (Lawler's algorithm works with integer weights hence we round weights to percent). It returns a function that assigns classes to selected items or to $\perp$ if no item was selected. The algorithm uses the auxiliary function Execute(m, cr) which executes m on cr and returns the produced binding. The set of pairs between bindings and requests is returned.

Example 1. Let $crSet = \{cr1, cr2\}$ with $cl1 = class(cr1)$ and $cl2 = class(cr2)$, we have $nExec = 10$ for both requests. Assume that methods $m1$ and $m2$ are efficient for $cl1$ with $ErelQ(m1, cl1) = 0.9$, $ErelQ(m2, cl1) = 0.8$, $Ecost(m1, cl1) = 10$, and $Ecost(m2, cl1) = 5$. Only $m1$ is efficient for $cl2$ with $ErelQ(m1, cl2) = 0.95$ and $Ecost(m1, cl2) = 5$. Our algorithm generates item set $\{(m2, 1.0, 5)\}$ for knapsack class $cl1$ and $\emptyset$ for $cl2$. The weight limit is 2.5 for $tQ = 0.8$.

6 Experimental Evaluation

In this section, we experimentally evaluate our dynamic selection approach. In subsection 6.1, we benchmark two classic algorithms for QDSC—integer linear programming and genetic algorithms—in different configurations for different classes of composition queries. The experimental data we obtain in subsection 6.1 forms the input for our selection algorithm that we evaluate in subsection 6.2.

We implemented a test suite in Java that randomly generates composition requests including workflow templates and available services. We treat workflows with between 5 and 45 tasks. We considered 8 quality attributes for services: two additive attributes that depend on all tasks, two that depend only on critical tasks, two multiplicative attributes that depend on all tasks, and two that depend only on critical tasks. The QoS properties of services were chosen with uniform random distribution. We considered 50 functional categories and generated 100 services for every category. Workflow tasks were randomly assigned to functional categories. The probability that a task belongs to the critical path for one of the quality attributes that depend only on critical tasks was 50%. The quality weights were chosen randomly as well as the quality requirements which were chosen with uniform distribution between 0.01 and 0.5. The number of workflow executions for every single request was chosen out of a Pareto distribution as



(a) Workflows with 5 to 15 tasks (b) Workflows with 31 to 45 tasks

Fig. 2. Characteristics of composition methods for different request classes

motivated before. For implementing the ILP based algorithm, we used IBM ILOG CPLEX 12.1 [2] as solver. We set the thread count to 1 and used the default parameters otherwise. For the GA composition approach, we use the same Java libraries and settings as Canfora et al. [5]. However, we vary the number of generations between 10 and 200. The approximation algorithm for the MCKP was implemented in Java as well. All experiments were executed on a 2.53 GHz Intel Core Duo processor with 2.5 GB RAM running Windows 7.

6.1 Benchmarking and Filtering Composition Methods

In this section, we characterize different configurations of the two composition algorithms. We partitioned requests into 3 classes, based upon the number of workflow tasks ($5-15$, $16-30$, and $31-45$ tasks). Note that a more fine-grained partitioning could additionally consider different request properties like the strength of the quality requirements. For every class we generated 100 test cases (corresponding to a randomly generated registry and workflow request). We executed every method 10 times for every test case and take the arithmetic average execution times. Fig. 2 shows the characteristics of different composition methods within the cost-quality space. For determining the relative quality, we compared with the optimal solution produced by ILP. We benchmark GA with different numbers of generations (10, 50, 100, and 200).

We make the following observations. *i)* The only case of dominance between different methods occurs for small workflows: ILP dominates GA 200 since it delivers better quality at lower cost. *ii)* The running time of the GA-based methods is approximately proportional to the number of generations and the average number of tasks. The growth of composition time for the ILP approach is over-proportional such that "GA 200" is not dominated anymore for large workflows. *iii)* For the same number of generations, the relative quality of the genetic algorithms slightly decreases when the number of workflow tasks grows. *iv)* The standard deviation was always below 1% (of average value) for the relative quality while reaching up to 9% for the running time.

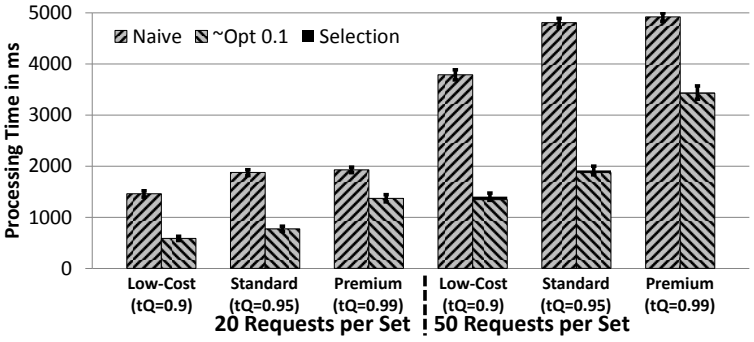
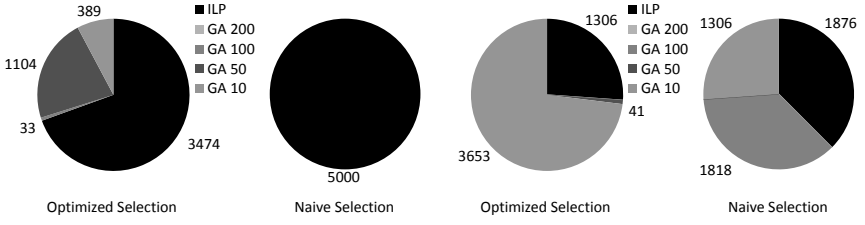


Fig. 3. Comparison of naive and optimized selection approaches



(a) Premium composition (tQ=0.99) (b) Low-Cost composition (tQ=0.9)

Fig. 4. No. selections for composition methods (100 times 50 requests per set)

6.2 Evaluating Selection Algorithms

We compare our near-optimal selection approach with $\epsilon = 0.1$ to a naive algorithm. Both variants work with the data from the previous subsection. The naive approach selects for a given request cr and target average quality tQ the composition method $m \in efficientMethods(class(cr))$ which has minimum expected cost among the methods that deliver the required target quality $ErelQ(m, class(cr)) \geq tQ$. Fig. 3 shows the results of our comparison with 5% confidence intervals. We generated and solved 100 request sets and report the arithmetic average times. We compare the two selection algorithms for request sets of different size (20 and 50 requests) and different quality requirements (from $tQ = 0.9$ to $tQ = 0.99$). Our criterion is the total processing time per request set. For the near-optimal selection strategy, we divide the time into time required to map requests to algorithms and time required for executing the selected algorithms. We observe the following. *i)* The processing time increases for higher number of requests and increasing quality requirements. *ii)* The time for the selection phase accounts only for between 0.2% and 4% of the total processing time for the near-optimal selection. *iii)* Our selection approach takes only 40% (37%) of the time of the naive approach for 20 requests per set (50 requests per set) and for $tQ = 0.9$, 40% (41%) for $tQ = 0.95$, and 71% (70%) for $tQ = 0.99$.

We verified that the relative quality of the result sets (rounded to percent) produced by our approach always met the specified bounds. Fig. 4 shows how many requests the different selection approaches assigned to the different composition methods. ILP is the dominant method for high target quality ($tQ = 0.99$) while GAs dominate for lower quality ($tQ = 0.9$). Our approach is able to select more low-cost composition methods which explains the higher efficiency.

7 Conclusion

In this paper, we classify existing composition algorithms in terms of running cost and expected quality. We dynamically assign different workflow templates to different composition algorithms based upon template structure and relative importance. Our experimental evaluation shows that our approach reduces composition cost significantly while introducing little overhead.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "SOSOA: Self-Organizing Service-Oriented Architectures" (SNF Sinergia Project No. CRSI22_127386/1).

References

1. Amazon elastic compute cloud, `http://aws.amazon.com/ec2/`
2. Ibm ilog cplex, `http://www.ibm.com/software/products/de/de/ibmilogcple/`
3. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering, 369–384 (2007)
4. Blake, M., Tan, W., Rosenberg, F.: Composition as a service (web-scale workflow). Internet Computing 14(1), 78–82 (2010)
5. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: An approach for QoS-aware service composition based on genetic algorithms. In: Conf. on Genetic and Evolutionary Computation, pp. 1069–1075. ACM (2005)
6. Comes, D., Baraki, H., Reichle, R., Zapf, M., Geihs, K.: Heuristic Approaches for QoS-Based Service Selection. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 441–455. Springer, Heidelberg (2010)
7. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate Quality of Service Computation for Composite Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
8. Koch, R.: Das 80-20-Prinzip. Campus-Verl. (1998)
9. Lawler, E.: Fast approximation algorithms for knapsack problems. In: 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 206–213. IEEE (1977)
10. Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., Dustdar, S.: Towards composition as a service-a quality of service driven approach. In: Int. Conf. on Data Engineering, pp. 1733–1740. IEEE (2009)
11. Trummer, I., Faltings, B.: Optimizing the Tradeoff between Discovery, Composition, and Execution Cost in Service Composition. In: Int. Conf. on Web Services (2011)
12. 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)

A Service Model for Development and Test Clouds

Debdoot Mukherjee[1], Monika Gupta[1],
Vibha Singhal Sinha[1], and Nianjun Zhou[2]

[1] IBM Research – India
{debdomuk,monikgup,vibha.sinha}@in.ibm.com
[2] IBM TJ Watson Research Center
jzhou@us.ibm.com

Abstract. A Development & Test Cloud (DTC) enables IT service enterprises to host standardized configurations of just about any tool-set on cloud – the hosted software need not be designed for multi-tenancy and they may come from a multitude of vendors. However, since most enterprise software are available only under perpetual licenses, DTCs cannot become truly pay-per-use – customers of a DTC have to upfront purchase software licenses. This paper proposes a service model for a DTC vendor wherein the vendor purchases software licenses and recovers the cost from its clients based on their period of usage. Our model allows the vendor to maximize returns from a purchased license by using it in multiple projects separated in time. We set up an optimization problem to decide how best a DTC operator can invest in buying software licenses such that it gets maximum opportunity to resale purchased licenses. We conduct empirical studies to validate the feasibility and usefulness of our approach. Also, we enlist characteristics of tool-sets that make them profitable for the DTC vendor.

1 Introduction

Even as the economy recovers from the downturn, IT services enterprises continue to cut down on all forms of operational expenditure so that receding profit margins of services contracts do not affect their balance sheets adversely. All large companies with massive, geographically distributed workforces are upset with burgeoning IT support costs, under-par utilization of hardware resources and sub-optimal management of software licenses. Again, they wish to improve productivity of their personnel by equipping them with the latest developer toolsets, which often require advanced hardware configurations to run effectively. A cloud based service delivery environment addresses the above issues and offers many interesting possibilities toward shaping the next generation IT services enterprise. Using a high performance cloud platform for hosting development and test environments, not only reduces IT infrastructure and support costs drastically but also helps to streamline delivery by provisioning pre-configured, standardized toolsets and leads to significant improvements in developer productivity. Moreover, it empowers lines-of-businesses (LOBs) in an enterprise with extreme agility to contend changing market realities; they can easily scale up or scale down their IT infrastructure because they do not incur any capital expenditure to own hardware/software but simply pay a price as per their usage.

A *Development & Test Cloud (DTC)* comes across as a unique offering specifically designed to ensure application development and maintenance activities can move

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 523–531, 2011.

to the cloud. A DTC is a service environment that can automatically provision pre-configured, integrated sets of software on hardware configurations chosen by the user (See our technical report [5] for details on DTC use-cases, architecture and benefits). It can turn-around defect-free, ready-to-use development and testing environments within minutes; thus results in faster time-to-market of deliverables as well as lower idle times for project personnel. Initial pilots [7,5] of Development & Test Clouds have shown drastic reduction in provisioning overheads, elimination of configuration defects and improvement in developer productivity. However, current DTC implementations force their customers to upfront purchase licenses for most software. Very seldom, one finds software being rented in a pay-per-use manner – mostly limited to cases where the software comes from the cloud vendor itself. This poses a serious issue for enterprise application development since service engagements typically leverage software coming from a multitude of technology vendors. Clearly, in such a scenario, the promises of lower software costs and easier scaling of usage levels will not be realized – the licenses have to be purchased at the same rates as they are available for lifelong standalone use. Hence, the adoption of DTCs may be hit. In fact, a 451 Group report[1] and Lori [4] point out that old models of software licensing are entirely incompatible with cloud computing environments and this fact proves to be a severe roadblock for greater cloud adoption.

In this paper, we propose a service model whereby the DTC vendor purchases all software licenses and recovers the cost from its clients based on their period of usage. Our model allows the vendor to maximize returns from a purchased license by using it in multiple projects separated in time. We set up an optimization problem to decide how best a DTC operator can invest in buying software licenses such that it gets maximum opportunity to resale purchased licenses. Also, we empirically study characteristics of tool-sets that can lead to profitable DTC hosting.

The main contributions of the paper include:

1. Design of a service model for a DTC operator that optimally transforms costs incurred in buying software licenses to pay-per-use prices. The model guides the DTC vendor to purchase a set of software so that it can maximize returns (Section 2).
2. Empirical evaluation of the DTC service model to demonstrate its feasibility and an evaluation of heuristics that can decide whether a toolset is a preferred candidate for stocking in a DTC (Section 3).

2 DTC Service Model

We propose a service model for a Development & Test Cloud (DTC) offering that helps a DTC vendor to decide which software *appliances*[2] to stock on a DTC and how to

[1] http://www.informationengineer.org/2010/02/06/the-451-groups-cloud-computing-outlook-for-2010.html

[2] An *appliance* is a common set of software that when installed and tuned to a certain configuration can support development and testing activities across service engagements of a particular kind. For example, SOA engagements may always use an integrated appliance consisting of certain software from Websphere stack.

price them (per unit usage) in order to run the operations most profitably. We suggest the following scheme whereby the DTC provider purchases licenses and the end-users pay a just fee per their usage:

1. A DTC provider purchases licenses of different kinds of software and collects the same in *license pools*.
2. Every time an appliance is provisioned for a client, each software in the appliance is assigned with a license available in its pool. A fee for license usage, which computed as per the proposed model, is bundled into the appliance cost. The user may have to pay a premium price (higher than the fixed rate, possibly close to the license cost) only if there are no licenses available in the pool for a particular software.
3. The licenses are returned back to the pool after the appliances get de-provisioned.

Now, the DTC vendor wishes to keep just enough licenses in the pool to serve demand for appliances at any point of time. Also, it is desirable that a license once purchased finds use in several projects over the course of time. Greater license reuse across projects separated in time will bring down the fees paid by the end-user and enhance the DTC vendor's profitability. Thus, we have the following problem:

> **Problem Definition:** *How can the DTC enterprise effectively invest a fixed amount of capital to buy licenses of software present in common appliances and then appropriately price the appliances in a pay-per-use model, based on available demand forecasts?*

At first sight, one may relate the above problem to the standard problem of inventory management [6] – how much goods do you stock in your inventory so that you do not run out of materials when you need them? Turns out, the drivers for these two problems are quite different! The reason for maintaining an inventory of goods is to avoid shortage costs. The time taken to refill stock after placing an order is generally significant, thus replenishment is ordered in advance. For purposes of our problem, it can be assumed that an order for a new license is served instantaneously and thus, shortage costs are not applicable. However, the strategy of purchasing licenses every time a provisioning request arrives is not optimal. We want licenses to be reused to increase profitability; therefore, we wish to invest in purchase of only those licenses for which we expect sufficient future demand. Again, buying licenses in advance may help save money if price increases are common. We set up an optimization problem to determine the number of licenses of each kind of software that should be purchased in order to maximize the return on investment for the DTC vendor. Solving such a problem also helps us ascertain the price that can be set for each appliance or software usage.

We consider a finite set, $A = \{\alpha_1, \alpha_2, \ldots \alpha_n\}$, composed of appliances that are sought after in typical service engagements. An appliance is a set of software, $\alpha_i = \{S_1, S_2, \ldots S_m\}$, with pre-built configurations commonly used in a particular form of engagement. It is assumed that engagements using a certain appliance, α_i, have similar duration. If not, new appliances are created in A such that we enforce the standard deviation of durations of all projects using a single appliance to be small. Furthermore, we conjecture that solutioning teams in service enterprises have the engagement pipeline data, which gives demand forecasts for each engagement type.

Our service model works with the following inputs:
Δ_i : Mean project duration of engagements using α_i
D_i : Demand for α_i as a function of time
T : Time period for which all price calculations are made
F : Capital that may be invested in license purchase during time interval $[0, T]$

We introduce the notion of a license unit for an appliance. One license unit for α_i includes one license each for every software S_j contained in it. Again, in our model, time t can take up discrete values in the time interval $[0, T]$. In practice, a time period T of a quarter or a year may be discretized in terms of the different weeks or months in them. Suppose,
χ_i : Number of license units of α_i purchased at $t = 0$
$L_i(t)$: Number of license units of α_i available in pool at time t
$U_i(t)$: Number of license units of α_i taken out of pool for use in projects starting at time t

As mentioned before, at any point of time, license units move out from the pool and get assigned to projects. Again, the license pool gets augmented by licenses from projects that have just ended. Thus, we can write:

$$L_i(t+1) = \begin{cases} L_i(t) + U_i(t - \Delta_i) - U_i(t), & t \geq \Delta_i \\ L_i(t) - U_i(t), & 0 < t < \Delta_i \\ \chi_i, & t = 0 \end{cases} \tag{1}$$

Solving the above recurrence relation we get:

$$L_i(t) = \begin{cases} \chi_i - \sum\limits_{t'=t-\Delta_i}^{t-1} U_i(t'), & t \geq \Delta_i \\ \chi_i - \sum\limits_{t'=0}^{t-1} U_i(t'), & 0 < t < \Delta_i \end{cases} \tag{2}$$

Now, we can only assign license units for an appliance only if there is demand for that appliance and there exist free units in its pool. Therefore, $U_i(t) \leq min[L_i(t), D_i(t)]$. The cost C_i of a license unit for appliance α_i is calculated as sum of license prices for each software $S_j \in \alpha_i$. Now, the returns derived by the DTC each time an appliance is used in a project are directly proportional to the cost of the appliance. We formulate an optimization problem in Equation 3 that seeks to maximize such returns. The constraints are: license purchases are limited to as many units as are permitted by the available capital, F; and existence of both demand and unassigned license units.

$$\begin{aligned} max. \sum_{\forall i} C_i \sum_{t=1}^{T} U_i(t) \\ s.t. \quad \sum_{\forall i} C_i \chi_i \leq F \\ U_i(t) \leq L_i(t) \ \forall i, t \in \{1, 2, \ldots T\} \\ U_i(t) \leq D_i(t) \ \forall i, t \in \{1, 2, \ldots T\} \end{aligned} \tag{3}$$

The optimization problem contains $n(T + 1)$ variables; where n is the number of appliances and T is the upper limit of the discrete time interval that we consider. For example, the variables for α_i are: $\chi_i, U_i(1), U_i(2), \ldots, U_i(T)$. All variables take up integer values only; so the problem is NP-complete like all integer programming problems.

Pricing Appliances: Once we solve the optimization problem in Eqn. 3, we can ascertain the price for using an appliance per unit time. This is computed by amortizing the total costs spent on licenses and configuration over the period of time when instances of that appliance find use. Additionally, support charges may be bundled; if support is important.

$$\text{Price of } \alpha_i \text{ per unit time} = \frac{\text{Configuration Cost} + C_i\chi_i}{\Delta_i \sum_{t=1}^{T} U_i(t)} \tag{4}$$

3 Experiments and Results

In this section, we experimentally show that our model is tractable and also validate its usefulness. Further, we design statistical tests to derive indicators that can help a DTC vendor to choose profitable appliances to stock in the cloud. First, we discuss the data and the setup used in our experiments and then we present the emprirical studies.

3.1 Experimental Data and Setup

Our service model is designed to source its inputs from the forecasted deal pipeline and the archives of financials for recent projects. Since, such data is highly confidential we resort to data synthesis to create data-sets for the purposes of this paper. Wherever possible we try to mimic real samples coming from services enterprises which engage in application development and maintenance. Table 1 shows how the different levers in our model are simulated for our evaluation (See [5] for further details).

To solve the integer linear program described in Equation 3, we used an IP solver engine called *Gurobi* available within a commercial optimization and simulation package[3]. All experiments were run on a machine with a configuration – 2.16 GHz, 2 GB RAM, Windows XP. The entire experimental data-set as well as the solution set-up in MS-Excel is available for download[4].

3.2 Study 1: Feasibility and Effectiveness of the DTC Service Model

Goals and Method: This study seeks to demonstrate the feasibility and validity of our approach on synthesized practical data-sets. We solve the optimization problem in Equation 3 on several data-sets that are produced as per Table 1. We vary the number of appliances and the available capital to generate different data-sets for this study.

After solving the optimization problem, we obtain values for the number of license-units to be purchased, χ_i, as well as the weekly allocations, U_i, for each appliance, α_i. Now, in order to quantify the value of our approach stemming from license reuse across projects, we compute the `Overall-Potential` score defined below:

$$\texttt{Overall-Potential (\%)} = \frac{\sum_{\forall i} C_i \sum_{t=1}^{T} U_i(t) - \sum_{\forall i} C_i\chi_i}{\sum_{\forall i} C_i\chi_i} \times 100$$

[3] http://www.solver.com

[4] http://researcher.watson.ibm.com/researcher/files/in-debdomuk/dtc-data.zip

Results and Analysis: Table 2 list various details of the solutions obtained to our optimization problem for 5 data-sets with different combinations of available capital, F and number of appliances, n.

Every run of the IP solver completes within 15 seconds and ends up with a globally optimal solution. The fact that the *Left-Over capital* ($F - \sum_{\forall i} C_i \chi_i$) is always less than the lowest license-unit cost available in the data-sets indicate the no more license allocations are possible beyond what is obtained in the solutions. *Unsatisfied Demand* points to cases where no allocations where possible despite presence of demand. Non-zero values for unsatisfied demand show that there were no trivial solutions to the IP.

Table 1. Data-set Parameters

Time Period, T	54 weeks
Weekly Demand Function, $D_i(t)$	Gaussian data sample with μ, σ from [5,10], [0,5] respectively.
License-Unit Cost, C_i	Random from [$500, $3000].
Project Duration Δ_i **(weeks)**	Uniformly from {2, 4, 6, 8, 12, 16, 20, 24, 32, > 54}

Table 2. Validating Feasibility and Effectiveness

n	**20**	**50**	**100**	**100**	**100**
F	**$0.5 mn**	**$5 mn**	**$2 mn**	**$5 mn**	**$10 mn**
Variable (= $n[T+1]$)	1100	2750	5500	5500	5500
Time taken (in secs)	5.45	9.23	13.6	12.7	13.28
Total Objective	11088550	27654900	22558400	35874900	49131000
Left-Over Capital(in $)	200	50	0	250	0
Unsatisfied Demand (%)	23	16.6	66.9	46.1	24.4
Overall-Potential (%)	2117	453	1028	617.5	391

Our approach yields high values of `Overall-Potential` in all cases. This underscores the significant financial benefits that can stem from optimal license management and license resale in DTC environments. `Overall-Potential` subsides as we add availability of capital in our experiments. With less capital, our solution always allocates licenses to the most profitable appliances. When we add more capital, the solution serves relatively less profitable appliances, so the average potential decreases.

3.3 Study 2: Identifying Profitable Appliances

Goals: Optimal solutions to Equation 3 indeed help a DTC provider to decide the correct amount of stock to keep for each appliance. However, in many practical scenarios, one needs to decide whether an appliance is a *good* candidate to stock on cloud without complete information of the other appliances in contention. It is often desirable to be able to independently form an opinion on an appliance's *profitability* simply by studying its characteristics – either in isolation or with respect to high level trends observed in other appliances. This study aims to identify some characteristics of an appliance that can possibly serve as indicators of profitability. We organize the study in terms of 3 research questions; each of which tries to evaluate a parameter in our service model on whether its value can throw some light upon the preferredness of the appliance.

- **RQ1:** Investigate the effect of *project duration*, Δ_i, on an appliance's profitability.
- **RQ2:** Investigate the effect of *cost of a license-unit*, C_i, on an appliance's profitability.

– **RQ3:** Investigate the effect of *demand function*, D_i, on an appliance's profitability.

Method: We empirically evaluate the dependence of different parameters in our service model, Δ_i, C_i and D_i, on a quantitative measure of an appliance's profitability – the *Profit-Potential (PP)* of an appliance:

$$\texttt{Profit-Potential (\%)} = \frac{C_i \sum_{t=1}^{T} U_i(t) - C_i\chi_i}{C_i\chi_i} \times 100$$

In the context of any solution to our service model, the `Profit-Potential` for an appliance signifies the degree of license re-sale that can be effected for the appliance in the solution. Of course, greater the re-sale, higher is the profit. Now, for each RQ, we empirically measure the correlation of the parameter in question with `Profit-Potential`. We analyse every parameter in isolation, i.e., while studying the effect of a parameter, we generate the data such that the other parameters take identical values in all appliances being monitored. Next, we perform statistical tests to decide whether the correlation is significant at 0.01 level. If the correlation is found to be significant, then we conclude that the parameter is indicative of an appliance's profitability.

Statistical Analysis and Results: To address all 3 research questions, we create samples of data pertaining to 100 appliances. Table 3 shows the values taken up by different model parameters in the three cases and also summarizes the experimental results. For **RQ1**, we measure the correlation between the project-duration and the observed `Profit-Potential` for appliances to be -0.632, which is significant at 0.01 level. Since, the project duration demonstrates a strong negative correlation we can conclude that lower project duration means greater profitability for the DTC vendor. The result is intuitively true since lower the project duration, greater are the chances of license re-sale. For **RQ2**, we measure the correlation between the cost of a single license-unit and the observed `Profit-Potential` for appliances to be -0.17. Such a value for correlation is not significant, thus we cannot make conclusive statements about the effect of license-unit cost on appliance's profitability. For **RQ3**, we wish to determine whether uniformity in demand is good for profitability of an appliance. Thus, we measure the correlation between the standard-deviation of weekly demand for appliances across the one year period and the observed `Profit-Potential` for appliances to be -0.382, which is significant at 0.01 level. Since, lower deviation in demand leads to greater profitability, we can conclude that uniform demand for an appliance augurs well for the DTC vendor.

In summary, appliances that find use in projects having smaller durations and appliances having uniform demand functions are better prospects for stocking in a DTC.

4 Related Work

License management for cloud computing environments has been recognized as an open problem in literature. The 451 group report[3] and [4] identified restrictive license terms to be a major threat to cloud adoption. Dalheimer et. al. [2] propose GenLM, a license management framework that allows ISVs to manage their license usage in a distributed

Table 3. Identifying Profitable Appliances

	RQ1	**RQ2**	**RQ3**
Weekly Demand, $D_i(t)$	10	10	Gaussian with μ, σ from [5, 10] and [0,5] resp.
License-Unit Cost, C_i	1000	Random from [$500, $3000]	1000
Project Duration, Δ_i	{2, 4, 6, 8, 12, 16, 20, 24, 32, 54} weeks.	6 weeks	6 weeks
Available Captial, F	$10000000	$2000000	$2500000
Time Period, T	54 weeks	54 weeks	54 weeks
Study Variable, X	Δ_i	C_i	$\sigma(D_i)$
Pearson's-Correlation(X, PP)	-0.632	-0.17	-0.382
Significance at 0.01 level	Yes	No	Yes

world. The main idea of GenLM is to attach the license not to a node or a person but to issue licenses for the input datasets, thus allowing users to buy a per-job license and run the job on any suitable resource. However, such a model is only applicable for web applications and does not suit standalone software that run on desktops.

Cacciari et. al. [1] propose elasticLM - Licence as a Service (LaaS) for creating and managing software licenses. The framework enables a user to negotiate terms and policies with a license provider service to be able to procure a license token to execute an application. [8,9] discuss issues related to implementation of license management systems for Grid environments. But, none of these address how a LaaS can be run profitably.

5 Conclusions

We note that unavailability of enterprise software under usage based pricing models can potentially affect adoption of a *Development & Test Cloud (DTC)* – a delivery platform that promises revolutionary cost advantages and efficiency improvements for an IT service enterprise. As a counter, we propose a novel service model wherein the DTC acts as a proxy to help transform software costs from perpetual licensing to pay-per-use. We suggest that a DTC can centrally buy all software licenses and maintain them in license pools. All software present in an image are assigned licenses from the pools whenever a new instance is provisioned; the licenses return back to their respective pools on deprovisioning of the instance. We set up an optimization problem that can determine how many licenses for every software should a DTC buy in order to meet the forecasted demand in the most effective yet profitable manner. Our empirical studies show that it is feasible to obtain optimal solutions to the service model. Also, we find that stocking appliances that are used in projects with a small duration and the ones that exhibit uniform demand can lead to greater profitability of the DTC vendor.

References

1. Cacciari, C., D'Andria, F., Gozalo, M., Hagemeier, B., Mallmann, D., Martrat, J., Peréz, D., Rumpl, A., Ziegler, W., Zsigri, C.: Elasticlm: A Novel Approach for Software Licensing in Distributed Computing Infrastructures. In: 2nd IEEE International Conference on Cloud Computing Technology and Science, pp. 67–74 (2010)

2. Dalheimer, M., Pfreundt, F.: GenLM: License Management for Grid and Cloud Computing Environments. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 132–139 (2009)
3. IDC and Flexera. Inc. 2010 Key Trends in Software Pricing & Licensing Survey (2010)
4. MacVittie, L.: Cloud Computing's Other Achilles' Heel: Software Licensing (2009)
5. Mukherjee, D., Gupta, M., Sinha, V.S., Zhou, N.: Development & Test Cloud: A Next Generation Service Delivery Platform. IBM Technical Report No. RI11007 (2011), http://domino.research.ibm.com/library/cyberdig.nsf/index.html
6. Silver, E., Pyke, D., Peterson, R., et al.: Inventory management and production planning and scheduling, vol. 2. Wiley, New York (1998)
7. Singh, A., Hung, E., Balepin, I., et al.: IBM Technology Adoption Program Cloud Sandbox Internal Pilot (2009)
8. Dornemann, K., Freisleben, B.: Licensing the Use of Grid Services, Citeseer (2007)
9. Dong, X., et al.: Floating License Sharing System in Grid Environment. In: 1st International Conference on Semantics, Knowledge and Grid, p. 96 (2005)

Time Based QoS Modeling and Prediction for Web Services

Leilei Chen[1], Jian Yang[2], and Liang Zhang[1]

[1] School of Computer Science, Fudan University, China
{081024012,lzhang}@fudan.edu.cn
[2] Department of Computing, Macquarie University, Australia
jian.yang@mq.edu.au

Abstract. Quality of Service (QoS) prediction and aggregation for composite services is one of the key issues in service computing. Existing solutions model service QoSs either as deterministic values or probabilistic distributions. However, these works overlooked an important aspect in QoS modeling, *time*. Most QoS metrics, such as response time, availability, are time-dependent. We believe time variation should be explicitly reflected in QoS modeling as well as aggregation. In this paper, we propose a dynamic web service QoS model to capture the time based QoS patterns, based on which QoS of composite services are aggregated.

1 Introduction

The QoS aspect of web services has attracted much attention from the research community [1,2,3,4,5,6,7]. QoS aggregation, i.e., estimating the QoS of a composite service based on the available QoS information of the component services, becomes crutial for QoS-based service selection [1,2,3,4]. Here, two issues need to be addressed: (1) how QoS of web services can be effectively modeled? (2) how the QoS of individual services can be aggregated according to the composition structure.

QoS modeling of web services in existing works can be classified as following: (1) *Deterministic QoS value* [2,3]. (2) *Probability Mass Function* (PMF) [4], which represents the probability of a QoS over its discrete values; (3) *Probability Density Function* (PDF) of some standard statistical distributions [1,5]; (4) PDF of irregular distributions [6]. However, all the above works have neglected a commonly observed phenomenon in reality: service QoSs often exhibit some time related patterns. Taking stock service as an example, usually the service is mostly accessed during trading hours and the usage of the service outside these peak hours is relatively low. As a result, within different time periods, service QoS can vary dramatically. A static QoS model is inadequate to represent service QoS with different characteristics in different time periods, and the aggregated QoS for composite service based on these models can not effectively reflect the impact of time.

Modeling and aggregating time based QoS is challenging. First of all, how can we find the *time-based QoS Change Cycle*? This cycle of QoS change presents

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 532–540, 2011.

repeating patterns on the regular basis. Secondly, how can we capture the individual time spans within the cycle, in which the QoSs exhibit different patterns? Thirdly, aggregating component QoSs need to consider both the composition structure and the distinctive characteristics for each QoS aspect. In this paper, we will tackle this challenge by establishing dynamic QoS models for web services and providing methods for QoS aggregation based on these dynamic models.

The paper is organized as follows: Section 2 discusses the related work. In Section 3, we develop the time based dynamic QoS model for web services. In Section 4, QoS aggregation based on the dynamic model is explained. Section 5 presents the simulation results and Section 6 concludes the paper.

2 Related Work

Several previous studies have addressed the problems of QoS modeling and QoS aggregation. Cardoso *et al* [1] propose a Stochastic Workflow Reduction (SWR) algorithm to compute the QoS of a composition with sequential, parallel, conditional and simple loop blocks. Jaeger *et al* [2] examine more composition patterns and Dumas *et al* [3] propose a QoS aggregation method that can handle unstructured acyclic fragments. Hwang *et al* [4] propose a probability-based QoS model where a QoS measure of an atomic or composite service is quantified as a PMF. Zheng *et al* [6] adopts a Gaussian Kernel Density estimation approach to generate the PDFs for component services, based on which, QoS aggregation is implemented to get more accurate estimations for composite services.

None of the above studies take the timing characteristics of QoS into consideration. Klein *et al* [7] address the inherent variability in the actual values of QoS at runtime. Our research is motivated by the similar concern as Klein's. However, we focus more on the details of setting up dynamic QoS models and computing aggregated QoSs for composite services.

3 Dynamic QoS Modeling for Web Services

Web services can exhibit different quality in different environmental conditions such as usage, network traffic. In many real-world cases, the change of the environmental conditions presents certain periodical patterns, which leads to the changes on the service QoSs. We use *time cycle (TC)* to represent the period when the service has different qualities which will re-appear in the next cycle.

We differentiate two kinds of QoS metrics based on the feature of their value space: continuous values and discrete values, which bring different complexities into QoS aggregation. We use PDF to model continuous QoS metrics such as response time. For discrete QoS metrics including reliability and cost, we use PMF. Moreover, as different QoS metrics always correlate in the same time period, it is preferable to integrate all QoS metrics into one dynamic model in order to capture the impact of time pattern on the service. In this work, we will consider three representative QoS metrics: response time, reliability and cost.

Definition. A Dynamic QoS Model(DQM) for a service is a tuple (TC, I, DM), where

• $TC = [T_0, T_0 + T)$ is the QoS change time cycle and T_0 is the cycle starting time; T is the length of the time cycle.

• $I = \{I_0, I_1, ..., I_{N-1}\}$ is a segmentation of TC, where $I_i = [t_i, t_{i+1})$, and $\bigcup_i I_i = TC$

• $DM =< SM_0, SM_1, ..., SM_{N-1} >$ is a sequence of QoS models in the time segments, where $SM_i = (f_{resp_i}, P_{rel_i}, P_{cost_i})$ is a vector of probability models corresponding to the segment I_i, and

▷f_{resp_i} is the PDF of response time

▷P_{rel_i} is the PMF of reliability

▷P_{cost_i} is the PMF of cost

Given the DQM of a service, the QoS probability model can be represented as a function of time, i.e., $Q(t) = (f_{resp_i}, P_{rel_i}, P_{cost_i})$, where $t \in [t_i, t_{i+1})$.

The construction of the segment models can be achieved using existing probabilistic methods that have been discussed in previous works. Here, we focus on identifying the time cycle TC of QoS and getting a reasonable segmentation of the cycle for the DQM of a service.

We regard a service combined with its executing environment as a 'QoS-generating' system. We observe the dynamic QoS displayed by the system to find the internal mechanism that generates the QoS. For this purpose, we adopt a Hidden Markov Model (HMM) [8] based method, assuming that the hidden mechanism is a state machine, each state corresponds to a certain environmental condition, within which the system exhibits stable behavior (i.e., the QoS the system generates within each state demonstrates the stable statistical properties), and state transitions have certain regularity.

A HMM is formally defined as $\lambda = (S, A, B, \Pi)$, where

• S is a finite set of states;

• $A = \{a_{ij}\}$ is the transition probability matrix, in which, a_{ij} is the probability of state i transiting to state j in the next step;

• $B = \{b_i(o)\}$ is the emission probability matrix, where o is an observation and $b_i(o)$ represents the probability of observing o when the system is in state i;

• $\Pi = \{\pi_i\}$, in which π_i is the probability of the system beginning in state i.

In this paper, we adopt two traditional HMM algorithms. The first one is Baum-Welch algorithm, i.e., given an observation sequence $O_i(i = 1, 2, ...)$ generated by a system, to get a model λ to best describe the system. And the second one is Viterbi algorithm, i.e., given a model λ and a sequence of observations O_i, to find the hidden state sequence S_i.

To model the system as a HMM, we first partition the time into equal intervals. Next, we get the QoS statistics of the service in each time interval and take them as the interval's observation. As discussed before, some QoS metrics have discrete value domains while others have inherently continuous values. In order to uniformly represent the QoS metrics in HMM, for each time interval, we compute the expectation value of each QoS metric by taking the mean of the observed values, and combine them into a QoS observation vector. The QoS observation vector of

the ith time interval is $O_i(E_{resp_i}, E_{rel_i}, E_{cost_i})$, where E_{resp_i},E_{rel_i},E_{cost_i} represent the expectation values of response time, reliability and cost respectively. In this vector, E_{rel_i} and E_{cost_i} are represented as continuous values with the sample spaces as $Dom(E_{rel_i}) = [0, 1]$ and $Dom(E_{cost_i}) = [min(cost), max(cost)]$, respectively. In this way, all the QoS metrics are represented as continuous variables uniformly. As different QoS features have quite different scales, we normalize all variables into the scale range of $[0, 1]$ according to the following formula:

$$E'_{q_i} = \begin{cases} \frac{E_{q_i} - E_q^{min}}{E_q^{max} - E_q^{min}} & \text{if } E_q^{max} - E_q^{min} \neq 0 \\ 1 & \text{if } E_q^{max} - E_q^{min} = 0 \end{cases}$$

where E_{q_i} represents the element of O_i, $E_q^{max} = max(E_{q_i})$ and $E_q^{min} = min(E_{q_i})$. So, we get a sequence of observation vectors represented as: $O'_i(E'_{resp_i}, E'_{rel_i}, E'_{cost_i})$, $i = 1, 2, ...n$, where n is the total number of the time intervals created. As now the elements of the vector have continuous values, we assume they obey Gaussian Mixture distribution. Therefore, the system is eventually modeled as a Gaussian Mixture HMM with the emission probability density function for each state defined by multivariate Gaussian distribution.

Given the observation vector sequence $O =< O'_1, O'_2, ..., O'_n >$, we use the Baum-Welch algorithm to get the HMM λ that best describes the system. Then, with λ and O, utilize the Viterbi algorithm to get the hidden state sequence. Next, self-transitive states are gradually merged, from which we get the segmentation of the state sequence. The time cycle TC will be eventually recognized from the segmentation.

With the time cycle pattern recognized, we can set up segment QoS models for each time segment and integrate them into the final DQM.

4 QoS Aggregation Based on DQM

In this section, we shall address the issue of estimating the QoS of a composite service based on the DQMs of its component services. We focus on composite services with four structured composition patterns, including sequential pattern, parallel pattern, conditional pattern and loop pattern. Three challenge issues remain: (1) determining the time cycle of a composite service; (2) estimating the QoS of the composite service if it is invoked at a certain time point; (3) obtaining the DQM of the composite service.

4.1 Estimating the Time Cycle Length of Composite Services

The cycle length of a composite service (denoted as T) is computed as the least common multiple (LCM) of all the cycle lengths of its component services. We then standardize the DQMs of all the component services. That is, expanding the cycles of all the component services to the standard T. By doing this, all services under consideration have the same cycle length, and the time can be represented uniformly in all the DQMs.

4.2 Estimating QoS of Composite Services for Single Invocation

Problem Statement: Suppose composite service CS is recursively constructed using the following basic composition patterns: sequential, parallel, conditional, and loop, its component services are $ws_1, ..., ws_M$, and the respective standardized DQMs are $DQM_1, ..., DQM_M$, given an invocation time t_0, ($T_0 \leq t_0 < T_0+T$, where T_0 is the standardized cycle starting time and T is the standardized cycle length), estimate the QoS of CS for this invocation.

The impact of a single component service ws_i on the overall QoS of CS correlates with the time when ws_i is invoked. So, we need to estimate the invocation time for every ws_i. Furthermore, as the response time of ws_i affects the invocation time of all services that are invoked after it, the estimation of invocation time have to be conducted in one direction, i.e., traversing the composition process step by step and from the beginning to the end. We identify two set of parameters in relation to the reduction of a composition pattern: input QoS parameters and output QoS parameters. We use $SubCS$ to represent a sub composition structure of CS. Given the input QoS parameters of $SubCS$, we compute its output parameters according to its composition pattern, which will then be transferred to its direct successor as the inputs. The reduction algorithm will be repeatedly executed from the very beginning and the effect of each component service is gradually reduced and merged till the end of CS's process.

Input QoS parameters of a $SubCS$ include: (1) $startTime$, a continuous variable that represents the time when $SubCS$ is invoked, with its PDF denoted as f_{stime}; (2) $startRel$, a discrete variable whose value domain is $\{0, 1\}$, where $P(startRel = 1)$ represents the probability of $SubCS$ can be successfully invoked, with its PMF denoted as P_{srel}; (3) $startCost$, a discrete variable that represents the cumulated cost of CS before $SubCS$ is invoked, with its PMF denoted as P_{scost}. And output parameters of a $SubCS$ include: (1) $endTime$, a continuous variable that represents the time when the executing of $SubCS$ is finished, with its PDF denoted as f_{etime}; (2) $endRel$, a discrete variable whose value domain is $\{0, 1\}$, where $P(endRel = 1)$ represents the probability that $SubCS$ is just successfully executed, with its PMF denoted as P_{erel}; (3) $endCost$, a discrete variable that represents the cumulated cost of CS just after $SubCS$ is executed, with its PMF denoted as P_{ecost}.

For the first $SubCS$ that should be invoked in CS, the initial input parameter models are as follows: $f_{stime}(x) = \delta(x - t_0)$, $P_{srel}(1) = 1$, $P_{srel}(0) = 0$, $P_{scost}(0) = 1$. Below, we will introduce the method for calculating output parameters for different structured $SubCS$, including a single service, parallel composition, conditional composition and loop composition. Sequential composition will not be discussed here since it is just the simple repeating of single services.

A Single Service: $endTime$ is the sum of $startTime$ and ws's response time; The probability of ws being successfully executed is the product of the probability of it being successfully invoked and the reliability of ws; $endCost$ is the sum of $startCost$ and ws's cost.

$$f_{etime}(x) = f_{stime}(x) \circledast \sum_{i=0}^{N-1} (\sum_{m} \int_{t_i+mT}^{t_{i+1}+mT} f_{stime}(x)dx) \cdot f_{resp_i}(x) \tag{1}$$

$$\begin{aligned} P_{erel}(1) &= P_{srel}(1) \cdot \sum_{i=0}^{N-1} (\sum_{m} \int_{t_i+mT}^{t_{i+1}+mT} f_{stime}(x)dx) \cdot P_{rel_i}(1) \\ P_{erel}(0) &= 1 - P_{erel}(1) \end{aligned} \tag{2}$$

$$P_{ecost}(z) = \sum_{y_1+y_2=z} P_{scost}(y_1) \cdot \sum_{i=0}^{N-1} (\sum_{m} \int_{t_i+mT}^{t_{i+1}+mT} f_{stime}(x)dx) \cdot P_{cost_i}(y_2) \tag{3}$$

Parallel Composition: We first compute the output models for each branch. The models for the jth branch are calculated according to Formula (1), (2) and (3), and we denote them as f^j_{etime}, P^j_{erel} and P^j_{ecost}. The *endTime* of the composition is the maximum *endTime* of each branch; the composition is successfully ended if and only if all the branches are successfully ended; the *endCost* of the composition is the sum of each branch's *endCost*.

$$\begin{aligned} F^j_{etime}(x) &= \int f^j_{etime}(x)dx \\ f_{etime}(x) &= \sum_{j=1}^{k} f^j_{etime}(x) \cdot \prod_{l=1,\ldots,k \& l \neq j} F^l_{etime}(x) \end{aligned} \tag{4}$$

$$P_{erel}(1) = \prod_{j=1}^{k} P^j_{erel}(1) \qquad P_{erel}(0) = 1 - P_{erel}(1) \tag{5}$$

$$P_{ecost}(z) = \sum_{y_1+\ldots+y_k=z} \prod_{j=1}^{k} P^j_{ecost}(y_j) \tag{6}$$

Conditional Composition: We first compute the output models for each branch. The output parameters are the probability weighted sum of each branch.

$$f_{etime}(x) = \sum_{j=1}^{k} p_j f^j_{etime}(x) \tag{7}$$

$$P_{erel}(1) = \sum_{j=1}^{k} p_j P^j_{erel}(1) \qquad P_{erel}(0) = 1 - P_{erel}(1) \tag{8}$$

$$P_{ecost}(z) = \sum_{j=1}^{k} p_j P^j_{ecost}(z) \tag{9}$$

Loop Composition: We model the iteration number as a PMF. So a loop composition can be transformed into a conditional composition with a sequential composition in each path. The output models are then calculated according to the corresponding formulas.

By applying the above calculation methods, we get the probabilistic model for a composite service's *endTime*, *endRel* and *endCost* when it is invoked at a certain time t_0. The definitions of *endRel* and *endCost* remain the same with reliability and cost respectively. However, *endTime* is not the response time of the composite service. We get the response time model as $f_{resp}(x) = f_{etime}(x + t_0)$.

4.3 Establishing DQM for a Composite Service

With the problem of QoS estimation for single invocation solved, now we discuss how DQM is established for a composite service to reveal its QoS change patterns in a time cycle. If two invocation time points (to the same service) are very close, it is very likely that the QoS of these two invocations are very similar. Based on this observation, we first select a sequence of time points t_0, t_1, ..., t_l ($T_0 \leq t_0 < t_1 < \ldots < t_l \leq T_0 + T$, where T_0 is the cycle starting time and T is the cycle length). Then we compute the corresponding QoS models and get a model sequence M_{t_0},...,M_{t_l}. Next, we gradually merge adjacent models which are most similar. A merged model is computed as the average of two adjacent models. The merge procedure will continue until the similarities of current resultant models are all below a certain threshold. When the merge is finished, a new model sequence $M_{t_0 \ldots t_a}, M_{t_{a+1} \ldots t_b}, \ldots\ldots, M_{t_x \ldots t_l}$ is generated. Then, we segment the time cycle in the following way: if two adjacent time point t_y and t_{y+1} are assigned to two different models, then we set the middle of them as a split time point: $st = \frac{t_y + t_{y+1}}{2}$. And $M_{t_p \ldots t_q}$ is the corresponding segment model for time span $[\frac{t_{p-1}+t_p}{2}, \frac{t_q+t_{q+1}}{2})$.

In the merging process, we use Kullback-Leibler divergence to measure the difference of two probability distributions. Two distributions are similar if the KL divergence of them is small. There are three elements in our QoS model and we need to compute the KL divergence for every element and compute the sum of them as the final distance of two models.

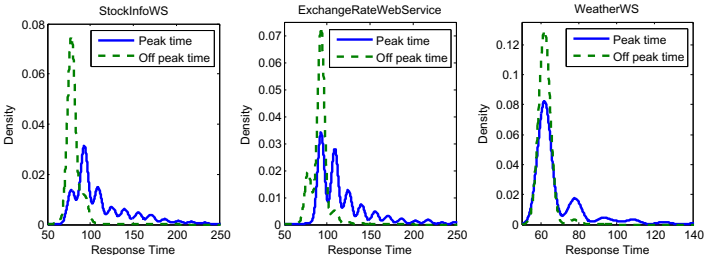


Fig. 1. PDFs of Real World Services

5 Experiment and Evaluation

To justify the effectiveness of the proposed dynamic QoS model and approach, we collect the response time from three real world web services: *StockInfoWS*[1], *ExchangeRateWebService*[2], and *WeatherWS*[3]. We first identify the QoS change

[1] http://www.webxml.com.cn/WebServices/StockInfoWS.asmx?wsdl
[2] http://webservice.webxml.com.cn/WebServices/ExchangeRateWebService.asmx?wsdl
[3] http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx?wsdl

pattern for these services based on the HMM method. The results show that all three services possess a circle length of 24 hours. *StockInfoWS* exhibits higher values for response time during work hours from approximately 9:30 to 15:00, medium values during lunch time and lower values for other time periods; The QoS pattern of *ExchangeRateWebService* is similar to *StockInfoWS* except that it does not present medium values during lunch time; For *WeatherWS*, the response time is high only within the peak hours between 9:15 and 10:45, and relatively low outside this time range. The PDFs of the services during peak hours and off peak hours are shown in Figure 1.

Next, we will show the accuracy of the QoS aggregation method proposed in Section IV. We create three web services $ws1$, $ws2$ and $ws3$, and their response time values are generated randomly from the collected samples of *StockInfoWS*, *ExchangeRateWebService* and *WeatherWS*, respectively. Then, these services are composed according to the composition structure shown in Figure 2. The probability of each conditional branch is set to be 0.5. And the probabilities that the loop structure will be executed 2 and 3 times are all set to be 0.5.

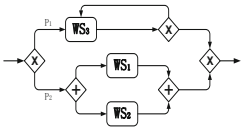


Fig. 2. A Composition

Monte Carlo simulation method is then used to simulate the QoS of the composite service. Based on the DQMs of the component services, we apply our estimation method to compute the PDF for the composite service. In addition, we also generate static QoS models for the component services when the time cycle related QoS changes are not taken into consideration, and compute the PDF of the composition accordingly. Figure 3 illustrates the experiment results when the invocation time is set that all component services are within peak hours. It shows that the PDF computed based on DQMs is very close to the simulation results, while the static models generate more deviations.

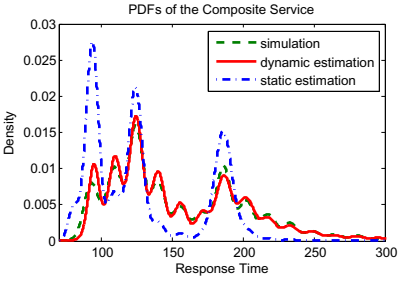


Fig. 3. PDFs of the Composition

6 Conclusion

In this paper we propose a dynamic QoS model to represent the time related characteristics of QoS. Various techniques are employed to develop the DQMs for component services as well as composite services. Experiments show that the proposed solution achieves high accuracy in QoS modeling and prediction.

Acknowledgment. This work is supported in part by NSFC grant 60873115.

References

1. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Journal of Web Semantics 1, 281–308 (2004)
2. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS aggregation for Web service composition using workflow patterns. In: EDOC 2004, pp. 149–159 (2004)
3. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate Quality of Service Computation for Composite Services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 213–227. Springer, Heidelberg (2010)
4. Hwang, S.-Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the QoS of web-services-based workfows. Information Sciences 177(23), 5484–5503 (2007)
5. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic qos and soft contracts for transaction-based web services orchestrations. IEEE Transactions on Services Computing 1(4), 187–200 (2008)
6. Zheng, H., Yang, J., Zhao, W.: Qos Probability Distribution Estimation for Web Services and Service Compositions. In: SOCA 2010 (2010)
7. Klein, A., Ishikawa, F., Bauer, B.: A Probabilistic Approach to Service Selection with Conditional Contracts and Usage Patterns. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 253–268. Springer, Heidelberg (2009)
8. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition, pp. 267–296 (1990)

CANPRO: A Conflict-Aware Protocol for Negotiation of Cloud Resources and Services

Marco A.S. Netto

IBM Research
Sao Paulo, Brazil

Abstract. In a Cloud environment, users face the challenge of selecting and composing resources and services from a single or multiple providers. As several negotiations can occur concurrently, information on service and resource availability may be out-of-date, thus requiring several iterations between users and providers until an agreement is achieved. To address this problem, we introduce CANPRO, a Conflict-Aware Negotiation Protocol for allocating Cloud resource and services aimed at reducing cancellation messages during negotiation. CANPRO allows users (or entities on their behalf) to know the amount of resources being concurrently negotiated by other users and the number of users interested in such an amount, while still keeping users' information private. By knowing this information, users can, for instance, confirm allocation requests with lower chances of having collisions with other users. In addition, for the same reason, users can increase their time deciding which (combination of) resources they want to allocate. The paper presents comparative results of CANPRO against the popular two-phase commit protocol (2PC) and a state-of-the-art protocol named SNAP-3PC. We used think time, network overhead, number of concurrent negotiations and providers as main metrics. The results are promising and the protocol can be used in scenarios other than Cloud Computing; for instance, bookings of health services, cars, tickets for venues, schedule of appointments, among others.

1 Introduction

Users have access to several services in the Internet to perform tasks that range from exchanging e-mails to allocating high performance computing resources. Some of these services and resources are offered by Cloud providers using a pay-as-you-go model. As the number of these providers and services increases, users face the challenge of selecting, and possibly, composing them in a complex and dynamic computing environment.

Several users may request resources and services at the same (or within the same time interval). During negotiations, information about resource availability, from the moment of selecting to the moment of confirming the allocations, may be out-of-date. This has a particular impact when users are composing services from multiple providers, as a failure in a single allocation may result in a renegotiation with all the resource providers. The failure generates a phenomenon

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 541–548, 2011.

called *livelock*, in which multiple users keep trying to allocate resources over and over gain without success, thus requiring a considerable number of negotiation messages to satisfy all users. This problem is well-known and investigated in the area of Distributed Transactions [2], mainly investigated in the data base community and in the last decade in the Grid community [10]. For Cloud Computing, such renegotiation may cause violation of Service Level Agreements (SLAs) for confirmed requests.

A number of protocols for allocating distributed resources have been proposed in the literature [5–8, 10–12], being most of them aimed at avoiding deadlocks and livelocks, and reducing the number of messages during negotiations. These protocols consider that a user is not aware that other users are concurrently negotiating for the same resources—the user only receives a message that was not possible to commit the selected resources. Therefore, users have no chance of optimizing their resource selection in order to avoid such a competition. As a consequence, negotiations require several messages and users have little time to make their allocation decisions.

The main contribution of this paper is the Conflict-Aware Negotiation Protocol (CANPRO), which aims at reducing the number of cancellation messages during negotiations and increasing the time users can spend to decide which resources they want to allocate. This is achieved by allowing users (or entities on their behalf) to know the amount of resources being concurrently negotiated by other users and the number of users interested in such an amount. In other words, all users are aware about intentions of other users concurrently negotiating for conflicting resources. By knowing this information, users can, for instance, confirm allocation requests with lower chances of having collisions with other users. In addition, for the same reason, users can increase their time deciding which (combination of) resources they want to allocate. We compare CANPRO against the popular two-phase commit (2PC) protocol and a state-of-the-art protocol named SNAP-3PC, which is a three-phase commit extension of the Service Negotiation and Acquisition Protocol. The results are promising and the protocol can be used in scenarios other than Cloud Computing.

2 Conflict-Aware Negotiation Protocol

The Conflict-Aware Negotiation Protocol (CANPRO) allows users to be aware about concurrent negotiations that conflict with one another. Whenever a provider receives a message from a user requesting for resources, this provider sends back to the user a message describing: (1) whether the requested capacity is available; (2) the percentage of that requested capacity that conflicts with the capacity being concurrently negotiated by other users; and (3) the number of those users negotiating the conflicting capacity. The users that already received offers from the provider(s) obtain an update on conflicts, which has an influence on their decision regarding where and how many resources they should commit.

Figure 1 illustrates an example of information flow between one resource provider and two users. In this example, User_2 requests for resources just after

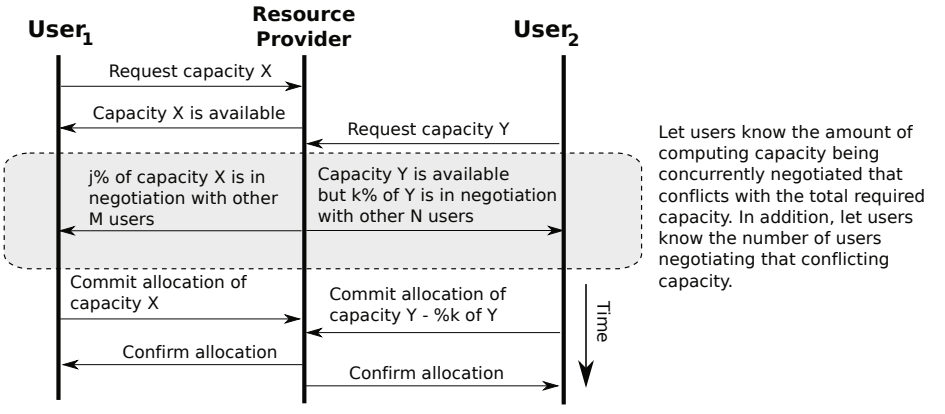


Fig. 1. CANPRO execution example

resource provider sends $User_1$ information about resource availability. Knowing that $User_1$ is negotiating resources, the provider sends $User_2$ information about resource availability considering the possible conflicts with $User_1$, who is also notified about a possible conflict. For this particular example, $User_1$ commits the original request, whereas $User_2$ commits only part of it in order to avoid collision. Both users receive confirmations on their commit requests.

CANPRO requires a data structure to keep track of the concurrent negotiations. Our current implementation utilizes a linked list of resource offers called *NegotiationQueue*. These resource offers [9] contain the resource availability given to a user by the provider. For CANPRO, these offers are extended to add conflict information: percentage of conflicting resources and the number of users negotiating those resources. The NegotiationQueue is updated when (i) a new request gets to the resource provider; (ii) the provider confirms the resource allocation (commit message); (iii) the provider rejects user request; (iv) the user decides not to continue the negotiation or when the user has to start a new negotiation.

Based on the NegotiationQueue, the provider can generate conflict information and notify users whenever this information changes. The conflict represents the percentage of resource being concurrent negotiated by multiple users. Before the resource provider receives the request from $User_2$, the NegotiationQueue contains only the offer given to $User_1$.

The NegotiationQueue is updated with the offer given to $User_2$ once the new request arrives. At this point, the provider triggers the algorithm to calculate conflicts and notify users if any original offer has been changed. Algorithm 1 presents a pseudo-code of how conflicts can be calculated. The algorithm receives the NegotiationQueue and returns a list of offers containing conflict information.

The variables used in the algorithm are:

- **bin:** data structure (e.g. array) that stores request capacity information;
- **binList:** list of bins;

Algorithm 1. Pseudo-code for generating list of offers with conflicting information based on the NegotiationQueue

```
1  bin ← create a bin with the size of the available capacity
2  for ∀request ∈ NegotiationQueue do
3      Fill bin with requested capacity
4      if bin full = true then
5          binList.add(bin)
6          Create another bin with same capacity
7          Fill it with the remaining requested capacity
8  for ∀request ∈ NegotiationQueue do
9      conflictPortion ← numberOfConflicts ← 0
10     for ∀bin ∈ binList do
11         capacityRange ← find capacity range in bin that contains request
12         if capacityRange found = true then
13             conflictPortion ← conflictPortion + conflict part with other requests
               in this capacityRange
14             Increment numberOfConflicts with requests on this capacityRange
15     offer ← request information plus conflictPortion and numberOfConflicts
16     offerList.add(offer)
17 return offerList
```

- **capacityRange:** if bin is an array, capacityRange is a range index;
- **offer:** resource availability information to be sent to users;
- **offerList:** list of offers.

From the user side, once they receive offers (new or updated ones), they can select providers that sent offers with fewer number of conflicts and lower percentage of conflicting capacity.

3 Evaluation

The basis for the design of CANPRO is predicated on the idea that by users knowing that other users are negotiating the same (or a portion of their) resources, they can select a group of resources with lower chances of failures when commiting the allocation. The experimental results in this section demonstrate that the principle is sound.

We evaluated CANPRO against the popular two-phase commit protocol (2PC) and SNAP-3PC protocol. The latter protocol allows users who are negotiating for resources to be notified when the status of a resource is changed. We developed a multi-thread event-driven simulator with the implementation of the two protocols and CANPRO. The simulator receives a file containing information on user requests, such as user think time, allocation strategy (single x multiple providers), network delay to communicate with the providers, and the provider's processing time for the request.

We created sets of workloads that vary *network delay* to send messages, *think time*, and *processing time* that follow a Gaussian distribution. The values vary 1000±200, 2000±1000, and 1000±200 respectively (time in milliseconds). We also varied the number of concurrent requests and resource providers. For each workload, half of users requests a specific provider and the other half requests a group of providers. The total number of requests processed for the three protocols in the experiments varying all parameters is 14400. As metrics, we measured the number of trials and number of cancellation messages until a request is successfully committed.

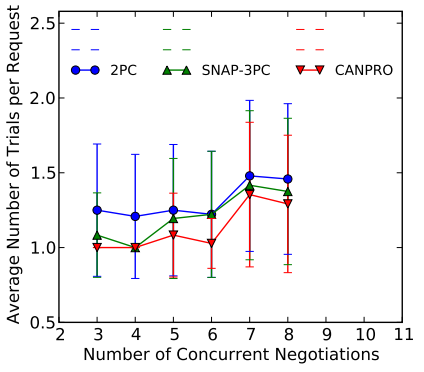
Figures 2 (a), (c), and (e) present the number of trial allocations for three, five, and seven providers, respectively, as a function of the number of concurrent negotiations (starting with the same number of providers, and a fixed think time). The lower the number of trials the better the protocol is. The average number of trials per request increases with both the number of concurrent negotiations and the number of providers. This happens because the chances of users receiving out-of-date information on resource availability increase with these two variables. CANPRO outperforms 2PC and SNAP-3PC with the same proportion for the three numbers of providers. This indicates that the improvement scalability of CANPRO is similar to 2PC and SNAP-3PC.

Figures 2 (b), (d), and (f) present the total number of cancellation messages for three, five, and seven providers, respectively, as a function of the number of concurrent negotiations. The behavior of this metric is similar to the previous one, however when the number of providers is three, it is observed that the higher the number of concurrent negotiations the higher the advantage of CANPRO in relation to 2PC. This happens because the number of negotiations is lower enough so that the percentage of conflicting capacity has more influence than the number of conflicting negotiations. This scenario is therefore the one where CANPRO has higher benefits.

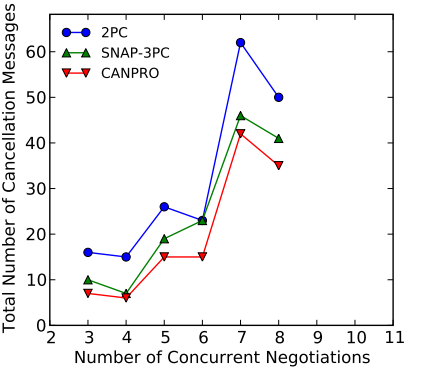
In order to observe the effect of the think time for both negotiation protocols, we fixed the number of providers in five, and the number of concurrent negotiations as ten. As observed in Figure 3, the higher the think time the better the performance of CANPRO in relation to 2PC and SNAP-3PC. This is due to the fact that users can receive messages about negotiation conflicts while they are deciding on resource selection. For 2PC and SNAP-3PC, the higher the think time values the higher the probability of users receiving out-of-date information. As it is showed in Figure 3, 2PC and SNAP-3PC do not produce a steady performance, i.e. there is high variability when changing the average think time, whereas for CANPRO, the number of cancellations is reduced with the increase in the think time. This is a particularly promising result, as we expect users to have access to more services in the Internet, so they require time to think about their decisions.

4 Related Work

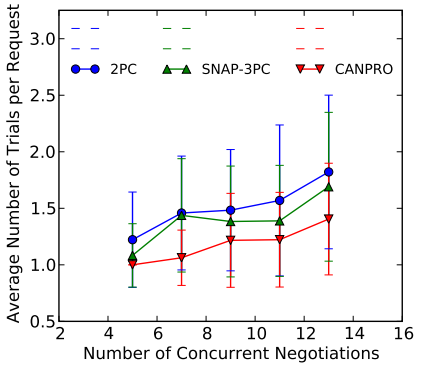
Existing protocols on resource negotiation fall into following categories: Two-Phase Commit (2PC), Three-Phase Commit (3PC), Order-based Deadline, and



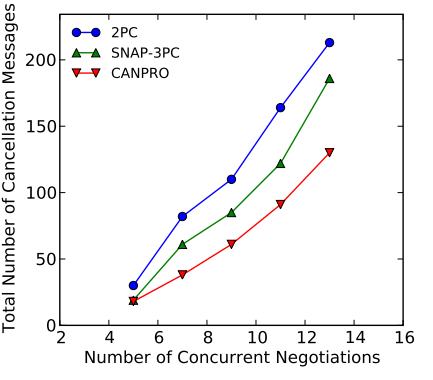
(a) Three Providers.



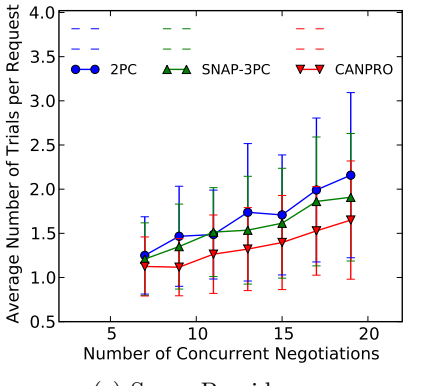
(b) Three Providers.



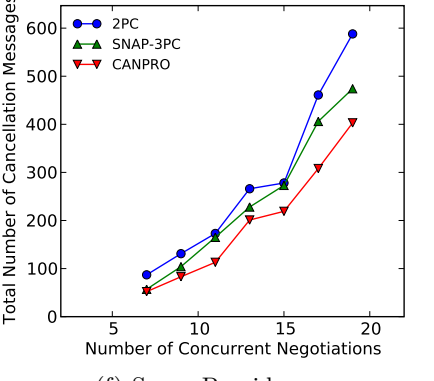
(c) Five Providers.



(d) Five Providers.



(e) Seven Providers.



(f) Seven Providers.

Fig. 2. Number of trial allocations and cancellation messages as a function of the number of providers and concurrent negotiations

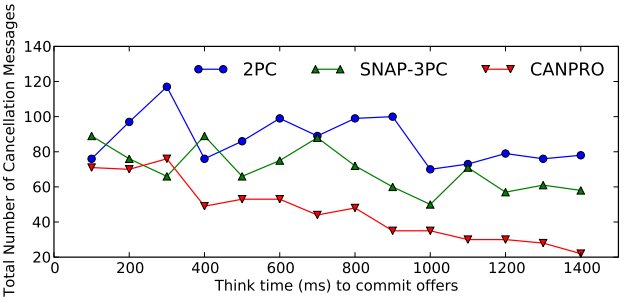


Fig. 3. Number of cancellation messages for five RPs as a function of think time

Polling-based protocols. Most of these projects have as motivation the problem of resource co-allocation for Grid Computing environments [10].

Kuo and Mckeown [7] presented a protocol for advance reservations and co-allocation, which extends the 2PC protocol with support for cancellations that may occur at any time. Park [11] introduced a decentralized protocol for allocating large-scale distributed resources, which is free from deadlocks and livelocks. The protocol is based on the Order-based Deadlock Prevention Protocol ODP^2, but with parallel requests in order to increase its efficiency. Another approach to avoid deadlock and livelock is the exponential back-off mechanism [6].

Takefusa et al. [12] developed a 2PC-based protocol that uses polling from the client to the server. Maclaren et al. [8] introduced a system called HARC (Highly-Available Robust Co-allocator), which uses 3PC protocol based on Paxos consensus algorithm [4] and focuses on fault tolerance aspects. Azougagh et al. [1] introduced the Availability Check Technique (ACT) to reduce the conflicts during the process of resource co-allocation. Requests wait for updates from providers until they fulfill their requirements. The main difference of ACT and CANPRO is that in the former, users are not aware about possible conflicts during negotiation, therefore it cannot optimize their resource selection decisions.

Czajkowski et al. [3] proposed the Service Negotiation and Acquisition Protocol (SNAP), which aims at managing access to and use of distributed computing resources by means of Service Level Agreements (SLAs). The protocol is not optimized to work with out-of-date information on resource availability. In order to solve this problem, Haji et al. [5] developed a 3PC protocol for SNAP-based brokers. Its key feature is the use of *probes*, which are signals sent from the providers to the candidates interested in the same resources to be aware of resource status' changes. Different from Haji et al.'s protocol, CANPRO notifies users on other concurrent negotiations, which is before the status' changes, so users can select resources with lower chances of being taken by other users.

5 Concluding Remarks

This paper presented CANPRO, a conflict-aware protocol for negotiation of Cloud resources and services. With CANPRO, users can have higher thinking time to commit requests and network communication can have higher latency.

This is achieved by allowing users to be aware that there are other concurrent users negotiating for the same resources. With this conflict information in hands, users can select resources/providers with lower probability of having requests rejected. Based on experimental results, CANPRO is able to reduce cancellation messages when there are concurrent negotiations compared to 2PC and SNAP-3PC protocols. We observed that it is quite frequent to have situations where users base their resource selection decisions on out-of-date information, and a conflict-aware protocol, in this case, CANPRO, is an important tool to handle this problem. CANPRO could also be used for other scenarios such as booking of air planes, cars, health services, and scheduling of people and rooms for meetings.

References

1. Azougagh, D., Yu, J.L., Kim, J.S., Maeng, S.R.: Resource co-allocation: A complementary technique that enhances performance in grid computing environment. In: Proceedings of ICPADS (2005)
2. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Computing Surveys 13(2), 185–221 (1981)
3. 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: Proceedings of JSSPP (2002)
4. Gray, J., Lamport, L.: Consensus on transaction commit. ACM Transactions on Database Systems 31(1), 133–160 (2006)
5. Haji, M.H., Gourlay, I., Djemame, K., Dew, P.M.: A SNAP-based community resource broker using a three-phase commit protocol: A performance study. The Computer Journal 48(3), 333–346 (2005)
6. Jardine, J., Snell, Q., Clement, M.J.: Livelock avoidance for meta-schedulers. In: Proceedings of HPDC (2001)
7. Kuo, D., Mckeown, M.: Advance reservation and co-allocation protocol for grid computing. In: Proceedings of e-Science 2005 (2005)
8. Maclaren, J., Keown, M.M., Pickles, S.: Co-allocation, fault tolerance and grid computing. In: Proceedings of the UK e-Science All Hands Meeting (2006)
9. Netto, M.A.S., Buyya, R.: Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In: Proceedings of HCW/IPDPS (2009)
10. Netto, M.A.S., Buyya, R.: Resource co-allocation in grid computing environments. In: Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, IGI Global (2009)
11. Park, J.: A deadlock and livelock free protocol for decentralized internet resource coallocation. IEEE Transactions on Systems, Man, and Cybernetics Part A 34(1), 123–131 (2004)
12. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., Sekiguchi, S.: GridARS: an advance reservation-based grid co-allocation framework for distributed computing and network resources. In: Proceedings of JSSPP (2007)

Game-Theoretic Analysis of a Web Services Collaborative Mechanism

Babak Khosravifar[1], Jamal Bentahar[1], Kathleen Clacens[2], Christophe Goffart[2], and Philippe Thiran[2]

[1] Concordia University, Montreal, Canada
[2] University of Namur, Namur, Belgium
b_khosr@encs.concordia.ca, bentahar@ciise.concordia.ca, {k.clacens,darkyunsung}@gmail.com, pthiran@fundp.ac.be

Abstract. Web services are business applications having the capability to cooperate within groups to increase the efficiency of serving customers. There have been a number of proposed frameworks aggregating web services for the purpose of enhancing their capabilities with respect to providing the required service. However, the grouping procedure has got less attention. In this paper, we discuss the mechanism web services can use to join existing groups of web services (known as communities). Moreover, we analyze the scenarios where the community is filled up with web services that lied about their capabilities before joining. The objective is to provide and maintain a truthful environment where involving components act truthfully.

Keywords: Web services, Reputation, Agents.

1 Introduction

During recent years, web services have obtained a strong attention as they represent distributed cooperation in business and IT networks. Web services hold predefined capabilities that let them realize their goals by engaging in interactions with one another. They are loosely-coupled business applications and willing to cooperate in distributed settings for the sake of efficiency. To this end, there have been efforts made in collaboration between web services [4]. The objective of this collaboration is to increase web services' capabilities. However, web services also need to be located in environments where requests are received from users in continuous manner, which requires a high reputation. Furthermore, there are many aspects that should be considered when analyzing web services working within groups (generally called *communities*). In this paper, we mainly concentrate on the joining aspect of web services to existing communities by focusing on their rational behaviors. The concept of community of web services together with its relative details are explained in Section 2.4.

In this paper, we present a game-theoretical model analyzing the communities of web services from the perspective of hosting different web services. A game is defined between the *master web service* acting as the representative of the community and agents acting as information providers within a group called *information service group*.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 549–556, 2011.

Agents in this group, called *information service agents*, provide the necessary information regarding the web service that is attempting to join a community. The involved information service agents can either lie or tell the truth about the requested information. A payment mechanism that provides diverse ranges of payoffs to the information service agents according to their chosen strategies is discussed and analyzed. Overall contribution of the paper is summarized in game-theoretic analysis investigating the stabilized situation where information service agents provide their actual information and act truthfully.

2 Related Preliminaries

2.1 Web Services

The main motivation behind the use of web services technology is the development of loosely-coupled, cross-enterprize business processes. This means web services can be used without worrying about how they are implemented or where in the world they are hosted. As in [4,5] and [8], we abstract web services as rational intelligent agents, which are benefit maximizers. In our framework, the goal of these agents is to receive user requests and satisfy them the best they can.

2.2 Information Service Agents

Information service agents provide a ranking of web services. They know all the active web services and depending on their personal behavior and incentives, they could provide accurate or non-accurate information about these web services.

2.3 Reputation

As for non-internet based services, for example buying a plane ticket from a travel agency, it is possible that several web services have the same functionalities. We therefore need to differentiate those different, but functionally similar web services. We will use the concept of reputation that strongly depends on the quality of service (QoS). The Quality represents the capability of a web service to handle the users' requests in a timely fashion. In order to use this concept of reputation, we use a system architecture having the following elements: (1) some registries containing entries that describe individual web services; and (2) a reputation mechanism [8]. The registries can be implemented using the UDDI protocol, which defines a standard method for publishing and discovering web services [3].

2.4 Community of Web Services

A web service alone can easily be overloaded by an intense flow of user's requests. It will lead to a poor efficiency and therefore a drop in the users' satisfaction and reputation [6]. As argued in [4] and [6], a solution is to group different web services

in a community. By aggregating the total number of requests that each single web service can handle, and redistributing them among all the members, a community will be granted a better availability and hence better efficiency.

In a community, we distinguish between two types of web services: a unique community master and two or more community slaves. The master takes the responsibility of leading the community. He must attract new web services in the community by using rewards, convince web services to stay in the community, and select the web service that will respond to a user's request. When a user sends a request to a community, the master has to nominate a slave from his community that can handle that request.

3 The Model

3.1 The Modelled Structure

In general, when a customer asks an information service agent, there are two possible strategies: lying or telling the truth. The utility $u_k(x)$ of an information service agent k choosing the strategy x is a function having 3 incentive components where the agent obtains rewards or penalties according to the chosen strategy. It is defined as follows:

$$u_k(x) = f_k(x) + g_k(x) + c.h_k(x) \quad where$$

$$f_k(x) > 0, f_k(x) < |g_k(x)| \text{ and } f_k(x) + |g_k(x)| < |h_k(x)| \tag{1}$$

The first component $f_k(x)$ is a positive reward that a customer gives to the information service agent k who is willing to provide the asked information. The second component incentive $g_k(x)$ corresponds to a value that will be granted to the information service agent depending on the similarity between the information she gives and the average information revealed by the other information service agents. This second value can be negative if the distance is high and therefore acts as a punishment preventing the information service agent to reveal incorrect information. Finally, after having used the service being evaluated, the customer can tell whether the provided QoS fits the information service agents' predictions. The difference between what was expected and what was actually experienced is used to calculate the third component incentive $h_k(x)$. Of course, the latter can only be considered if the customer decides to have a transaction with the provider. In this case, c will be set to 1 in the utility function, otherwise, c will be equal to 0.

It is important that each incentive has to be higher than the summation of the previous ones, that is to say $f_k(x) < |g_k(x)|$ and $f_k(x) + |g_k(x)| < |h_k(x)|$. It guarantees the incentive compatibility property which means that information service agents, as a utility maximizers, will reveal exactly what they believe about the service being evaluated or, in other words, they will tell the truth.

Lemma 1. *The utility function $u_k(x)$ satisfies the following properties:*

1. *Revealing the true trust value about the service being evaluated is a Nash equilibrium strategy.*

2. *If the service being evaluated is untrustworthy, revealing a false trust value is not a Nash equilibrium strategy.*
3. *If the provider is trustworthy, revealing a false trust value is a Nash equilibrium strategy.*

Proof. As the first component is guaranteed, an information service agent has an incentive to lie only if the second incentive $g_k(x)$ is strictly positive and the third incentive is not obtained ($c = 0$). Therefore, the gained utility will be $|f_k(x) + g_k(x)|$. However, by truth telling, the information service agent can get the third incentive as well, so the gained utility will be $|f_k(x)+g_k(x)+h_k(x)|$. Consequently, telling the truth is the best strategy under Nash equilibrium. Adversely, if the service is untrustworthy, the agent will gain less by lying as $c.|h_k(x)|$ will be strictly negative. Thus, lying is not a Nash equilibrium. However, if the service is trustworthy but all the information service agents decide to lie, no one can gain a better payoff by deviating from the group's telling as $g_k(x)$ will be negative and $c.|h_k(x)|$ null, which proves the third property.

The problem we would like to investigate is the situation where a web service, aiming to join a community, pays the information service agent each time he gets chosen to join a community after being referenced by this agent. The web service could be tempted to cheat and pay more the information service agents so that they improve his reputation when informing a community master. If the reward a web service gives to information service agents to fake their opinion is big enough, these agents can be tempted to lie. In that case, a community will hire the web service and maybe expect more than what the service can really handle.

3.2 The Modelled Game

In the set up game, there is a number of involving agents: (1) a typical community (M_i); (2) a typical single web service (S_j); and (3) a typical information service agent (I_k). In the set up game, both the community of web services and single web service handle user requests according to their capacities. In the ideal case, M_i and S_j desire to be neither overloaded, nor idle. When M_i asks I_k for information about the quality of S_j, a reputation value is produced. This value, representing what I_k reports to be the reputation of S_j, is assigned to the triplet (M_i, S_j, I_k). Such a value is saved by M_i in registries to keep a record of who reported what about the different services so the value of the actual quality can be compared with the provided one. We define strategies of truth telling and lying within strategy profile $st = \{0, 1\}$, where 0 and 1 respectively reflect lying and telling the truth. As rational agent, an information service agent would choose his strategy based on the gained utility. For instance, he might obtain an acceptable offer that encourages him to lie and provide inaccurate information. Meanwhile, there might be other effective parameters that encourage agents to provide truthful information. This paper mainly focuses on the truth telling issues and discusses the way of converging towards this situation.

3.3 Payments

The payments information service agents receive can come from M_i and from the service being evaluated. For simplicity reasons and to avoid notation confusion, we use

three simple variables α, β, and γ that refer to the previously described incentives $f_k(x)$, $g_k(x)$, and $h_k(x)$ in Equation 1. M_i pays α to I_k as an incentive to provide information about S_j. Parameter β is the payment M_i gives to I_k after collecting reports about S_j from $I_0, \ldots, I_n$. β is calculated in Equation 2 where $Tr_{M_i}^{I_x}$ represents the value of trust (confidence) M_i has towards I_x, and $Rr_{I_x}^{S_j}$ represents the value of the reputation of S_j reported by I_x. β is then a decreasing logarithmic function ($0 < b < 1$) on the difference between the average reported value by all the information service agents and the value reported by I_k. Consequently, I_k receives the highest payment for the closest reported value to this average (we assume that this distance is always different from 0, otherwise a fixed payment, close to the highest, can be given).

$$\beta = log_b(|\frac{\sum_{x=0}^{n} Tr_{M_i}^{I_x} Rr_{I_x}^{S_j}}{\sum_{x=0}^{n} Tr_{M_i}^{I_x}} - Rr_{I_k}^{S_j}|) \tag{2}$$

γ is the payment M_i gives to I_k if M_i has registered S_j in C_i and evaluated his reputation. After this evaluation, M_i can compare $Rr_{I_x}^{S_j}$ to $Ro_{M_i}^{S_j}$ and pay I_k with a value of γ, which is computed in Equation 3, where $Rr_{I_x}^{S_j}$ represents the value of the reputation of S_j reported by I_x, and $Ro_{M_i}^{S_j}$ represents the value of the actual reputation of S_j observed by M_i. As in Equation 2, I_k receives the highest payment when the reported reputation value is the closest to the observed one. If the distance between these two values is 0, a fixed payment can be set.

$$\gamma = log_b(|Rr_{I_x}^{S_j} - Ro_{M_i}^{S_j}|) \tag{3}$$

The single web service S_j can pay π to I_k in order to encourage him to increase his reputation when reporting to communities. This payment will only be received if M_i chooses to add S_j to the community. As shown in Equation 4, this payment is an exponentially decreasing function on the actual web service's reputation Rr^{S_j}, which means if S_j has a high reputation, only a small payment can be given to I_k, but if this reputation is low, S_j has to reward high I_k if he is getting selected by M_i. In this equation, λ is application-dependent that measures the decreasing slope. In this paper, we assume $\lambda = 1$.

$$\pi = e^{-\lambda Rr^{S_j}} \tag{4}$$

4 Cases Overview

In this section, we analyze different cases using a game involving two players ($I.S$ for a typical information service agent and $O.I.S$ for the other information service agents). Each game is represented as a table where the rows show the strategies of $I.S$ and the columns indicate the strategies of $O.I.S$. Each cell of the table represents the action profile, i.e. the outcome that each player has according to the adopted strategy. The first outcome is for $I.S$ and the second one for $O.I.S$. If the received payment is negative, we use the superscript $-$. For example, β^- means a negative β, otherwise, β is positive.

We focus on the cases where S_j is honest (with good and bad QoS). Due to space limitations, we skip the case where S_j is dishonest, however, this case could be easily

generalized. In our considered cases, S_j does not try to corrupt $I.S$ and $O.I.S$ by rewarding them. Therefore, we will only take into account the first three payments α, β and γ. Then, we apply the same analysis on the dishonest S_j, where the information service agents can receive the incentive π in order to improve fraudulently the information they report about S_j.

S_j has good QoS. The assigned payoff regarding each strategy is set up in Table 1. If every information service agent tells the truth, i.e. informing M_i that S_j is good, everyone will receive a maximum payment of $\alpha + \beta + \gamma$. Indeed, the information services will receive α in reward for processing the request. They will gain β because the value of the report that each information service agent will give will be close to the average of all the reported values. Finally, they will receive the third payment γ since the observed reputation and announced one will be close to each other.

Table 1. Honest single web services - S_j has good QoS

		O.I.S	
		Truth	Lie
I.S	Truth	$(\alpha+\beta+\gamma)$, $(\alpha+\beta+\gamma)$	$(\alpha+\beta^-)$, $(\alpha+\beta_-)$
	Lie	$(\alpha+\beta^-+\gamma^-)$, $(\alpha+\beta_-+\gamma)$	$(\alpha+\beta)$, $(\alpha+\beta)$

If $I.S$ decides to lie while $O.I.S$ continue to tell the truth, $I.S$ will degrade his total payment to $\alpha + \beta^- + \gamma^-$. He will still receive α but a negative β and γ as computed in Equations 2 and 3 because the reputation he reported will be far from the average and observed reputation. On the other hand, $O.I.S$ will also perceive a degradation in their total payment. In fact, because one information service agent decided to report a reputation totally different from those reported by the others, the average will be smaller than if every information service agent had reported close values. In this case we use β_- instead of β. If $O.I.S$ decide to change their strategies and lie, they will only get α and β_- as payment. Majority announcement that S_j is bad implies that the community will not accept him, and therefore the third payment γ will not be granted. As $I.S$ did not change his strategy, he will get α and a negative β because of the big distance from the average and not γ because S_j will not enter the community. If everyone decides to lie, all the information services will get α and β and nobody will receive γ as the service will not join the community. Thus, there is an incentive to tell the truth for everyone because it corresponds to the situation that guarantees the maximum payment $\alpha + \beta + \gamma$. This game has the following properties:

Lemma 2. *Telling the truth by all the information service agents is the only Pareto optimal and Nash equilibrium (Pareto-Optimal-Nash).*

Proof. The proof is straightforward from Table 1 since any change of strategy of telling the truth would degrade other agents' payoffs and no other situation has this property.

Lemma 3. *Lying by all the information service agents is Pareto optimal.*

Proof. The proof is straightforward from Table 1 since any change of one's strategy of lying would degrade the opponent's payoff.

S_j has bad QoS. In this second case, we assume that the honest S_j provides a bad QoS.

Table 2. Honest single web services - S_j has bad QoS

		O.I.S.	
		Truth	Lie
I.S.	Truth	$(\alpha+\beta)$, $(\alpha+\beta)$	$(\alpha+\beta^-+\gamma)$, $(\alpha+\beta_-+\gamma^-)$
	Lie	$(\alpha+\beta^-)$, $(\alpha+\beta_-)$	$(\alpha+\beta+\gamma^-)$, $(\alpha+\beta+\gamma^-)$

If every player tends to tell the truth and reveals that S_j has bad QoS, they will all get a payment of $\alpha + \beta$ (see Table 2). If $I.S$ decides to modify his strategy and lie by announcing that S_j has good QoS, he will degrade his total payment. He will still receive α but a negative β because the reputation he announced is far from the average. However, $O.I.S$ will get the two first payments but the second one will be slightly decreased in comparison to the previous situation. Therefore, they will perceive $\alpha+\beta_-$. In the next situation, assume that $O.I.S$ change their strategies and start to lie. Because the majority of information service agents will declare S_j as good, the web service will join the community. Soon, M_i will realize that S_j has actually bad QoS. Therefore, the group of $O.I.S$ will receive $\alpha+\beta_-+\gamma^-$. On the other hand, $I.S$ who kept his strategy of telling the truth will get $\alpha+\beta^-+\gamma$. Indeed, the information service will receive a negative β but will be rewarded by γ as he reported correctly that S_j has bad QoS. If everyone decides to lie, all the information service agents will get $\alpha+\beta+\gamma^-$. γ will be negative because S_j will join the community, so M_i will discover his bad QoS. The following property results directly from Table 2.

Lemma 4. *Telling the truth is the only Pareto-Optimal-Nash.*

5 Related Work

In many frameworks proposed in the literature, service selection and task allocation are regulated based on the reputation parameter [10,11]. In [1], the proposed framework regulates the service selection based on the trust policies expressed by the service users. In [9], authors propose ontology for quality of service. Users compute the web services' reputation using ratings. The frameworks proposed in [3,8] address effective reputation mechanism for web services. All these models address the reputation in environments where web services function alone. In such models, web service efficiency is not discussed in details and in general, balancing the market share with the capacity is not considered as an issue for web service besides his reputation.

There have been few work addressing the communities of services. The objective is to facilitate and improve the process of service selection and effectively regulate the process of request and task allocation [2]. In [4], authors propose a reputation-based architecture for communities and investigate the collusion scenarios that might falsely increase communities' reputation in the network. In [5], the authors mainly address the overall assessed reputation that is used as a main reason for service selection. The authors do not consider truth/lie telling analysis as a factor impacting service selection.

6 Conclusion

The contribution of this paper is the proposition of a game-theoretic based model to analyze the best efficiency characteristics for the active services in open networks. The proposed framework considers the chances of web services in joining a community in different cases with truthful and lying information service agents. The proposed game analyzes the existing Nash equilibrium and situations where the maximum payoff is obtained. Our model has the advantage of being simple and taking into account three important factors: (1) rational services seek better status in the environment by joining the community; (2) rational information service agents obtain higher payoff by truth telling; and (3) the community is obtaining more effective web services while the information service agents challenge for providing truthful information. As future work, we plan to consider the user role in the game to obtain more accurate results when users act rationally. Moreover, we would like to achieve a collusion resistant efficiency mechanism, which is still an open problem in open environments.

References

1. Ali, A.S., Ludwig, S.A., Rana, O.F.: A cognitive trust-based approach for Web service discovery and selection. In: Proc. of the Euro. Conference on Web Services, pp. 38–40 (2005)
2. Jacyno, M., Bullock, S., Luck, M., Payne, T.R.: Emergent Service Provisioning and Demand Estimation through Self-Organizing Agent Communities. In: 8th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 481–488 (2009)
3. Kalepu, S., Krishnaswamy, S., Loke, S.W.: A QoS metric for selecting Web services and providers. In: Proc. of the 4th International Conference on Web Information Systems Engineering Workshops, pp. 131–139 (2003)
4. Khosravifar, B., Bentahar, J., Moazin, A., Thiran, P.: Analyzing communities of web services using incentives. International Journal of Web Services Research 7(3), 30–51 (2010)
5. Khosravifar, B., Bentahar, J., Thiran, P., Moazin, A., Guiot, A.: An approach to incentive-based reputation for communities of Web services. In: Proc. of IEEE 7th International Conference on Web Services, pp. 303–310 (2009)
6. Khosravifar, B., Bentahar, J., Moazin, A., Maamar, Z., Thiran, P.: Analyzing communities vs. single agent-based web services: trust perspectives. In: Proc. of the IEEE international Conference on Services Computing, pp. 194–201 (2010)
7. Maamar, Z., Subramanian, S., Thiran, P., Benslimane, D., Bentahar, J.: An approach to engineer communities of web services: Concepts, architecture, operation, and deployment. International Journal of E-Business Research 5(4), 1–21 (2009)
8. Maximilien, E.M., Singh, M.P.: Conceptual model of Web service reputation. SIGMOD Record 31(4), 36–41 (2002)
9. Maximilien, E.M.: Multiagent system for dynamic web services selection. In: The 1st Workshop on Service-Oriented Computing and Agent-based Eng., pp. 25–29 (2005)
10. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic QoS and soft contracts for transaction based web services. In: IEEE International Conference on Web Services, pp. 126–133 (2007)
11. Ruth, M., Shengru, T.: Concurrency issues in automating RTS for Web services. In: IEEE International Conference on Web Services, ICWS 2007, pp. 1142–1143 (2007)

Importance Sampling of Probabilistic Contracts in Web Services

Ajay Kattepur

IRISA/INRIA, Campus Universitaire de Beaulieu, Rennes, France

Abstract. With web services quality of service (QoS) modeled as random variables, the accuracy of sampled values for precise service level agreements (SLAs) come into question. Samples with lower spread are more accurate for calculating contractual obligations, which is typically not the case for web services QoS. Moreover, the extreme values in case of heavy-tailed distributions (eg. 99.99 percentile) are seldom observed through limited sampling schemes. To improve the accuracy of contracts, we propose the use of variance reduction techniques such as importance sampling. We demonstrate this for contracts involving *demand* and *refuel* operations within the *Dell* supply chain example. Using measured values, efficient forecasting of future deviation of contracts may also be performed. A consequence of this is a more precise definition of sampling, measurement and variance tolerance in SLA declarations.

Keywords: Web Services, QoS, Importance Sampling, SLA.

1 Introduction

Web services continue to attract applications in many areas [1]. With increasing efforts to standardize performance of web services, focus has shifted to Quality of Service (QoS) levels. This is important to consider in case of orchestrations that specify the control flow for multiple services. To this end, contractual guarantees and service level agreements (SLAs) [2] are critical to ensure adequate QoS performance.

QoS metrics being random variables, the treatment of contractual obligations tends toward probabilistic criterion [3]. Contractual obligations may be specified as varying percentile values of such distributions rather than "hard" values. In [4], composition and monitoring such contracts with stochastic dominance have been examined.

As metrics such as response time and throughput rates can have heavy tails, estimating extreme values becomes difficult with few observations. The *availability* of a web service might need contracts for extreme percentiles in the response time profile (99.99 percentile). For instance, an ambulance or disaster management web service must be available 24×7, indicating a high availability requirement. These values are dependent on sampled random values and can lead to high variance in contractual guarantees.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 557–565, 2011.

The use of *importance sampling* [5] is proposed as a solution to these problems. Disadvantages of conventional Monte-Carlo techniques such as high variance of percentile values may be eliminated. In case of heavy tailed distributions, unobserved extreme percentiles can be quantified with higher accuracy. These are stochastically "important" observations to estimate contractual deviations. These issues are demonstrated with the *Dell* example [6], a choreography involving *Dell Plant* and *Supplier* orchestrations. We study more accurate bounds for supplier contracts with varying plant demand rates. Further, we show how QoS metrics such as stock level deviations (specially long delays) can be estimated with low variance.

The rest of the paper is organized as follows: Section 2 introduces the probabilistic contract composition procedure for web services' QoS. Importance sampling is briefly introduced in Section 3 with emphasis on contractual sampling in web services and sample deviations. The Dell application is introduced in Section 4 with two workflows interacting in a choreography. The two application of importance sampling with respect to the Dell supply chain are described in Sections 4.1 and 4.2. Related work and conclusions of the paper are included in Sections 5 and 6, respectively.

2 Probabilistic QoS Contracts

Available literature on industry standards in QoS [7] provides a family of QoS metrics that are needed to specify SLAs. These can be subsumed into the following four general QoS observations: *Service Latency*, *Per Invocation Cost*, *Output Data Quality* and *Inter-Query Intervals*. To handle such diverse domains, metrics and algebra for QoS, a framework is proposed in [4]. Using such an algebra, QoS metrics may be defined explicitly with domains, increments and comparisons within service orchestrations.

For a domain $\mathbb{D}_Q$ of a QoS parameter Q, behavior can be represented by its distribution F_Q:

$$F_Q(x) = \mathbb{P}(Q \leq x) \tag{1}$$

Making use of stochastic ordering [8], this is refined for probability distributions F and G over a totally ordered domain $\mathbb{D}$:

$$G_Q \preceq F_Q \quad \iff \quad \forall x \in \mathbb{D}_Q, \quad G_Q(x) \geq F_Q(x) \tag{2}$$

That is, there are more chances of being less than x (partial order $\preceq$) if the random variable is drawn according to G than according to F. A QoS contract must specify the obligations of the two parties:

- The obligations that the orchestration has regarding the service are seen as *assumptions* by the service - the orchestration is supposed to meet them.
- The obligations that the service has regarding the orchestration are seen as *guarantees* by the service - the service commits to meeting them as long as assumptions are met.

Definition 1. *A probabilistic contract is a pair (Assumptions, Guarantees), which both are lists of tuples* $(Q, \mathbb{D}_Q, F_Q)$*, where* Q *is a QoS parameter with QoS domain* $\mathbb{D}_Q$ *and distribution* F_Q*.*

Once contracts have been agreed, they must be monitored by the orchestration for possible violation as described in [3].

3 Importance Sampling

In case of web services' SLAs, these rare event simulations can be used to determine the occurrence of failure or deviation from contracts. Traditional Monte-Carlo (MC) methods waste a lot of time in a region of the state space which is "far" from the rare set of interest. Modifying the underlying distributions to move "near" the states of interest provides a more efficient means of analysis. With typical Monte-Carlo (MC), if the mean $\mu = 10^{-5}$ and if we want the expected number of occurrences of this event to be at least 100, we must take approximately $N = 10^7$ runs. For lower values of N, not even a single occurrence of this event may be seen - leading to the faulty conclusion that the event does not occur.

Importance sampling (IS) [5] increases the probability of the rare event while multiplying the estimator by an appropriate likelihood ratio so that it remains unbiased. Consider the case of a random variable Q with probability density function (PDF) F_Q for which the probability of a rare event $\mathbb{P}(H(Q) > \Phi)$ is to be estimated. Here $H(Q)$ is a continuous scalar function and Φ is the threshold. Using Monte-Carlo, one generates independent and identically distributed samples $Q_1, Q_2, ...Q_N$ from the PDF F_Q and then estimates the probability:

$$\mathbb{P}_{MC} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{H(Q_i)>\Phi} \tag{3}$$

where $\mathbf{1}_{H(Q)>\Phi}$ is 1 if $H(Q) > \Phi$ and 0 otherwise. For a rare event, such a technique needs many runs for low variance estimates.

With Importance Sampling (IS) [5], variance can be reduced without increasing the number of samples. The idea is to generate samples $Q_1, Q_2, ...Q_N$ from an auxiliary PDF G_Q and then estimate probability:

$$\mathbb{P}_{IS} = \frac{1}{N} \sum_{i=1}^{N} H(Q_i) \mathbf{1}_{H(Q_i)>\Phi} \frac{F_Q(Q_i)}{G_Q(Q_i)} \tag{4}$$

It is evident that G_Q should be chosen such that it has a thicker tail than F_Q. If F_Q is large over a set but G_Q is small, then $\left(\frac{F_Q}{G_Q}\right)$ would be large and it would result in a large variance. It is useful if we can choose G_Q to be similar to F_Q in terms of shape. Analytically, we can show that the best G_Q is the one that would result in a variance that is minimized [5]. In order to perform this selection, some sort of knowledge about the distribution is assumed, either through theory or pre-collected statistical data.

As in the case of most statistical techniques, the monitoring of contracts is also based on *samples* of the *population* of QoS. If the variance in values of the sample set is large then the mean is not as representative of the data as if the spread of data is small. If only a sample is given and we wish to make a statement about the population standard deviation(from which the sample is drawn), then we need to use the sample standard deviation. If $Q_1, Q_2, ..., Q_N$ is a sample of N observations, the sample variance is given by:

$$s^2 = \frac{\sum_{i=1}^{N}(Q_i - \bar{Q}^2)}{N-1} \tag{5}$$

with $\bar{Q}$ as the sample mean. This sample standard deviation can be used to represent the deviation in the population QoS output and is used in this paper.

4 Dell Supply Chain

To demonstrate the variation in the QoS domains in real-world services, we study the *Dell* example [6]. The *Dell* application is a system that processes orders from customers interacting with the Dell webstore. According to [6], this consists of the following prominent entities:

- *Dell Plant* - Receive the orders from the Dell webstore and are responsible for the assembly of the components. For this they interact with the *Revolvers* to procure the required items.
- *Revolvers* - Warehouses belonging to Dell which are stocked by the suppliers. Though Dell owns the revolvers, the inventory is owned and managed by the *Suppliers* to meet the demands of the *Dell Plant*.
- *Suppliers* - They produce the components that are sent to the revolvers at Dell. Periodic polling of the *Revolvers* ensure estimates of inventory levels and their decrements.

Essentially, there are a *Dell Plant* and *Supplier* orchestrations that are choreographed through common *Revolvers*. The critical aspect in the Dell choreography is efficient management of revolver levels. It is a shared *buffer* resource that is accessed by both the Dell Plant and the Suppliers. As discussed in [6], for the efficient working of the supply chain, the interaction between the Dell Plant and the Supply-side workflows should be taken into account.

The requests made by the plant for certain items will be favorably replied to if the revolvers have enough stock. This stocking of the revolvers is done independently by the suppliers. The suppliers periodically poll (withdraw inventory levels) from the revolvers to estimate the stock level. In such a case, a contract can be made on the levels of stock that must be maintained in the revolver. The customer side agreement limits the throughput rate. The supplier side agreement ensures constant refueling of inventory levels, which in turn ensures that the delay time for the customer is minimized. Thus, it represents a *choreography* comprising two plant-side and supplier-side orchestrations interacting via the revolver as a shared resource.

4.1 Contract Composition

For the *Dell* example, as QoS metrics are inherent to the functionality of the choreography, specifying explicitly probabilities of outage is necessary. Proposed are the following two concrete metrics that qualitatively evaluate these workflows:

- **Assumption:** The *demand* (number of orders/hour) distributions from the Dell plant made to a particular revolver. It is the prerogative of the plant to maintain demand within acceptable range of the contracts.
- **Guarantee:** The *delay* (hours) distribution in obtaining products from revolvers. This, in turn, is dependent on the availability of products in the revolver. The suppliers ensure efficient and timely refueling to maintain acceptable delays in the supply chain.

Consider the *assumption* on the query rate of the customer shown as an exponential distribution as in Fig. 1. Repeatedly pinging the service in order to receive boundary values of the distribution is expensive and not reflective of run-time performance. This is demonstrated for three values in Table 1 with 10000 runs. Conventional Monte-Carlo does not detect the probability of inter-query periods being less than 100, 50 or 20 minutes (which can be fallaciously interpreted as the rare event never occurring). Using an importance sampling distribution, accurate mean and sampling variance values are produced for the probability of crossing these thresholds. Such a level of accuracy is needed specially for critical web services (crisis management such as ambulance or fire stations). For conventional web services contracts as well, such precise contractual obligations can reduce the need for extended monitoring of services contracts.

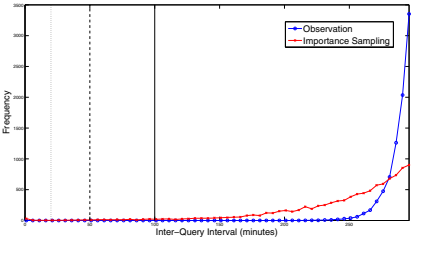


Fig. 1. Inter-query period fitting

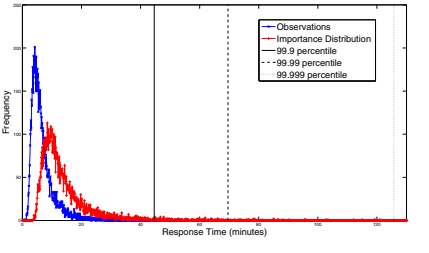


Fig. 2. Response time fitting

A corresponding *guarantee* from the service provider regarding the response time may be estimated as a long tailed distribution (Fig. 2, Table 2). Once again we concentrate on the outlying percentile values. The outputs for the traditional Monte-Carlo runs produce higher sample variance compared to the importance sampling scheme.

Table 1. Inter-query periods by Monte-Carlo (MC) and Importance Sampling (IS)

Inter-query period (mins.)	mean MC	variance MC	mean IS	variance IS
100	0	0	0.0086	0.0094
50	0	0	0.0018	7.36×10^{-5}
20	0	0	7.99×10^{-5}	7.37×10^{-6}

Table 2. Latency by Monte-Carlo (MC) and Importance Sampling (IS) schemes

Percentile	Latency (mins.)	mean MC	variance MC	mean IS	variance IS
99.9	44.61	0.0022	2.456×10^{-6}	0.0018	3.5548×10^{-7}
99.99	69.58	5.2×10^{-4}	5.65×10^{-7}	3.04×10^{-4}	3.82×10^{-8}
99.999	125.70	1.1×10^{-4}	1.19×10^{-7}	3.47×10^{-7}	3.12×10^{-9}

An advantage of this scheme is that the contracts will be formulated as explicit probabilities of contractual deviation. The WSLA framework [10], refined with precise probabilistic percentile values of QoS distributions specified as:

```
<Assumptions>
<SLAParameter name="InterQueryPeriod" type="float" unit="seconds" />
<Predicate xsi:type="wsla:Greater">
   <Percentile> 95 </Percentile>   <Value> 30 </Value>
   <SampleVariance> 10^-3 </SampleVariance>  </Predicate>  </Assumptions>
<Guarantees>
<SLAParameter name="ResponseTime" type="float" unit="seconds" />
<Predicate xsi:type="wsla:Less">
   <Percentile> 99 </Percentile>   <Value> 15 </Value>
   <SampleVariance> 10^-3 </SampleVariance>  </Predicate>  </Guarantees>
```

The contract now specifies the contract from the *assumption-guarantee* viewpoint. For any measurement period, the `Percentile` values of the `ResponseTime` should be less than the specified bounds. On the other hand, the `InterQueryPeriod` should be greater than the threshold values. In both cases, the `SampleVariance` is taken into account. Such a framework allows for distributions to be used for both contractual specification and monitoring deviations.

4.2 Forecasting

Traditional forecasting models like autoregressive moving averages [9] rely heavily on accurate mathematical modeling of workflow processes. In this section, we propose using pre-identified contracts / observations to provide an easier method of forecasting outages in web services orchestrations. Consider a Dell revolver with critical stock of 10 items, refueling batch 50 items and a polling period of 10 hours. With an *assumption* distribution of orders/hour shown in Fig. 3, the response time distribution obtained over a period of 1 week is shown in Fig. 4. If an item is available, it is procured immediately. Else, it is refueled with a supplier delay when polling detects sub-critical revolver levels.

In order to develop a *guarantee* distribution, the Dell plant must estimate the probability that delays over 72, 96 or 120 hours are experienced (leading to cancellation in orders). Through importance sampling, these values can be better estimated as in Table 3. Notice that the variance through importance

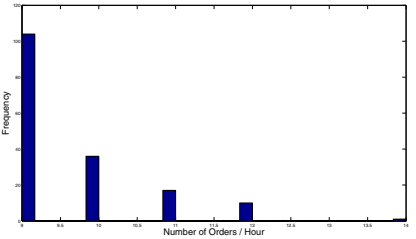


Fig. 3. Assumption: *Plant* side demand distributions

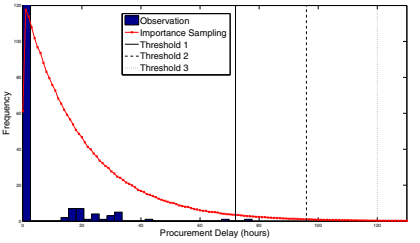


Fig. 4. Guarantee: *Supplier* side procurement delays

sampling is several orders of magnitude lower than conventional Monte-Carlo. The Dell plant can provision more stringent supplier obligations to reduce the delays. For instance, changing the critical stock to 50 items, refueling batch 200 items produces a new set of values, with lower probabilities of crossing outlying values as shown in Table 4.

Such changes produced by improved supplier performance is barely observed through traditional Monte-Carlo sampling, thus proving the efficacy of Importance Sampling. Application of forecasting through pre-negotiated contracts emphasize the need for precise contractual obligations needed in web services.

Table 3. Original contract estimates

Delay (hours)	mean MC	variance MC	mean IS	variance IS
72	0.002	3.2×10^{-3}	0.0016	1.72×10^{-7}
96	0	0	3.88×10^{-4}	3.71×10^{-8}
120	0	0	1.02×10^{-4}	6.25×10^{-9}

Table 4. Reformulated contract estimates providing lower probabilities of delay

Delay (hours)	mean MC	variance MC	mean IS	variance IS
72	0	0	4.08×10^{-4}	1.35×10^{-8}
96	0	0	9.91×10^{-5}	1.89×10^{-9}
120	0	0	2.734×10^{-5}	6.74×10^{-10}

5 Related Work

The use of probabilistic QoS and contracts was introduced by Rosario et al [3] and Bistarelli et al [11]. Instead of using hard bound values for parameters such as response time, the authors proposed a probabilistic contract monitoring approach to model the QoS bounds. The composite service QoS was modeled using probabilistic processes by Hwang et al [12] where the authors combine orchestration constructs to derive global probability distributions.

In [14], Gallotti et al propose using a probabilistic model checker to assess non-functional quality attributes of workflows such as performance and reliability. Validating SLA conformance is studied by Boschi et al [15]. A series of experiments to evaluate different sampling techniques in an online environment is studied.

The use of importance sampling to change probability of occurrence of events in well known [5]. An associated work in this area is importance splitting [13]. Importance splitting considers the estimation of a rare event by deploying several conditional probabilities during simulation runs, reducing the need to identify importance distributions as used in this case.

6 Conclusion

QoS aspects are critical to the functioning of most web service orchestrations and choreographies, needing more precise specifications of SLAs. This is difficult as distributions of QoS values have high variance when sampled with inefficient Monte-Carlo techniques. In most cases, the tails of QoS distributions are either neglected or averaged out in contractual specifications. Applying importance sampling to such distributions can provide better estimates of outlying values with relatively low variance. As demonstrated in this paper on the *Dell* supply chain application, importance sampling can have significant imperatives for both contract composition as well as forecasting deviations for critical services. The extension of this approach in case of WSLA specifications are also provided with a precise definition of sample variance.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer (2004)
2. Bhoj, P., Singhal, S., Chutani, S.: SLA management in federated environments. In: Symp. on Dist. Mgmt. for the Networked Millennium, pp. 293–308 (1999)
3. Rosario, S., Benveniste, A., Haar, S., Jard, C.: Probabilistic QoS and soft contracts for transaction based Web services. In: IEEE ICWS (2007)
4. Rosario, S., Benveniste, A., Jard, C.: Flexible Probabilistic QoS Management of Orchestrations. Int. J. Web Service Res. 7(2), 21–42 (2010)
5. Bucklew, J.A.: Introduction to rare event simulation. Springer, Heidelberg (2004)
6. Kapunscinski, R., Zhang, R.Q., Carbonneau, P., Moore, R., Reeves, B.: Inventory Decisions in Dells Supply Chain. Interfaces 34(3), 191–205 (2004)
7. World Wide Web Consortium, "QoS for Web Services: Requirements and Possible Approaches," W3C Working Group Note (November 2003)
8. Shaked, M., Shanthikumar, J.G.: Stochastic Orders. Springer Statistics (2006)
9. Makridakis, S., Wheelwright, S.: Adaptive Filtering: An Integrated Autoregressive/Moving Average Filter for Time Series Forecasting. Operational Research Quarterly 28(2), 425–437 (1977)
10. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. IBM Corporation (2003)

11. Bistarelli, S., Santini, F.S.: Soft Constraints for Quality Aspects in Service Oriented Architectures. In: Workshop on Service Oriented Computing, Italy (2009)
12. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. Elsevier Information Sciences 177, 5484–5503 (2007)
13. Morio, J., Pastel, R., Le Gland, F.: An overview of importance splitting for rare event simulation. European J. of Physics 31, 1295–1303 (2010)
14. Gallotti, S., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Quality Prediction of Service Compositions Through Probabilistic Model Checking. In: Becker, S., Plasil, F., Reussner, R. (eds.) QoSA 2008. LNCS, vol. 5281, pp. 119–134. Springer, Heidelberg (2008)
15. Boschi, E., Denazis, S., Zseby, T.: A measurement framework for inter-domain SLA validation. Elsevier Computer Communications 29, 703–716 (2006)

Particle Filtering Based Availability Prediction for Web Services

Lina Yao and Quan Z. Sheng

School of Computer Science
The University of Adelaide, Australia
{lina,qsheng}@cs.adelaide.edu.au

Abstract. Guaranteeing the availability of Web services is a significant challenge due to unpredictable number of invocation requests the Web services have to handle at a time, as well as the dynamic nature of the Web. The issue becomes even more challenging for composite Web services in the sense that their availability is inevitably affected by corresponding component Web services. Current Quality of Service (QoS)-based selection solutions assume that the QoS of Web services (such as availability) is readily accessible and services with better availability are selected in the composition. Unfortunately, how to real-time maintain the availability information of Web services is largely overlooked. In addition, the performance of these approaches will become questionable when the pool of Web services is large. In this paper, we tackle these problems by exploiting particle filtering-based techniques. In particular, we have developed algorithms to precisely predict the availability of Web services and dynamically maintain a subset of Web services with higher availability. Web services can be always selected from this smaller space, thereby ensuring good performance in service compositions. Our implementation and experimental study demonstrate the feasibility and benefits of the proposed approach.

1 Introduction

Web services and service-oriented computing (SOC) represent a new paradigm for building distributed computing applications over the Internet. Unfortunately, after the development of nearly one decade, Web services are still in their infancy [10,17,13]. According to a recent study in Europe [2], the Web currently contains 30 billion Web pages, with 10 million new pages added each day. In contrast, only 12,000 real Web services exist on the Web. Even worse, many Web services have been deployed with dependability problems (e.g., unexpected behaviors, reliability, availability etc).

Guaranteeing the availability of a Web service is a significant challenge due to the unpredictable number of invocation requests the Web service has to handle at a time, as well as the dynamic nature of the Web. Over the last few years, many works have emerged in solving Web service availability problem. Almost all of these approaches are based on the concept of *service community* where Web services with similar functionalities (but different non-functional properties such as quality of service (QoS)) [1,18] are grouped in a particular community. The basic idea on improving the availability of Web service in a composition is to substitute Web services with poor quality using other

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 566–573, 2011.

services with better quality from the same service community. This typically involves QoS based service selection.

Most QoS service selection approaches assume that the QoS information (e.g., availability of Web service) is pre-existing and readily accessible. This unfortunately is not true. In reality, the availability status, as well as other QoS properties, of a Web service is highly uncertain, which changes over the time. How to accurately estimate and predict the availability status of a Web service becomes an important research problem. In addition, given the wide adoption of Web service in industry, more and more Web services will be available and the size of service communities will be inevitable large. Selecting from such a big space will lead to performance problem. Ideally, low quality Web services should be automatically filtered during service composition.

In this paper, we focus on solving above problems. In particular, we propose a particle filter based approach to precisely predicate and adjust Web service availability in real time. By continuously monitoring the service status, our approach offers more efficient and effective solution in service composition while ensure the high availability of composite Web services. Our work can be summarized as the following three original contributions:

- A model for availability of Web services using particle filter technique, which can perform precise prediction of the availability of Web services. Service availability is considered by combining both historical information and the predicted availability.
- An algorithm to optimize Web services selection by dynamically reducing the candidate Web services search space during Web services composition, and
- An implementation and experimental studies to validate the proposed approach.

The rest of the paper is organized as follows. Section 2 briefly introduces service availability model and the particle filter techniques. Section 3 describes the details of our approach and the algorithms. Section 4 reports the implementation and some preliminary experimental results. Finally, Section 5 overviews the related work and Section 6 provides some concluding remarks.

2 The Service Availability Model and the Particle Filter

In this section, we briefly introduce the service availability model and the particle filter technique, which serves as the core component of our approach on high availability of Web services composition.

2.1 Modeling Web Services Availability

There are different classifications of availability and many ways to calculate it [3]. Almost all existing approaches (e.g., [19,8,4]) use *operational* availability that measures the average availability over a period of time (i.e., the ratio of the service uptime to total time). Although this is simple to calculate, it is hard to measure the availability of a Web service at a specific time.

In this work, we model Web service availability as *instantaneous* (or point) availability. The instantaneous availability of a Web service s is the probability that s will

be operational (i.e., up and running) at a specific time t. The following discusses how to calculate the instantaneous availability of a Web service.

At given time t, a Web service s will be available if it satisfies one of the following conditions:

- The Web service s is working in the time frame of [0,t] (i.e., it never fails by time t). We represent the probability of this case as $\mathcal{R}(s,t)$.
- The Web service s works properly since the latest repair at time u ($0 < u < t$). The probability of this condition is $\int_0^t \mathcal{R}(s,t-u)m(s,u)du$, where $m(s,u)$ is the renewal density function of service s.

Based on these two conditions, the availability of service s at time t, $\mathcal{A}(s,t)$, can be calculated using the following formula:

$$\mathcal{A}(s,t) = \mathcal{R}(s,t) + \int_0^t R(s,t-u)m(s,u)du \tag{1}$$

2.2 The Particle Filter

We consider the availability of Web services as a dynamic system (i.e., it changes from time to time), which can be modeled as two equations: *state transition* equation and *measurement* equation. The states can not be observed directly and need to be estimated, while the measurements can be observed directly. Specifically, state transition is represented as:

$$x_k = f_k(x_{k-1}, u_{k-1}, v_{k-1}) \tag{2}$$

where f_k is a non-linear function, x_k, x_{k-1} are current and previous states, v_{k-1} is the state noise in non-Gaussian distribution, and u_{k-1} is the known input. Similarly, measurement is represented as

$$z_k = h_k(x_k, u_k, n_k) \tag{3}$$

where h_k is a non-linear function, z_k is a measurement, x_k is a state, and u_k is the known input.

The availability of Web services changes over time, which is full of uncertainty due to problems of network issues, hosting servers, and even service requester environments. We exploit the generic particle filter [7] to solve the dynamic availability of Web services, which will be discussed in the next section.

3 The Approach

Figure 1 shows the basic idea of our approach. Specifically, we propose to add a *filtering layer* between Web service layer and composition layer (right side of Figure 1). The layer of Web services contains several service communities and each of them consisting of Web services with similar functionalities. Each community may have large number of members.

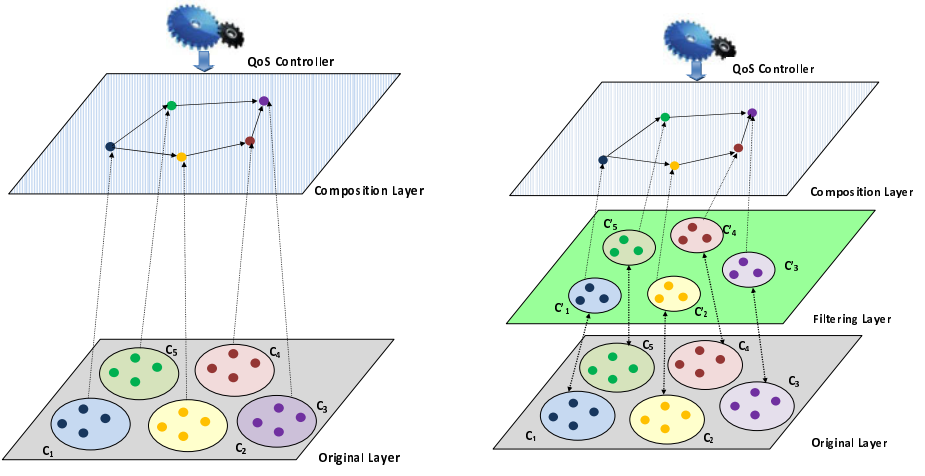


Fig. 1. (a) Existing approaches and (b) Our proposed approach

The filtering layer is essentially a subset of service communities, which consists of Web services with high availability that will directly involve in service compositions. The Web services are selected based on the accurate estimation and ranking algorithm described in this section. It should be noted that the relationship between Web service communities and the filtering layer is *dynamic* and *adaptive*. Our approach dynamically adjusts the members in the filtered service communities where degrading Web services will be replaced automatically with Web service with better availability from service communities. Web services' availability state is highly dynamic and therefore needs an adaptive approach to monitor and track each Web service's state. This is important to conduct optimized selection algorithm for composite Web services.

In our approach, we model the availability of a Web service i at time t as $x_i(t)$, which maintains the probability distribution for service availability estimation at time t, and inducted as the belief $Bel(x_i(t)) = \{x_i(t), w_i(t)\}$, $i = 1, 2, ..., \mathcal{M}$, where $w_i(t)$ are the different weight values, which indicate the contribution of the particle to the overall estimation, also called *important factors* ($\sum w_i(t) = 1$). Algorithm 1 shows the brief process on how it works.

Based on Algorithm 1, we can sort the top k Web services with high availability according to the monitoring and prediction. We call this estimated availability $\mathcal{E}_i$. In addition, for the overall filtering algorithm, we also take the history information on availability $\mathcal{H}_i$ into account, on top of the estimated availability by using the particle filter technique. The historical fluctuation of Web services availability has important impact on the current availability of the services. We call this historical fluctuation $\mathcal{H}$ impact as *availability reputation*. The most common and effective numerical measure of the center tendency is using the $mean$, however, it is sensitive to the extreme values

Algorithm 1. Particle Filter based Algorithm

1. **Initialization:** compute the weight distribution $\mathcal{D}_w(a)$ according to IP address distribution.
2. **Generation:** generate the particle set and assign the particle set weight, which means $\mathcal{N}$ discrete hypothesis

- generate initial particle set $\mathcal{P}_0$ which has $\mathcal{N}$ particles, $\mathcal{P}_0$ = $(p_{0,0}, p_{0,1}, ...p_{0,\mathcal{N}-1})$ and distribute them in a uniform distribution in the initial stage. Particle $p_{0,k} = (a_{0,k}, weight_{0,k})$ where a represents the Web service availability.
- assign weight to the particles according to our weight distribution $\mathcal{D}_w(a)$.

3. **Resampling:**

- Resample $\mathcal{N}$ particles from the particle set from a particle set $\mathcal{P}_t$ using weights of each particles.
- generate new particle set $\mathcal{P}_{t+1}$ and assign weight according to $\mathcal{D}_w(a)$

4. **Estimation:** predict new availability of the particle set $\mathcal{P}_t$ based on availability function $f(t)$.
5. **Update:**

- recalculate the weight of $\mathcal{P}_t$ based on measurement m_a, $w_{t,k}$= $\prod(\mathcal{D}_w(a))(\frac{1}{\sqrt{2\pi\phi}})exp(-\frac{dx_k^2 + dy_k^2}{2\phi^2})$, where $\delta a_k = m_a - a_{t,k}$
- calculate current availability by mean value of $p_t(a_t)$

6. Go to step 3 until convergence

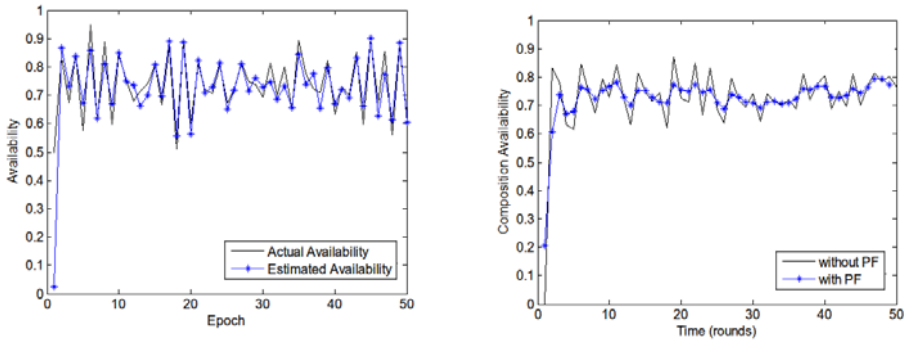
Algorithm 2. Overall Adaptive Filtering Algorithm

Input: initial availability values, α, τ.
Output: predicted availability, referencing availability, candidate list.
1. Read in the initial parameters;
2. Calculate each values for Web service $a_{ij}(s,t)$ in Web service community j at time t;
3. Predict the availability state of next time slot using particle filter (see Algorithm 1);
4. Looking up database and calculate the *mean* values of availability $\mathcal{H}$.
5. Calculating the reference availability $\mathcal{R}$.
6. Update the top k candidate list in each Web services community for every time interval τ;
7. Go to step 2.

(e.g., outliers) [5]. In our work, we define the final availability of a Web service as *reference availability* $\mathcal{R}$, which is calculated using:

$$\mathcal{R}_i(\tau) = \alpha\mathcal{E}_i(\tau) + (1-\alpha)\mathcal{H}_i(\sum_{1}^{\tau-1}(\tau-1)) \tag{4}$$

where $\alpha \in [0,1]$ is the weight and users can assign different weight based on their different preference, τ is a time span which can be defined by users. Finally, we summarize the overall particle filter algorithm in Algorithm 2.



(a) (b)

Fig. 2. (a) Actual availability vs estimated availability and (b) Availability of a composite Web service

4 Experimental Results

The proposed approach has been implemented in a prototype system in Java. In this section, we present two experimental results. For the experiments, we simulated 500 Web services of five different Web service communities (i.e., 100 Web services for each service community). We set the failure probability for the Web services as 3.5 percent, which complies with the findings in [6].

The first experiment studies the estimation accuracy of our approach, we simulated Web services' availability fluctuation and tracked their fluctuation of availability for 50 time steps (each time step counted as an *epoch*). The actual availability of Web services and corresponding estimated availability using our particle filter approach were collected and compared. Figure 2 (a) shows the result of one particular Web service. From the figure, we can see that our approach works well in tracing and predicting the availability of Web services.

The second experiment studies the impact our approach brought to the availability of composite Web services. We randomly generated composite Web services by composing services from five different communities. We simulated a comparatively significant fluctuation on the availability (i.e., changes in availability) of Web services for 50 different rounds and collected the availability information of the composite services under the situations of i) using our approach and ii) without using our approach. The availability of a composite Web service, $\mathcal{A}_c$, is represented as the mean value of its component Web services, i.e., $\mathcal{A}_c(c,t) = \alpha(\sum_{i=1}^{n} \mathcal{A}(s_i,t))/n$. Figure 2 (b) shows the availability of a particular composite Web service. From the figure we can see that the availability of the composite Web service is more stable when using our approach. In contrast, without using our approach, its availability is very sensitive to the fluctuations of service availability.

5 Related Work

There is a large body of research work related to the topic we have discussed in this paper. One important area on achieving high availability of Web services focuses on replication technology [11,12,14]. Serrano et al. [12] discuss an autonomic replication approach focusing on performance and consistency of Web services. Salas et al. [11] propose a replication framework for highly available Web services. Sheng et al. [14] further developed the idea by proposing an on-demand replication decision model that offers the solution to decide how many replicas should be created, when and where they should be deployed in the dynamic Internet environment. While these approaches focus on improving service availability through replication, our work concentrates on monitoring and predicting service availability. Our work is complementary to these works in the sense that the estimations provide a good source of information for replication decisions.

Many works achieve high availability of Web services based on the concept of *service communities* where Web services are selected based on QoS [8,19,16,9]. The basic idea is that services with similar functionalities are gathered as communities. If a Web service is unavailable, another service will be selected. However, most approaches assume that QoS is readily accessible and ignore its dynamic nature.

The works presented in [4,15] are the most similar ones to our work. In [4], Guo et al. model a composition process into the Markov Decision Process and use Kalman Filter to tracking the state of composite Web services. Sirin et al. [15] propose a filtering methodology that exploit matchmaking algorithms to help users filter and select services based on semantic Web services in composition process. However, these works focus on adaptive maintaining the composition of Web services and do not pay attention on the availability of component Web services. Our approach uses particle filter to precisely predict the availability of Web services and dynamically maintains a subset of Web services with higher availability, from which service developers can choose in their compositions.

6 Conclusion

Despite active development and research over the last decade, Web service technology is still not mature yet. In particular, guaranteeing the availability of Web services is a significant challenge due to unpredictable number of invocation requests the Web services have to handle at a time, as well as the dynamic nature of the Web. Many existing approaches ignore the uncertain nature of service availability and simply assume that the availability information of a Web service is readily accessed. In this paper, we have proposed a novel approach to monitor and predict Web service's availability based on particle filter techniques. Furthermore, we have developed algorithms to filter Web services for efficient service selection. The implementation and experimental results validated our approach.

Our ongoing work includes validating our approach on real Web services, conducting more experiments to study the performance of our approach (e.g., scalability). We also consider to extend our approach to other important service dependability properties (e.g., reputation, reliability, security).

References

1. Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing 7(1) (January/February 2003)
2. Domingue, J., Fensel, D.: Toward A Service Web: Integrating the Semantic Web and Service Orientation. Service Web 3.0 Project, `http://www.serviceweb30.eu`
3. Elsayed, A.: Reliability Engineering. Addison-Wesley (1996)
4. Guo, H., Huai, J.-p., Li, Y., Deng, T.: KAF: Kalman Filter Based Adaptive Maintenance for Dependability of Composite Services. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 328–342. Springer, Heidelberg (2008)
5. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2006)
6. Kim, S., Rosu, M.: A Survey of Public Web Services. In: Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York, NY, USA (May 2004)
7. Kitagawa, G.: Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models. Journal of Computational and Graphical Statistics 5(1), 1–25 (1996)
8. Liu, Y., Ngu, A., Zeng, L.: QoS Computation and Policing in Dynamic Web Service Selection. In: Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York, NY, USA (May 2004)
9. Maamar, Z., Sheng, Q.Z., Benslimane, D.: Sustaining Web Services High Availability Using Communities. In: Proceedings of the 3rd International Conference on Availability, Reliability, and Security (ARES 2008), Barcelona, Spain (March 2008)
10. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer 40(11), 38–45 (2007)
11. Salas, J., Pérez-Sorrosal, F., Patiño-Martínez, M., Jiménez-Peris, R.: WS-Replication: A Framework for Highly Available Web Services. In: Proceedings of the 15th International Conference on World Wide Web (WWW 2006), Edinburgh, Scotland (May 2006)
12. Serrano, D., Patiño-Martìnez, M., Jimenez-Peris, R., Kemme, B.: An Autonomic Approach for Replication of Internet-based Services. In: Proceedings of the 27th IEEE International Symposium on Reliable Distributed Systems (SRDS 2008), Napoli, Italy (October 2008)
13. Sheng, Q.Z., Maamar, Z., Yahyaoui, H., Bentahar, J., Boukadi, K.: Separating Operational and Control Behaviors: A New Approach to Web Services Modeling. IEEE Internet Computing 14(3), 68–76 (2010)
14. Sheng, Q.Z., Maamar, Z., Yu, J., Ngu, A.H.: Robust Web Services Provisioning Through On-Demand Replication. In: Proceedings of the 8th International Conference on Information Systems Technology and Its Applications (ISTA 2009), Sydney, Australia (April 2009)
15. Sirin, E., Parsia, B., Hendler, J.: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems 19(4), 42–49 (2004)
16. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A QoS-Aware Selection Model for Semantic Web Services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 390–401. Springer, Heidelberg (2006)
17. Yu, Q., Bouguettaya, A., Medjahed, B.: Deploying and Managing Web Services: Issues, Solutions, and Directions. The VLDB Journal 17(3), 537–572 (2008)
18. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality Driven Web Services Composition. In: Proceedings of The 12th International World Wide Web Conference (WWW 2003), Budapest, Hungary (2003)
19. 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)

A Penalty-Based Approach for QoS Dissatisfaction Using Fuzzy Rules

Barbara Pernici[1], S. Hossein Siadat[1], Salima Benbernou[2], and Mourad Ouziri[2]

[1] Politecnico di Milano, Italy
[2] LIPADE, Université Paris Descartes, France

Abstract. Quality of Service (QoS) guarantees are commonly defined in Service Level Agreements (SLAs) between provider and consumer of services. Such guarantees are often violated due to various reasons. QoS violation requires a service adaptation and penalties have to be associated when promises are not met. However, there is a lack of research in defining and assessing penalties according to the degree of violation. In this paper, we provide an approach based on fuzzy logic for modelling and measuring penalties with respect to the extent of QoS violation. Penalties are assigned by means of fuzzy rules.

Keywords: QoS, service level agreement, penalty, fuzzy logic.

1 Introduction

QoS guarantees defined in contracts may be violated due to various reasons. This situation needs to be handled through applying adaptation techniques not to bring dissatisfaction. The concept of penalty has been used in SLAs to compensate the conditions under which guarantee terms are not met [1]. Despite some research have been done on the description, negotiation and monitoring of SLAs, however there is not much work on the definition of penalty clauses. [4] studied on WS-Agreement specification to define penalties based on different types of violation. However, penalties are assigned to violation of a single property instead of assigning penalties to violation of overall QoS. Moreover, the approach introduces a method for measuring penalties which is for fixed predefined number of violations, instead of measuring the extent of violation and assigning penalties accordingly.

One main issue is how to determine the appropriate amount of penalties as compensations from providers to satisfy customers. As quality parameters can be satisfied partially, the assessment of penalties can be based on the degree of quality violation. Understanding the violation degree is a prerequisite for assessing penalties. However, measuring such violation is yet an open research challenge. In addition, the influencing factors in defining penalties need to be identified. A static amount of penalty (manual approaches) does not reflect the extent of violation at runtime. The amount and level of penalties are related to the degree of quality violation provided from the provider side. On the other side, the customers characteristics may also affect the amount of penalties. For

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 574–581, 2011.

example a penalty to satisfy a gold/loyal customer is different with the one for an occasional customer. To the best of our knowledge, there is no formal relation between the assigned penalty and its influencing factors. Moreover, the extent and type of penalties are not clearly expressed in related work. However, understanding such relation and providing a mapping between them are complicated issues. We argue what is missing is a suitable mechanism for modelling penalties that takes into account both provider and consumer sides. Apart from the degree of violation, we also consider the state of customer and service provider with respect to their past history (e.g. whether the service has been penalised previously) in determining the right amount of penalties. However, as the relation between a given penalty and its influencing factors is not linear, conventional mathematical techniques are not applicable for modelling penalties.

Recent approaches are dealing with the issue of partial satisfaction for quality commitments and different techniques were used such as applying soft constraint [6], fuzzy sets [3] and semantic policies [2]. Among them, [6] introduced the concept of penalties for unmet requirements. However, defining penalties and finding a relation between the assigned penalties and the violated guarantees are remained challenges in similar approaches. The goal of this paper is to apply an inference technique using fuzzy logic as a solution [5] to propose a penalty-based approach for compensating conditions in which quality guarantees are not respected. Fuzzy logic is well suited for describing QoS and measuring quality parameters [3]. We demonstrate a penalty inference model with a rule-based mechanism applying fuzzy set theory. Measuring an appropriate value for penalties with respect to the amount of violation is the main contribution of the paper.

In the following, we start by a motivating example in Section 2. In Section 3 we show the descriptions of penalties and in Section 4 we provide a rule-based system using fuzzy set theory for modelling and reasoning penalties. Section 5 shows some experiments in applying penalty for the problem of QoS dissatisfaction and we conclude the paper in Section 6.

2 Motivating Example

Let's assume that a user is wishing to use a food delivery service. Therefore, a contract is established between the user and the service provider. The contract defines non functional criteria such as delivery time, quality of the perceived service (the quality of food during the delivery service, for example the food is maintained at the ideal temperature), and availability of the delivery service. Therefore we define a list of parameters for our example as follows: time to delivery (t_d), quality of delivered food (q_d), availability of delivery service (a_d). These quality parameters together with a list of penalty terms are defined in a contract and illustrated in Table 1.

The delivery service will be penalized if it is not able to provide the quality ranges defined in the contract. An overall QoS violation will be calculated first and afterwards a penalty is assigned with respect to the extent of the violation.

Table 1. Motivating example

Quality Parameters	time to delivery	between 10 to 15 min
	quality of delivered food	between 0.8 to 1
	availability	between 90 to 100% of the time
Penalties:	Minor or Null penalties	
	Penalties on quality parameters	
	Extra Service penalties	
	Termination penalty	

We also take into account customer and provider perspectives by considering parameters from both parties. Parameters such as history of a delivery service and state of a customer can be involved. The history of a service shows whether the service is penalized previously. This can influence the amount of given penalties for future. The current state of a customer presents the importance of the customer for service provider. For example, minor violation of service delivery can cause a major penalty for provider in case the customer is gold (with good history). In contrast, a normal customer (with ordinary history) will not be given any extra service if the quality of delivered food is not good.

3 Definition of Penalties

In order to provide a formal model of penalties and build a reasoning mechanism to handle the penalties in the contract, in the following we try to summarize the different types of penalties that can be applied. We categorize the penalties into two main classes:

1. **Numerical penalties:** They are related to measurable qualities of service. In other words, we have to handle and work with variables of the service (e.g *the availability* > 0.9, the *responsetime* $< 0.2ms$).

2. **Behavioural penalties:** They are related to the behaviour of either the customer or the service provider. Consider the following case: a merchant wishes to obtain a service for online payment by bank card. The financial institution offered a 25% off if the settlement proceeds within two days of the request. Beyond these two days, the penalty is such as the trader does not have the discount and will therefore be required to pay all fees.

A penalty clause in an SLA may be of the following types:

- Penalty Null and denoted by P_0: no penalty is triggered because all agreed QoS are satisfied or minor violation has occurred.
- Penalty on the QoS: a penalty should be triggered on one of the QoS parameters Q_j in the contract if Q_i is not fulfilled.
- Penalty on the penalty: a new penalty P_j should be triggered if the previous one P_i is unfulfilled. Such penalty will be handled through the long term contract validation. The reasoning on the time aspect of the contract is out of the scope of the paper.

– Extra service penalty: if a QoS is not fulfilled by the service provider, to penalize him, an extra service might be offered to the customer.
– Cancellation penalty: this is a dead penalty for the service provider. A service substitution occurs.

4 Modelling Penalties

We present a fuzzy model to express penalties in a rule-base system. Our fuzzy penalty model is defined by the couple $FP =< \mathcal{S}, \mathcal{R} >$, where $\mathcal{S}$ is a fuzzy set on penalties and $\mathcal{R}$ is a set of inference rules.

4.1 Fuzzy Sets for Penalties

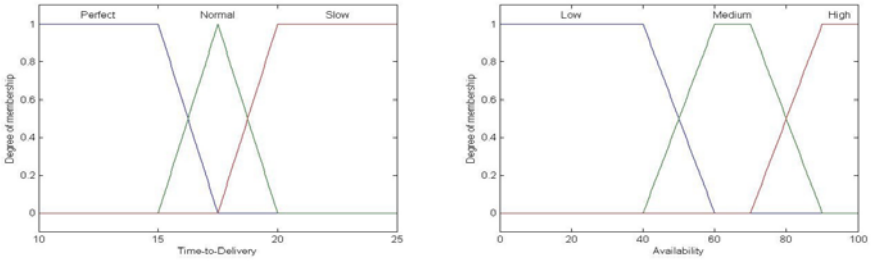
Our knowledge system includes linguistic variables defined by tuple $(\mathcal{Q}, \mathcal{C}, \mathcal{H}, \mathcal{P})$, where $\mathcal{Q}$ is a set of QoS parameters defined by fuzzy parameters as $\mathcal{Q} = \{t_d, q_d, a_d\}$ where t_d is the time to delivery, q_d is the quality of the delivered service and a_d is the availability of the delivery service. $\mathcal{C}$ is the current state of the customer, $\mathcal{H}$ is the history of the service to show whether the service is penalized previously and $\mathcal{P}$ is the set of penalties. We define these linguistic variables by fuzzy sets in the following.

The linguistic parameter of customer is defined by three fuzzy sets as in $\mathcal{C} = \{Normal, Silver, Gold\}$. We define two fuzzy sets to represent the state of service with respect to previous penalties as in $\mathcal{H} = \{Penalized, Not - penalized\}$. Finally penalties are described by five fuzzy sets to show the diverse range of penalties as in $\mathcal{P} = \{Null, Minor, Average, Major, Termination\}$, where *null* is no penalty, and *termination* is the situation in which the customer will terminate his contract with the delivery service. A fuzzy set represents the degree to which an element belongs to a set and it is characterized by membership function $\mu_{\tilde{A}}(x) : X \mapsto [0, 1]$. A fuzzy set $\tilde{A}$ in X is defined as a set of ordered pairs

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X, \mu_{\tilde{A}}(x) \in [0, 1]\} \tag{1}$$

where $\mu_{\tilde{A}}(x)$ is the membership function of x in $\tilde{A}$. Therefore, a membership function shows the degree of affiliation of each parameter by mapping its values to a membership value between 0 and 1.

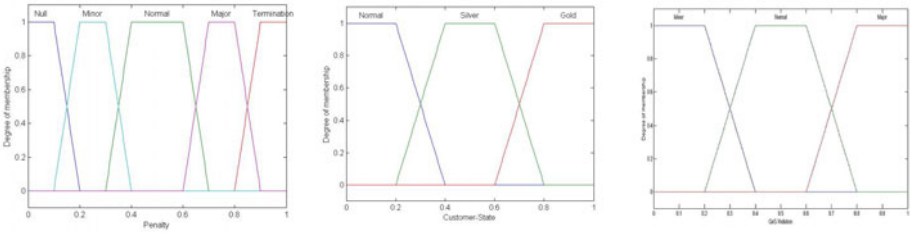
We associate membership functions to a given fuzzy set to define the appropriate membership value of linguistic variables. We start by providing membership functions for quality parameters from the motivating example. We take an approach that calculate an overall degree of violation with respect to the violation of each quality parameters. This way, we perform a trade-off mechanism and quality parameters are not treated independently. For each quality parameter a membership function is provided to show the degree of their satisfaction. We define three linguistic variables for each parameters such that t_d belongs to the set $\{Slow, Normal, Perfect\}$ and q_d is in the set $\{Unacceptable, Bad, Good\}$ and a_d is in the set $\{Low, Medium, High\}$. Figure 1 depicts the membership



(a) Time-to-Delivery membership function (b) Availability membership function

Fig. 1. Membership function for quality parameters

functions of time to delivery (a) and service availability (b). The functions are defined according to the contract and by an expert of the system. For example, the time to delivery between 10 to 15 min is *perfect*, between 15 to 20 min is *good* and more than 20 min is *slow*. Membership functions of penalty and customer state are shown in Figure 2 in (2a) and (2b) respectively.



(a) Penalty mf (b) Customer-state mf (c) QoS-violation mf

Fig. 2. Membership function for penalty ,state of the customer and QoS violation

4.2 Inference Rules on Penalties

The inference rules to trigger penalties are expressed as follows:

– $R_{\mathcal{Q}}$: QoS-based penalty rules. These are rules that reflect the violation of quality parameters. Penalties will be applied to a service if QoS guarantees stipulated in SLA are not fulfilled. It will be presented formally by $R_{\mathcal{Q}} : \mathcal{Q} \rightarrow \mathcal{P}$.

For instance, in the SLA, the delivery service agreed with the customer: $10mns \leq delivery\ time \leq 15mns$ and *good* quality of delivered food. If the QoS delivery time is not fulfilled (partially), then penalty p_{e1} (e.g. 10% discount) will be applied. Depending on the severity of the violation a *harder* penalty might be applied. For example, if both QoS are not fulfilled then penalty p_{e2} (e.g. 20% discount) will be applied. The fuzzy inference system gives us such degrees for penalties. Both cases are presented respectively below by rules:

- R1 $(t_d = Slow) \wedge (q_d = Good) \rightarrow p_{e1}$
- R2 $(t_d = Slow) \wedge (q_d = Bad) \rightarrow p_{e2}$

– $R_{\mathcal{P}}$: penalty on penalty rules. These rules reflect whether the service was given a penalty. If a service was penalized previously and again does not fulfil a QoS, then a penalty will be *harder*. It will be presented formally by $R_{\mathcal{P}} : \mathcal{Q} \times \mathcal{P} \rightarrow \mathcal{P}$ such that $R_{\mathcal{P}}(q, p_1) = p_2 \Rightarrow p_1 \prec p_2$.
For instance, let us consider a service having a penalty p_{e1} w.r.t rule $R1$ and again provides a slow delivery time, then the penalty p_{e3} (e.g. 10% discount plus free delivery) will be applied. The rule can be presented as below:
  - R3 $(t_d = Slow) \wedge p_{e1} \rightarrow p_{e3}$

– $R_{\mathcal{C}}$: customer-related penalty rules. The rules defined here will be adapted according to a customer qualification. Such rules will be presented formally by $R_{\mathcal{C}} : \mathcal{Q} \times \mathcal{P} \times \mathcal{C} \rightarrow \mathcal{P}$.
For instance, if the provided QoS is not fulfilled knowing that a penalty is assigned to the service, and if a customer is gold (has a good history), then extra service penalty p_{e4} (giving some extra service to the gold customer e.g. one movie ticket) will be harder than the one applied for normal customer p_{e3}. The rules can be presented as below:
  - R4 $(t_d = Slow) \wedge p_{e1} \wedge (C = Normal) \rightarrow p_{e3}$
  - R5 $(t_d = Slow) \wedge p_{e1} \wedge (C = Gold) \rightarrow p_{e4}$

5 Experiments and Implementation

We have simulated our approach in a simulator based on fuzzy inference system. Initial membership functions were designed based on the contract in the motivating example and fuzzy rules are defined by the expert of the system. Figure 2c illustrates membership function for QoS violation (see [3] for further details). Having defined the QoS violation, we measure the extent of penalties taken into account the state of customers and previously applied penalties for the same service. For this, fuzzy rules are defined considering all three influencing factors. Figure 3 depicts fuzzy rules for penalty based on QoS violations, customer's state and service status with respect to previous penalties which are defined by the *service-state* parameter represented by fuzzy set $\{Penalized, Not - penalized\}$.

For example rule no. 8 shows that a major penalty will be given to a silver customer if major violation occurs from defined QoS, while rule no. 7 will give a normal penalty (has lesser effect than major penalties) to the normal customer when the same amount of violation happens. The role of service-state can be seen in the rule, e.g. by comparing the rule no. 5 with the rule no. 14. In general, a harder penalty will be given to the service which is already penalized from the provider side.

The inference system calculates the degree of penalty by applying all the rules in a parallel approach for given input values of influencing factors. For example assume a QoS violation of 0.7 which has a membership degree of 0.5 for both

5. If (QoS-Violation is Average) and (Customer-State is Silver) and (Service-State is Not-penalized) then (Penalty is Normal) (1)
6. If (QoS-Violation is Average) and (Customer-State is Gold) and (Service-State is Not-penalized) then (Penalty is Major) (1)
7. If (QoS-Violation is Major) and (Customer-State is Normal) and (Service-State is Not-penalized) then (Penalty is Normal) (1)
8. If (QoS-Violation is Major) and (Customer-State is Silver) and (Service-State is Not-penalized) then (Penalty is Major) (1)
9. If (QoS-Violation is Major) and (Customer-State is Gold) and (Service-State is Not-penalized) then (Penalty is Termination) (1)
10. If (QoS-Violation is Minor) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Minor) (1)
11. If (QoS-Violation is Minor) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Normal) (1)
12. If (QoS-Violation is Minor) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Major) (1)
13. If (QoS-Violation is Average) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Normal) (1)
14. If (QoS-Violation is Average) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Major) (1)
15. If (QoS-Violation is Average) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Termination) (1)
16. If (QoS-Violation is Major) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Major) (1)
17. If (QoS-Violation is Major) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Termination) (1)
18. If (QoS-Violation is Major) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Termination) (1)

Fig. 3. Fuzzy rules for penalty based on QoS violations, customer's state and previous penalties on the service

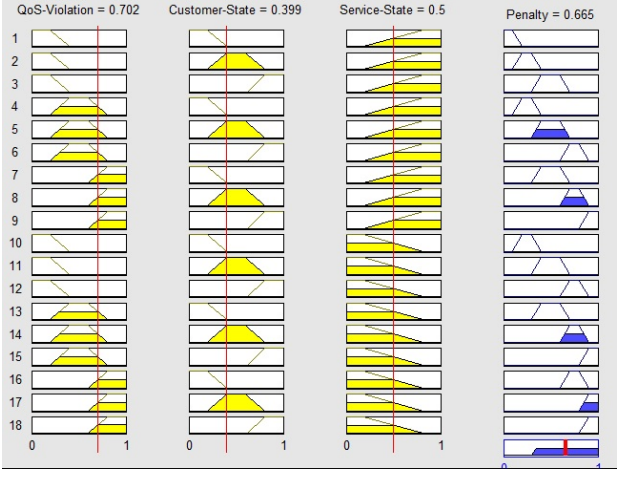


Fig. 4. A view of the inference system for applying penalties

normal and *major* fuzzy sets (according to their membership functions presented in the figure 2c). Such a violation, can trigger all the rules that include normal and major QoS violations. Note that the result of each rule depends on the membership degrees of other linguistic variable. For this example, rules with *minor* QoS-violation are not triggered at all. This situation is demonstrated in Figure 4. The result of each rule is integrated with an aggregation method to include the effect of all the rules. Figure 5 depicts a plot showing the penalties regarding QoS violation and customer's state. The figure represents possible values for penalties after defuzzification for all values of QoS violation and customer's state. For example, for the QoS violation of 0.7 and customer-state of 0.4 the penalty degree is 0.66 which is shown in the figure. The relation between QoS violation and customer's state can also be seen in the figure.

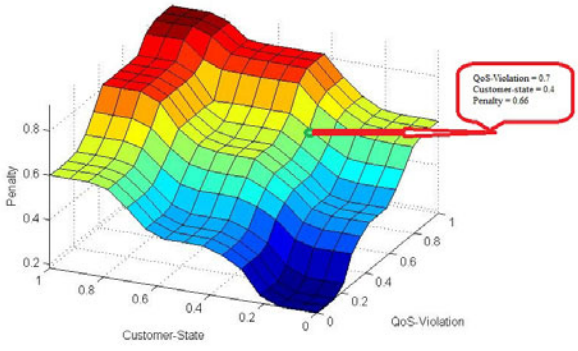


Fig. 5. The plot showing the penalties regarding QoS violation and customer's state

6 Conclusions and Future Work

Applying penalties is a complex research issue in service oriented computing which has not been paid enough attention in the literature. In this work, we elaborated the concept of penalty and propose a mechanism for modelling and measuring penalties. Penalties are modelled using a fuzzy approach and applying fuzzy set theory. The relation between penalties and their influencing factor are defined by fuzzy rules through an inference method. We have demonstrated the proposed penalty model through a motivating example and performed some initial result in measuring penalties.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

1. Andrieux, A., et al.: Web Services Agreement Specification (WS-Agreement). Recommended standard, Open Grid Forum (March 2007)
2. Li, P., Comerio, M., Maurino, A., De Paoli, F.: Advanced non-functional property evaluation of web services
3. Pernici, B., Siadat, S.H.: A Fuzzy Service Adaptation Based on Qos Satisfaction. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 48–61. Springer, Heidelberg (2011)
4. Rana, O., Warnier, M., Quillinan, T.B., Brazier, F., Cojocarasu, D.: Managing violations in service level agreements (2008)
5. Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338–353 (1965)
6. Zemni, M.A., Benbernou, S., Carro, M.: A Soft Constraint-Based Approach to Qos-Aware Service Selection. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 596–602. Springer, Heidelberg (2010)

Cellular Differentiation-Based Service Adaptation

Ichiro Satoh

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ichiro@nii.ac.jp

Abstract. This paper proposes an approach to adapting services in a distributed system whose computational resources are dynamically changed. It supports the notions of cellular differentiation and dedifferentiation. When a service delegates a function to another component coordinating with it, if the former has the function, this function becomes less-developed and the latter's function becomes well-developed. When some differentiated services are not available, it enables remaining services to automatically support the functions provided from the unavailable services. The approach was constructed as a middleware system and allowed us to define agents as Java objects. We present several evaluations of the framework in a distributed system.

1 Introduction

Cellular differentiation is the mechanism by which cells in a multicellular organism become specialized to perform specific functions in a variety of tissues and organs. Different kinds of cell behaviors can be observed during embryogenesis: cells double, change in shape, and attach at and migrate to various sites within the embryo. This paper introduces the notion of cellular differentiation into distributed systems as an adaptive approach. Distributed systems tend to be dynamic by nature because units, e.g., computers and networks, may fail and new units may have to be added to include new resources, applications, or users. Furthermore, such units tend to be heterogeneous and have limited computational resources. A federation of services running on multiple computers whose computational resources are different is needed to support functions beyond the capabilities of individual computers.

Our approach involves service matching as the service differentiation factors in services. When a service delegates a function to another service, if the former has the function, its function becomes less-developed and the latter's function becomes well-developed. As a result, differentiated services become specialized or degenerated according to demands from other services. Furthermore, the approach also supports the concept of *dedifferentiation* for managing contingencies, e.g., network partitioning and system or service failures, where dedifferentiation is the process of reverting partially or terminally differentiated cells to an earlier developmental stage.

2 Background

Service composition involves the development of customized services often by discovering, integrating, and executing existing services. Most existing work on service

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 582–589, 2011.

composition has been designed to support coordination/mashups between services rather than adapting service themselves. Several researchers have explored genetic computation/programming [5] and swarm intelligence [2]. Such approaches have often been too diverse, but real systems may have no chance of ascertaining the fitness of randomly generated parameters or programs, and many agents because they have an effect on the real world and no surplus computational resources. There have been several attempts to support software adaptation in the literature on adaptive computing. Blair et al. [1] tried to introduce self-awareness and self-healing into a CORBA-compatible Object Request Broker (ORB). Their system had a meta-level architecture with the ability of dynamically binding CORBA objects. Several researchers had proposed bio-inspired middleware for dynamic and large-scale networks [7]. Although they introduced the notion of energy into distributed systems and have enabled agents to be replicated, moved, and deleted according to the number of service requests, they have had no mechanism for adapting agents' behaviors unlike ours.

3 Basic Approach

Our approach assumes that each service is defined as an autonomous software component, called an *agent*, that is like a cell and consists of one or more functions that can be invoked by itself or other agents. The approach introduces the undertaking/delegation of behaviors in agents from other agents as a differentiation factor. Behaviors in an agent, which are delegated from other agents more frequently in a cell, are well developed, whereas other behaviors, which are delegated from other agents less frequently, in the cell are less developed. Finally, the agent only provides the former behaviors and delegates the latter behaviors to other agents.

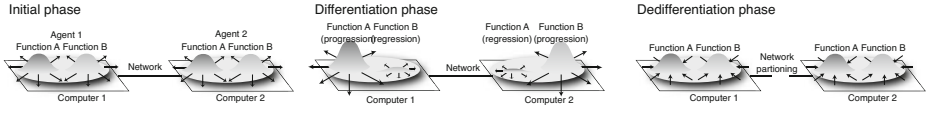


Fig. 1. Differentiation mechanism for software configuration

Differentiation: When dictyostelium discoideum cells aggregate, they can be differentiated into two types: prespore cells and prestalk cells. Each cell tries to become a prespore cell and periodically secretes cAMP to other cells. If a cell can receive more than a specified amount of cAMP from other cells, it can become a prespore cell. There are three rules. 1) cAMP chemotaxically leads other cells to prestalk cells. 2) A cell that is becoming a prespore cell can secrete a large amount of cAMP to other cells. 3) When a cell receives more cAMP from other cells, it can secrete less cAMP to other cells (Fig. 1).

Each agent has one or more functions with weights, where each weight corresponds to the amount of cAMP and indicates the superiority of its function. Each agent initially intends to improve all its functions and periodically multicasts *restraining* messages

to other agents federated with it instead of cAMP. The messages lead other agents to degenerate their functions specified in the messages and to decrease the superiority of these functions. As a result, agents complement others in the sense that each agent can provide some functions to other agents and delegate other functions to other agents that can also provide the functions.

Dedifferentiation: Agents may lose their functions due to differentiation as well as being busy or having failed. The approach also offers a mechanism for recovering from such problems based on dedifferentiation, which is a mechanism for regressing specialized cells to simpler, more embryonic, unspecialized forms. If there are no other agents that are sending restraining messages to an agent, the agent can perform its dedifferentiation process and strengthen their less-developed or inactive functions again.

4 Design and Implementation

The whole system consists of two parts: runtime systems and agents. The former is a middleware system for running at computers and the latter is a self-contained and autonomous entity and corresponds to a service and consists of more than one function, called the *behavior* part, and its state, called the *body* part, with information for differentiation, called the *attribute* part. The first part defines more than one application-specific behavior, which is a general/practical-purpose program defined in JavaBean-compatible Java objects. The second is responsible for maintaining program variables shared by its behaviors parts. When it receives a request message from an external system or other agents, it dispatches the message to the behavior part that can handle the message. The third maintains descriptive information with regard to the agent, including its own identifier in its two key-value databases for maintaining the weights of its own behaviors and for recording information on the behaviors that other agents can provide. Note that we never expect that the latter is complete. In fact, the mechanism can still work, even if it is not complete.

4.1 Differentiation

Each agent (k-th) assigns its own maximum to the total of the weights of all its behaviors. The agent has behaviors $b_1^k, \ldots, b_n^k$ and w_i^k is the weight of behavior b_i^k. The W_i^k is the maximum of the weight of behavior b_i^k. The maximum total of the weights of its behaviors in the k-th agent must be less than W^k. ($W^k \geq \sum_{i=1}^{n} w_i^k$), where $w_j^k - 1$ is 0 if w_j^k is 0. The W^k may depend on agents. Our mechanism consists of two phases. The first-step phase involves the progression of behaviors in four steps.

Step 1: When an agent (k-th agent) receives a request message from another agent, it selects the behavior (b_i^k) that can handle the message from its behavior part and dispatches the message to the selected behavior (Figure 2 (a)).
Step 2: It executes the behavior (b_i^k) and returns the result.
Step 3: It increases the weight w_i^k of the behavior.
Step 4: It multicasts a restraining message with the signature of the behavior, its identifier (k), and the behavior's weight (w_i^k) to other agents (Figure 2 (b)).

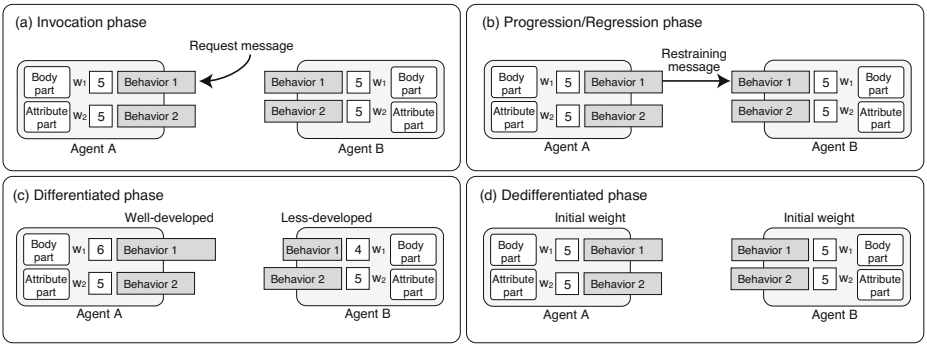


Fig. 2. Differentiation mechanism for agent

Note that when behaviors are internally invoked by their agents, their weights are not increased. The key idea behind this approach is to distinguish between internal and external requests. If the total weights of the agent's behaviors, $\sum w_i^k$, is equal to their maximal total weight W^k, it decreases one of the minimal (and positive) weights (w_j^k is replaced by $w_j^k - 1$ where $w_j^k = \mathbf{min}(w_1^k, \ldots, w_n^k)$ and $w_j^k \geq 0$). The above phase corresponds to the degeneration of agents. Restraining messages correspond to cAMP in differentiation. When the runtime system multicasts information about the signature of a behavior in restraining messages, the signature is encoded into a hash code by using Java's serial versioning mechanism and transmitted as code. The second-step phase supports the retrogression of behaviors in three steps.

Step 1: When an agent (k-th agent) receives a restraining message with regard to b_i^j from another agent (j-th), it looks for the behaviors ($b_m^k, \ldots b_l^k$) that can satisfy the signature specified in the received message.

Step 2: If it has such behaviors, it decreases their weights ($w_m^k, \ldots w_l^k$) in its first database and updates the weight (w_i^j) in its second database (Figure 2 (c)).

Step 3: If the weights ($w_m^k, \ldots, w_l^k$) are under a specified value, e.g., 0, the behaviors ($b_m^k, \ldots b_l^k$) are inactivated.

4.2 Service Matching

When an agent wants to execute a behavior, it needs to select one of the behaviors, even if it has the behavior, according to the values of their weights. This involves three steps.

Step 1: When an agent (k-th agent) wants to execute behavior b_i, it looks up the weight (w_i^k) of the same or compatible behavior from its first database and the weights ($w_i^j, \ldots, w_i^m$) of such behaviors ($b_i^j, \ldots, b_i^m$) from the second database.[1]

Step 2: If multiple agents, including itself, can provide the wanted behavior, it selects one of the agents according to selection function ϕ^k, which maps from w_i^k and $w_i^j, \ldots, w_i^m$ to b_i^l, where l is k or $j, \ldots, m$.

Step 3: It delegates the selected agent to execute the behavior and waits for the result from the agent.

[1] The agent (k-th) may have more than one same or compatible behavior.

The approach permits agents to use their own evaluation functions, ϕ, because the selection of behaviors often depends on their applications. For example, one of the simplest evaluation functions makes the agent that wants to execute a behavior select a behavior whose weight has the highest value and whose signature matches the wanted behavior if its first and second databases recognizes one or more agents that provide the same behavior, including itself. There is no universal selection function for mapping from behaviors' weights to at most one appropriate behavior like that in a variety of creatures. Therefore, the approach is open to defining functions by over-writing Java classes for a selection function.

4.3 Dedifferentiation

We need a mechanism for detecting failures in networking, remote computers, and other agents. To do this, each agent (j-th) periodically multicasts messages, called *heartbeat messages*, for behavior (b_i^j), which is still activated with its identifier (j). This involves two cases.

Case 1: When an agent (k-th) receives a heartbeat message with regard to behavior (b_i^j) from another agent (j-th), it keeps the weight (w_i^j) of the behavior (b_i^j) in its second database.

Case 2: When an agent (k-th) does not receive any heartbeat messages with regard to behavior (b_i^j) from another agent (j-th) for a specified time, it automatically decreases the weight (w_i^j) of the behavior (b_i^j) in its second database, and resets the weight (w_i^k) of the behavior (b_i^k) to the initial value or increases the weight (w_i^k) in its first database (Figure 2 (d)).

The weights of behaviors provided by other agents are automatically decreased without any heartbeat messages from the agents. Therefore, when an agent terminates or fails, other agents decrease the weights of the behaviors provided from the agent and if they have the same or compatible behaviors, they can then activate the behaviors, which may be inactivated. After a request message is sent to another agent, if the agent waits for the result to arrive for longer than a specified time, it selects one of the agents that can handle the message from its first and second databases and requests the selected agent. If there are no agents that can provide the behavior that can handle the behavior quickly, it promotes other agents that have the behavior in less-developed form (and itself if it has the behavior).

4.4 Current Status

Each runtime system is constructed as a middleware system that enables agents to duplicate themselves and migrate to other computers by using mobile agent technology [6]. It is responsible for executing agents and exchanging messages in runtime systems on other computers through a network. Restraining and heartbeat messages are multicasted as UDP packets, which may be unreliable. When a runtime system is (re)connected to a network, it multicasts heartbeat messages to other runtime systems to advertise itself, including its network address through UDP multicasts. Request and reply messages are implemented through TCP sessions.

Although the current implementation was not constructed for performance, we evaluated that of several basic operations in a distributed system where eight computers (Intel Core 2 Duo 1.83 GHz with MacOS X 10.6 and J2SE version 6) were connected through a Giga Ethernet. The cost of transmitting a heartbeat or restraining message through UDP multicasting was 11 ms. The cost of transmitting a request message between two computers was 22 ms through TCP.[2]

5 Evaluation

This experiment was aimed at evaluating the basic performance of adaptation, where one or more simple agents at computers and each agent issued heartbeat messages every 100 ms. Figure 3 shows the evaluation of differentiation. Each agent had three behaviors, called A, B, and C. The A behavior periodically issued messages to invoke its B and C behaviors or those of other agents every 200 ms and the B and C behaviors were null behaviors. Each agent that wanted to execute a behavior, i.e., B or C, selected a behavior whose weight had the highest value if its database recognized one or more agents that provided the same or compatible behavior, including itself. When it invoked behavior B or C and the weights of its and others behaviors were the same, it randomly selected one of the behaviors. The weights of the B and C behaviors of each agent in this experiment would initially be five and the maximum weight of each behavior and the total maximum of weights would be ten.

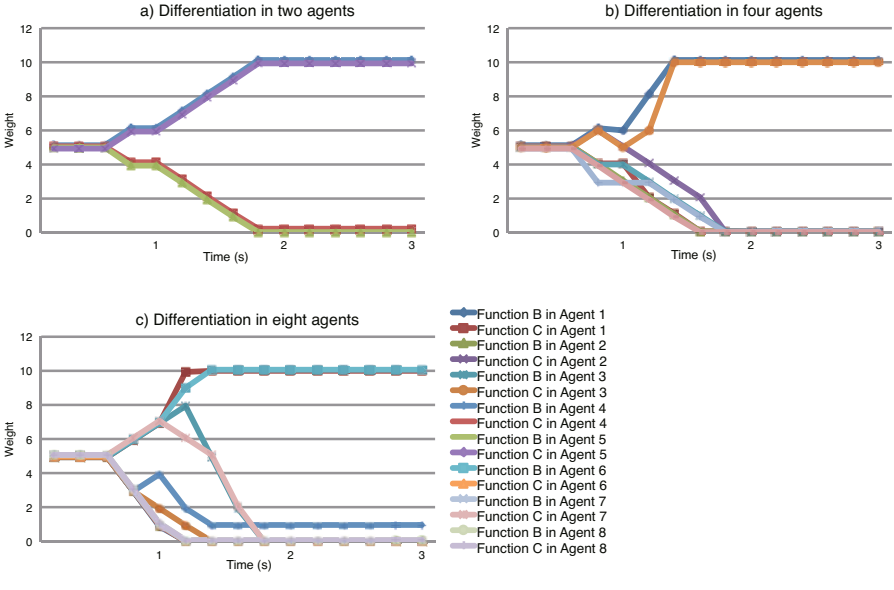


Fig. 3. Degree of progress in differentiation-based adaptation

[2] These costs were estimated from measurements of round-trip times between computers.

Figure 3 presents the results we obtained from the experiment. Both diagrams have a timeline on the x-axis and the weights of behavior B in each agent on the y-axis. Figure 3 (a) details the results obtained from our differentiation between two agents. Their weights were not initially varied and then they forked into progression and regression sides. Figure 3 (b) shows the detailed results of our differentiation between four agents and Figure 3 (c) shows those of that between eight agents. The results in (b) and (c) fluctuated more and then converged faster than those in (a) because the weights of behaviors in four agents were increased or decreased more than those in two agents. Although the time of differentiation depended on the period behaviors were invoked it was independent of the number of agents. This is important to prove that this approach is scalable.

Figure 4 shows the evaluation of dedifferentiation against failures in a distributed system. We assumed in the following experiment that three differentiated agents would be running on different computers and each agent had four behaviors, called A, B, C, and D, where the A behavior invoked other behaviors every 200 ms. The maximum for each behavior was ten and the agents' total maximum of weights was twenty. The initial weights of their behaviors (w_B^i, w_C^i, w_D^i) in the i-th agent were $(10, 0, 0)$ in the first, $(0, 10, 0)$ in the second, and $(0, 0, 10)$ in the third.

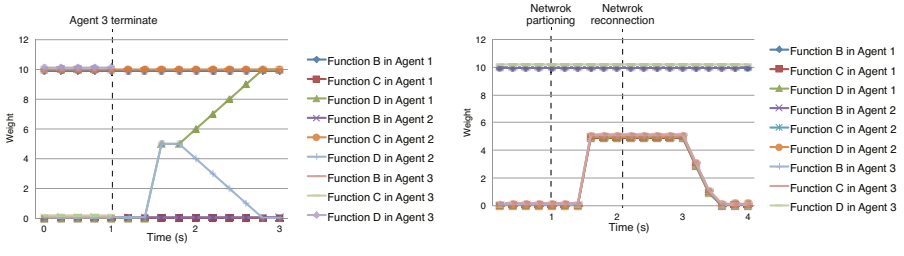


Fig. 4. Degree of progress in adaptation to failed agent

Figure 4 shows how remaining agents adapt to the termination of an agent. The third agent was terminated one second later. The remaining agents, i.e., the first and second agents, could not invoke the behavior, i.e., D, provided by it, and they dedifferentiated the behavior inside themselves.

6 Application

Here, we present a practical application with this approach to illustrate the utility of our (de)differentiation for service deployment and composition in a disaggregated computing setting. This application was inspired by the concept of *disaggregated computing*, which is an approach to dynamically composing devices, e.g., displays, keyboard, and mice that are not attached to the same computer, into a virtual computer among distributed computers in an ambient computing environment [3].

Our application is constructed as a single drawing service consisting of *model*, *view*, and *control* behaviors like a model-view-control (MVC) pattern. The first manages and stores drawing data and should be executed on a computer equipped with a powerful processor and a lot of memory. The second part displays drawing data on the screen

of its current host and should be deployed at computers equipped with large screens. The third part forwards drawing data from the pointing device, e.g., touch panel, of its current computer to the first behavior.

The service is initially deployed at a server, which lacks displays and storage. When the server is connected to a network, its runtime system discovers a computer equipped with a pointing device and a large display, e.g., a smart TV; the agent makes a clone of it with its behaviors and deploys the clone agent at the smart TV. The original agent, which is running on the server, decreases the weights of its behaviors corresponding to the view and control parts and the clone agent, which is running on the smart TV, decreases the weight of its behavior corresponding to the model part. This is because the server has no display or pointing device and the smart TV had no storage device. Therefore, each of the agents delegates the behaviors that its computer does not support to another agent according to the capabilities of their current computers. When a user disconnects the server from the network, the agent running on the server dedifferentiates itself, because it lacks co-partners, and it delegates the behaviors corresponding to the view and control parts.

7 Conclusion

This paper proposed a approach to adapting services on distributed systems, in particular ambient computing environments. It is unique to other existing software adaptations in introducing the notions of differentiation and dedifferentiation in cellular slime molds, e.g., dictyostelium discoideum, into software agents. When an agent delegates a behavior to another agent, if the former has the behavior, its behavior becomes less-developed and the latter's behavior becomes well-developed. The approach was constructed as a middleware system on real distributed systems instead of any simulation-based systems.

References

1. Blair, G.S., Coulson, G., Blair, L., Duran-Limon, H., Grace, P., Moreira, R., Parlavantzas, N.: Reflection, self-awareness and self-healing in OpenORB. In: Proceedings of 1st Workshop on Self-healing systems (WOSS 2002), pp. 9–14. ACM Press (2002)
2. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press (1999)
3. Brumitt, B.L., Meyers, B., Krumm, J., Kern, A., Shafer, S.: EasyLiving: Technologies for Intelligent Environments. In: Proceedings of International Symposium on Handheld and Ubiquitous Computing, pp. 12–27 (2000)
4. Georgiadis, I., Magee, J., Kramer, J.: Self-Organising Software Architectures for Distributed Systems. In: Proceedings of 1st Workshop on Self-healing systems (WOSS 2002), pp. 33–38. ACM Press (2002)
5. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
6. Satoh, I.: Handbook of Ambient Intelligence and Smart Environments, pp. 771–791. Springer, Heidelberg (2010)
7. Suda, T., Suzuki, J.: A Middleware Platform for a Biologically-inspired Network Architecture Supporting Autonomous and Adaptive Applications. IEEE Journal on Selected Areas in Communications 23(2), 249–260 (2005)

Graceful Interruption of Request-Response Service Interactions*

Mila Dalla Preda, Maurizio Gabbrielli, Ivan Lanese,
Jacopo Mauro, and Gianluigi Zavattaro

Lab. Focus, Department of Computer Science/INRIA, University of Bologna, Italy
{dallapre,gabbri,lanese,jmauro,zavattar}@cs.unibo.it

Abstract. Bi-directional request-response interaction is a standard communication pattern in Service Oriented Computing (SOC). Such a pattern should be interrupted in case of faults. In the literature, different approaches have been considered: WS-BPEL discards the response, while Jolie waits for it in order to allow the fault handler to appropriately close the conversation with the remote service. We investigate an intermediate approach in which it is not necessary for the fault handler to wait for the response, but it is still possible on response arrival to gracefully close the conversation with the remote service.

1 Introduction

Service-oriented computing (SOC) is a programming paradigm based on the composition of services, computational entities available on the net. According to WSDL [9], the standard for describing web service interfaces, services can be invoked according to two main modalities: one-way and request-response. In one-way communication a message is sent to a remote service. In request-response communication a message is sent and an answer is waited for before continuing the computation.

Interaction with remote services may incur in errors of different kinds: the remote service may disconnect, messages may be lost, or a client may interrupt the interaction with a remote service exactly in between the request and the corresponding response. To avoid that such an error causes the failure of the whole application, error handling techniques have been developed. They are commonly based on the concept of fault handler and compensation. A fault handler is a piece of code devoted to take the application to a consistent state after a fault has been caught. A compensation is a piece of code devoted to undoing the effect of a previous activity because of a later error.

As an example, consider a hotel reservation service. A reservation can be canceled, but if it is not annulled the cost of one night will be charged in case of no show. If the trip has to be annulled, the compensation for the hotel reservation has to be executed, thus canceling the reservation and avoiding the cost of a no show.

Jolie [3] is a language for programming service-oriented applications. Jolie request-response invocation establishes a strong connection between the caller and callee, thus it should not be disrupted by faults. To this end, callee faults are notified to the caller that can thus manage them. Symmetrically, in case of caller faults the answer from the

* Partly funded by the projects EU FP7-231620 HATS and ANR-2010-SEGI-013 AEOLUS.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 590–600, 2011.

callee is waited for and used during recovery. This allows, in particular, to compensate successful remote activities which are no more needed because of the local fault. This is the case of the hotel reservation above.

WS-BPEL [8], a main standard in the field, has a different approach: in case of caller faults execution can continue without waiting for the response, and the response is discarded upon arrival. In particular, it is not possible to write code that will be executed upon receipt of the response. The Jolie approach allows for programming safer applications. The fact that the request-response pattern is not disrupted by errors has been proved in [3], by relying on SOCK [4,2], a calculus defining the formal semantics of Jolie. A nasty side effect of the Jolie approach is that the client has to wait for answers of request-response invocations before proceeding in its execution. This slows down the caller execution. For instance, referring to the hotel reservation example, the client cannot continue its operations before the answer from the hotel has been received and (s)he gets stuck whenever the answer is lost. This drawback is unacceptable for programming applications over the net. Such a kind of problem is normally solved using timeouts, but they are not available in Jolie. Also, they are not easy to mimic.

We propose here a new approach to error handling in Jolie, allowing on one side to compensate undesired remote side effects, and ensuring on the other side that local computation is not slowed down in case of late answers. In particular, this new approach allows to easily program timeouts. We also extend the approach to deal with concurrent invocations of multiple services, as needed for implementing speculative parallelism.

2 SOCK

We first introduce SOCK [4], the calculus that defines the semantics of Jolie [3] programs, and then we extend it to account for request-response and multiple request-response service invocations. SOCK is suitable for illustrating our approach since it has a formal SOS semantics, it provides request-response as a native operator, and it has a refined approach to error handling.

In the following we present the three layers in which SOCK is structured, omitting the aspects that are not central for us (see [4] for a detailed description).

Service behavior layer. The service behavior layer describes the actions performed by services. Actions can be operations on the state or communications. Services are identified by the name of their operations, and by their location.

SOCK error handling is based on the concepts of *scope*, *fault*, and *compensation*. A scope is a process container denoted by a unique name. A fault is a signal raised by a process when an error state is reached. A compensation is used either to smoothly stop a running activity in case of an external fault, or to compensate the activity after its successful termination (this encompasses both WS-BPEL termination and compensation mechanisms). Recovering mechanisms are implemented by exploiting processes called *handlers*. We use *fault handlers* and *compensation handlers*. They are executed to manage respectively internal faults and external faults/compensation requests.

SOCK *syntax* is based on the following (disjoint) sets: Var, ranged over by x, y, for variables, Val, ranged over by v, for values, $\mathcal{O}$, ranged over by o, for one-way operations, $Faults$, ranged over by f, for faults, and $Scopes$, ranged over by q, for

Table 1. Service behavior syntax with faults

$P, Q ::= \overline{o}@l(\boldsymbol{y})$	output	$o(\boldsymbol{x})$	input
$x := e$	assignment	$P; Q$	sequence
$P \vert Q$	parallel comp.	$\sum_{i \in I} o_i(\boldsymbol{x}_i); P_i$	external choice
$if\ \chi\ then\ P\ else\ Q$	det. choice	$while\ \chi\ do\ (P)$	iteration
$\mathbf{0}$	null process	$\{P : \mathcal{H} : u\}_{q_\perp}$	active scope
$\mathrm{inst}(\mathcal{H})$	install handler	$\mathrm{throw}(f)$	throw
$\mathrm{comp}(q)$	compensate	$\langle P \rangle$	protection

scope names. Loc is a subset of Val containing locations, ranged over by l. We denote as SC the set of service behavior processes, ranged over by $P, Q, \ldots$. We use $q_\perp$ to range over $Scopes \cup \{\perp\}$, whereas u ranges over $Faults \cup Scopes \cup \{\perp\}$. Here $\perp$ is used to specify that a handler is undefined. $\mathcal{H}$ denotes a function from $Faults$ and $Scopes$ to processes (or $\perp$). The function associating P_i to u_i for $i \in \{1, \ldots, n\}$ is $[u_1 \mapsto P_1, \ldots, u_n \mapsto P_n]$. Finally, we use the notation $\boldsymbol{k} = \langle k_0, k_1, ..., k_i \rangle$ for vectors.

The syntax of service behavior processes is defined in Table 1. A one-way output $\overline{o}@l(\boldsymbol{y})$ invokes the operation o of a service at location l, where $\boldsymbol{y}$ are the variables that specify the values to be sent. Dually, in a one-way $o(\boldsymbol{x})$, $\boldsymbol{x}$ contains the variables that will receive the communicated values. Assignment, sequence, parallel composition, external and deterministic choice, iteration, and null process are standard.

We denote with $\{P\}_q$ a scope named q executing process P. An active scope has instead the form $\{P : \mathcal{H} : u\}_{q_\perp}$, where $\mathcal{H}$ specifies the defined handlers. Term $\{P\}_q$ is a shortcut for $\{P : \mathcal{H}_0 : \perp\}_q$, where $\mathcal{H}_0$ evaluates to $\perp$ for all fault names and to $\mathbf{0}$ for all scope names. The argument u is the name of a handler waiting to be executed, or $\perp$ if there is no such handler. When a scope has failed its execution, either because it has been killed from a parent scope, or because it has not been able to manage an internal fault, it reaches a zombie state. Zombie scopes have $\perp$ as scope name. Primitives $\mathrm{throw}(f)$ and $\mathrm{comp}(q)$ respectively raises fault f and asks to compensate scope q. $\langle P \rangle$ executes P in a protected way, i.e. not influenced by external faults. Handlers are installed into the nearest enclosing scope by $\mathrm{inst}(\mathcal{H})$, where $\mathcal{H}$ is the required update of the handler function. We assume that $\mathrm{comp}(q)$ occurs only within handlers, and q can only be a child of the enclosing scope. For each $\mathrm{inst}(\mathcal{H})$, $\mathcal{H}$ is defined only on fault names and on the name of the nearest enclosing scope. Finally, scope names are unique.

The service behavior layer *semantics* generates all the transitions allowed by the process behavior, specifying the constraints on the state that have to be satisfied for them to be performed. The state is a substitution of values for variables. We use σ to range over substitutions, and write $[\boldsymbol{v}/\boldsymbol{x}]$ for the substitution assigning values in $\boldsymbol{v}$ to variables in $\boldsymbol{x}$. Given a substitution σ, $\mathrm{Dom}(\sigma)$ is its domain.

Let Act be the set of labels of the semantics, ranged over by a. We use structured labels of the form $\iota(\sigma : \theta)$ where ι is the kind of action while σ and θ are substitutions containing respectively the assumptions and the effects on the state. We also use the unstructured labels $th(f), cm(q, P), inst(\mathcal{H})$. We use operator $\boxplus$ for updating the handler function:

$$(\mathcal{H} \boxplus \mathcal{H}')(u) = \begin{cases} \mathcal{H}'(u) \text{ if } u \in \mathrm{Dom}(\mathcal{H}') \\ \mathcal{H}(u) \text{ otherwise} \end{cases}$$

Table 2. Standard rules for service behavior layer ($a \neq th(f)$)

$$\text{(ONE-WAYOUT)}\quad \overline{o}@l(\boldsymbol{x}) \xrightarrow{\overline{o}(\boldsymbol{v})@l(\boldsymbol{v}/\boldsymbol{x}:\emptyset)} \mathbf{0} \qquad \text{(ONE-WAYIN)}\quad o(\boldsymbol{x}) \xrightarrow{o(\boldsymbol{v})(\emptyset:\boldsymbol{v}/\boldsymbol{x})} \mathbf{0}$$

$$\text{(ASSIGN)}\ \frac{\mathrm{Dom}(\sigma) = \mathrm{Var}(e) \quad [\![e\sigma]\!] = v}{x := e \xrightarrow{\tau(\sigma:v/x)} \mathbf{0}} \qquad \text{(IF-THEN)}\ \frac{\mathrm{Dom}(\sigma) = \mathrm{Var}(\chi) \quad [\![\chi\sigma]\!] = true}{if\ \chi\ then\ P\ else\ Q \xrightarrow{\tau(\sigma:\emptyset)} P}$$

$$\text{(SEQUENCE)}\ \frac{P \xrightarrow{a} P'}{P;Q \xrightarrow{a} P';Q} \qquad \text{(PARALLEL)}\ \frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad \text{(CHOICE)}\ \frac{o_i(\boldsymbol{x}_i) \xrightarrow{a} Q_i \quad i \in I}{\sum_{i\in I} o_i(\boldsymbol{x}_i); P_i \xrightarrow{a} Q_i; P_i}$$

STRUCTURAL CONGRUENCE

$$P \mid Q \equiv Q \mid P \qquad P \mid \mathbf{0} \equiv P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad \mathbf{0}; P \equiv P \qquad \langle \mathbf{0} \rangle \equiv \mathbf{0}$$

Intuitively, handlers in $\mathcal{H}'$ replace the corresponding ones in $\mathcal{H}$. We also use $\mathrm{cmp}(\mathcal{H})$ to denote the part of $\mathcal{H}$ dealing with compensations.

The SOCK semantics is defined as a relation $\rightarrow \subseteq SC \times Act \times SC$. The main rules for standard actions are in Table 2, while Table 3 defines the fault handling mechanism.

Rule ONE-WAYOUT defines the output operation, where $\boldsymbol{v}/\boldsymbol{x}$ is the assumption on the state. Rule ONE-WAYIN corresponds to the input operation: it makes no assumption on the state, but it specifies a state update. The other rules in Table 2 are standard. The internal process P of a scope can execute thanks to rule SCOPE in Table 3. Handlers are installed in the nearest enclosing scope by rules ASKINST and INSTALL. According to rule SCOPE-SUCCESS, when a scope successfully ends, its compensation handlers are propagated to the parent scope. Compensation execution is required by rule COMPENSATE. The actual compensation code Q is guessed, and the guess is checked by rule COMPENSATION. Faults are raised by rule THROW. A fault is caught by rule CATCHFAULT when a scope defining the corresponding handler is met. Activities involving the termination of a sub-scope and the termination of internal error recovery are managed by the rules for fault propagation THROW-SYNC, THROW-SEQ and RETHROW, and by the partial function $killable$. Function $killable$ computes the activities that have to be completed before the handler is executed and it is applied to parallel components by rule THROW-SYNC. Moreover, function $killable$ guarantees that when a fault is thrown there is no pending handler update. This is obtained by making $killable(P, f)$ undefined (and thus rule THROW-SYNC not applicable) if some handler installation is pending in P. The $\langle P \rangle$ operator (described by rule PROTECTION) guarantees that the enclosed activity will not be killed by external faults. Rule SCOPE-HANDLE-FAULT executes a handler for a fault. A scope that has been terminated from the outside is in zombie state. It can execute its compensation handler thanks to rule SCOPE-HANDLE-TERM, and then terminate with failure using rule SCOPE-FAIL. Similarly, a scope enters the zombie state when reached by a fault it cannot handle (rule RETHROW). The fault is propagated up along the scope hierarchy. Zombie scopes cannot throw faults any more, since rule IGNORE-FAULT has to be applied instead of RETHROW.

Table 3. Faults-related rules for service behavior layer ($a \neq th(f)$)

$$\text{(SCOPE)}\ \frac{P \xrightarrow{a} P' \quad a \neq inst(\mathcal{H}), cm(q', \mathcal{H}')}{\{P : \mathcal{H} : u\}_{q_\perp} \xrightarrow{a} \{P' : \mathcal{H} : u\}_{q_\perp}} \qquad \text{(INSTALL)}\ \frac{P \xrightarrow{inst(\mathcal{H})} P'}{\{P : \mathcal{H}' : u\}_{q_\perp} \xrightarrow{\tau(\emptyset:\emptyset)} \{P' : \mathcal{H}' \boxplus \mathcal{H} : u\}_{q_\perp}}$$

$$\text{(ASKINST)}\ \mathrm{inst}(\mathcal{H}) \xrightarrow{inst(\mathcal{H})} \mathbf{0} \qquad \text{(THROW)}\ \mathrm{throw}(f) \xrightarrow{th(f)} \mathbf{0} \qquad \text{(COMPENSATE)}\ \mathrm{comp}(q) \xrightarrow{cm(q,Q)} Q$$

$$\text{(SCOPE-SUCCESS)}\ \{\mathbf{0} : \mathcal{H} : \perp\}_q \xrightarrow{inst(\mathrm{cmp}(\mathcal{H}))} \mathbf{0} \qquad \text{(SCOPE-HANDLE-FAULT)}\ \{\mathbf{0} : \mathcal{H} : f\}_{q_\perp} \xrightarrow{\tau(\emptyset:\emptyset)} \{\mathcal{H}(f) : \mathcal{H} \boxplus [f \mapsto \perp] : \perp\}_{q_\perp}$$

$$\text{(COMPENSATION)}\ \frac{P \xrightarrow{cm(q,Q)} P', \mathcal{H}(q) = Q}{\{P : \mathcal{H} : u\}_{q'_\perp} \xrightarrow{\tau(\emptyset:\emptyset)} \{P' : \mathcal{H} \boxplus [q \mapsto \mathbf{0}] : u\}_{q'_\perp}}$$

$$\text{(SCOPE-HANDLE-TERM)}\ \{\mathbf{0} : \mathcal{H} : q\}_\perp \xrightarrow{\tau(\emptyset:\emptyset)} \{\mathcal{H}(q) : \mathcal{H} \boxplus [q \mapsto \mathbf{0}] : \perp\}_\perp \qquad \text{(SCOPE-FAIL)}\ \{\mathbf{0} : \mathcal{H} : \perp\}_\perp \xrightarrow{\tau(\emptyset:\emptyset)} \mathbf{0}$$

$$\text{(PROTECTION)}\ \frac{P \xrightarrow{a} P'}{\langle P \rangle \xrightarrow{a} \langle P' \rangle} \qquad \text{(THROW-SYNC)}\ \frac{P \xrightarrow{th(f)} P', killable(Q, f) = Q'}{P|Q \xrightarrow{th(f)} P'|Q'} \qquad \text{(THROW-SEQ)}\ \frac{P \xrightarrow{th(f)} P'}{P; Q \xrightarrow{th(f)} P'}$$

$$\text{(CATCH-FAULT)}\ \frac{P \xrightarrow{th(f)} P', \mathcal{H}(f) \neq \perp}{\{P : \mathcal{H} : u\}_{q_\perp} \xrightarrow{\tau(\emptyset:\emptyset)} \{P' : \mathcal{H} : f\}_{q_\perp}} \qquad \text{(IGNORE-FAULT)}\ \frac{P \xrightarrow{th(f)} P', \mathcal{H}(f) = \perp}{\{P : \mathcal{H} : u\}_\perp \xrightarrow{\tau(\emptyset:\emptyset)} \{P' : \mathcal{H} : u\}_\perp}$$

$$\text{(RETHROW)}\ \frac{P \xrightarrow{th(f)} P', \mathcal{H}(f) = \perp}{\{P : \mathcal{H} : u\}_q \xrightarrow{th(f)} \langle \{P' : \mathcal{H} : \perp\}_\perp \rangle}$$

where

$killable(\{P : \mathcal{H} : u\}_q, f) = \langle \{killable(P, f) : \mathcal{H} : q\}_\perp \rangle$ if $P \not\equiv \mathbf{0}$
$killable(P \mid Q, f) = killable(P, f) \mid killable(Q, f)$
$killable(P; Q, f) = killable(P, f)$ if $P \not\equiv \mathbf{0}$
$killable(\langle P \rangle, f) = \langle P \rangle$ if $killable(P, f)$ is defined
$killable(P, f) = \mathbf{0}$ if $P \in \{\mathbf{0}, o(\boldsymbol{x}), \overline{o}@l(\boldsymbol{x}), x := e, if\ \chi\ then\ P\ else\ Q, while\ \chi\ do\ (P)$
$\sum_{i \in W} o_i(\boldsymbol{x}_i); P_i, \mathrm{throw}(f), \mathrm{comp}(q)\}$

Service engine layer. The service engine layer manages the service state and instances. A service engine Y can be a session $(P, \mathcal{S})$, where P is a service behavior process and $\mathcal{S}$ is a state, or a parallel composition $Y|Y$ of them. The service engine layer allows to propagate only labels such that the condition σ (if available) is satisfied by the current state, and applies to the state the state update ρ.

Services system layer. The service system layer allows the interaction between different engines. A service system E can be a located service engine $Y@l$ or a parallel composition $E \parallel E$ of them. The services system layer just allows complementary communication actions to interact, transforming them into internal steps τ, and propagates the other actions.

3 Request-Response Interaction Pattern

A request-response pattern is a bi-directional interaction where a client sends a message to a server and waits for an answer. When a server receives such a message, it elaborates the answer and sends it back to the client. In the literature there are two proposals to deal with a client that fails during a request-response interaction. The WS-BPEL approach kills the receive activity and, when the message arrives, it is silently discarded. In Jolie instead, clients always wait for the answer and exploit it for error recovery.

Here we present an intermediate approach: in case of failure we wait for the answer, but without blocking the computation. Moreover, when the answer is received we allow for the execution of a compensation activity. Let $\mathcal{O}_r$ be the set of request-response operations, ranged over by o_r. We define the request-response pattern in terms of the output primitive $\overline{o_r}@l(\boldsymbol{y}, \boldsymbol{x}, P)$, also called solicit, and of the input primitive $o_r(\boldsymbol{x}_1, \boldsymbol{y}_1, Q)$. When interacting, the client sends the values from variables $\boldsymbol{y}$ to the server, that stores them in variables $\boldsymbol{x}_1$. Then, the server executes process Q and, when Q terminates, the values in variables $\boldsymbol{y}_1$ are sent back to the client who stores them in variables $\boldsymbol{x}$. Only at this point the execution of the client can restart. If a fault occurs on the client-side after the remote service has been invoked, but before the answer is received, we allow the client to handle the fault regardless of the reply, so that recovery can start immediately. However, we create a receiver for the missing message in a fresh session so that, if later on the message is received, the operation can be compensated. The compensation is specified by the parameter P of the solicit operation. If instead a fault is raised on the server-side during the computation of the answer, the fault is propagated to the client where it raises a local fault. In this case there is no need to compensate the remote invocation, since we assume that this is dealt with by local recovery of the server.

Service behavior calculus - extension. We extend here the behavioral layer with the request-response and with few auxiliary operators used to define its semantics.

$\overline{o_r}@l(\boldsymbol{y}, \boldsymbol{x}, P)$	Solicit	$o_r(\boldsymbol{x}_1, \boldsymbol{y}_1, Q)$	Request-Response
$Exec(l, o_r, \boldsymbol{y}, P)$	Req.-Resp. execution	$Wait(o_r, \boldsymbol{y}, P)$	Wait
$\overline{o_r}!f@l$	Fault output	$Bubble(P)$	Bubble

$Exec(l, o_r, \boldsymbol{y}, P)$ is a server-side running request-response: P is the process computing the answer, o_r the name of the operation, $\boldsymbol{y}$ the vector of variables to be used for the answer, and l the client location. Symmetrically, $Wait(o_r, \boldsymbol{y}, P)$ is the process waiting for the response on client-side: o_r is request-response operation, $\boldsymbol{y}$ is the vector of variables for storing the answer and P is the compensation code to run in case the client fails before the answer is received. When a fault is triggered on the server-side, an error notification has to be sent to the client: this is done by $\overline{o_r}!f@l$, where o_r is the operation, f the fault and l the client location. If a fault occurs on client-side, we have to move the receipt operation to a fresh, parallel session, so that error recovery can start immediately. This is done by the primitive $Bubble(P)$, which allows to create a new session (a "bubble") executing code P. This primitive is the key element that allows a failed solicit to wait for a response outside its scope and potentially allowing its termination regardless of the arrival of the answer.

The semantics of the behavior layer is extended with the rules presented in Table 4 (the last rule refers to the engine). Function $killable$ is also extended, as follows:

Table 4. Request-response pattern and engine rules

(SOLICIT)
$$\overline{o_r}@l(\boldsymbol{y},\boldsymbol{x},P) \xrightarrow{\overline{o_r}(\boldsymbol{v})@l(\emptyset:\boldsymbol{v}/\boldsymbol{x})} Wait(o_r,\boldsymbol{x},P)$$

(REQUEST)
$$o_r(\boldsymbol{x},\boldsymbol{y},P) \xrightarrow{o_r(\boldsymbol{v})::l(\emptyset:\boldsymbol{v}/\boldsymbol{x})} Exec(l,o_r,\boldsymbol{y},P)$$

(REQUEST-EXEC)
$$\frac{P \xrightarrow{a} P'}{Exec(l,o_r,\boldsymbol{y},P) \xrightarrow{a} Exec(l,o_r,\boldsymbol{y},P')}$$

(THROW-REXEC)
$$\frac{P \xrightarrow{th(f)} P'}{Exec(l,o_r,\boldsymbol{y},P) \xrightarrow{th(f)} P' \mid \langle \overline{o_r}!f@l \rangle}$$

(REQUEST-RESPONSE)
$$Exec(l,o_r,\boldsymbol{y},\mathbf{0}) \xrightarrow{\overline{o_r}(\boldsymbol{v})@l(\boldsymbol{v}/\boldsymbol{y}:\emptyset)} \mathbf{0}$$

(SOLICIT-RESPONSE)
$$Wait(o_r,\boldsymbol{x},P) \xrightarrow{o_r(\boldsymbol{v})(\emptyset:\boldsymbol{v}/\boldsymbol{x})} \mathbf{0}$$

(SEND-FAULT)
$$\overline{o_r}!f@l \xrightarrow{\overline{o_r}(f)@l(\emptyset:\emptyset)} \mathbf{0}$$

(RECEIVE FAULT)
$$Wait(o_r,\boldsymbol{x},P) \xrightarrow{o_r(f)(\emptyset:\emptyset)} \mathrm{throw}(f)$$

(CREATE BUBBLE)
$$Bubble(P) \xrightarrow{\tau(\emptyset:\emptyset)[\![P]\!]} \mathbf{0}$$

(ENGINE-BUBBLE)
$$\frac{P \xrightarrow{\tau(\emptyset:\emptyset)[\![Q]\!]} P' \quad Q \neq \mathbf{0}}{(P,\mathcal{S}) \xrightarrow{\tau} (P',\mathcal{S}) \mid (Q,\mathcal{S})}$$

- $killable(Exec(l,o_r,\boldsymbol{y},P),f) = killable(P,f) | \langle o_r!f@l \rangle$
- $killable(\,Wait(o_r,\boldsymbol{x},P),f) = Bubble(\,Wait(o_r,\boldsymbol{x},\mathbf{0});P)$
- $killable(\overline{o_r}!f@l,f) = \overline{o_r}!f@l$
- $killable(Bubble(P),f) = Bubble(P)$

Rules SOLICIT and REQUEST start a solicit-response operation on client and server side respectively. Upon invocation, the request-response becomes an active construct executing process P, and storing all the information needed to send back the answer. The execution of P is managed by rule REQUEST-EXEC. When P terminates, rule REQUEST-RESPONSE sends back an answer. This synchronizes with rule SOLICIT-RESPONSE on the client side.

A running request-response reached by a fault is transformed into a fault notification (see rule THROW-REXEC and the definition of function $killable$) on server side. Fault notification is executed by rule SEND-FAULT, and it interacts with the waiting receive thanks to rule RECEIVE-FAULT. When received, the fault is re-thrown at the client side.

A fault on client side instead gives rise to a bubble, creating the process that will wait for the answer in a separate session. The bubble is created by rule CREATE BUBBLE, and will be installed at the service engine level by rule ENGINE-BUBBLE. The label for bubble creation has the form $\tau(\emptyset:\emptyset)[\![P]\!]$, where P is the process to be run inside the new session. We will write $\tau(\emptyset:\emptyset)$ for $\tau(\emptyset:\emptyset)[\![\mathbf{0}]\!]$. The new receive operation inside the bubble has no handler update, since it will be executed out of any scope, and its compensating code P has been promoted as a continuation. In this way, P will be executed only in case of successful answer. In case of faulty answer, the generated fault will have no effect since it is in a session on its own.

Service engine calculus - extension. We have to add to the service engine layer a rule for installing bubbles: when a bubble reaches the service engine layer, a new session is started executing the code inside the bubble (rule ENGINE-BUBBLE in Table 4).

Service system calculus - extension. The service system calculus semantics is extended by allowing the labels for request-response communication to be matched.

Example. We present now an example of usage of the request-response primitive. A first solution for the hotel reservation example described in the introduction is:

```
CLIENT :== book_r@hotel_Imperial(⟨CC,dates⟩,⟨res_num⟩,
             annul@hotel_Imperial(⟨res_num⟩) );
           P
```

The $book_r$ operation transmits the credit card number CC and the dates of the reservation and waits for the reservation number. In case the user wants to cancel the reservation before receiving an answer from the hotel a fault can be used to kill this operation. In such a case the annul operation is invoked when the answer is received to compensate the $book_r$ operation. The annul operation will be executed in a new session by using our mechanism based on bubbles.

As a more concrete instance, we could consider the case where the user is willing to wait a limited amount of time for the answer from the hotel, after which (s)he will cancel the reservation. This case could be programmed by assuming a service $timeout$ that offers a request-response operation that sends back an answer after n seconds[1]:

```
CLIENT:==
    res_num:= 0;
    { inst(f ↦ if res_num==0 then throw(tm));
      (timeout_r@timeout(⟨60⟩,⟨ ⟩,0); throw(f)
      | book_r@hotel_Imperial(⟨CC,dates⟩,⟨res_num⟩,
          annul@hotel_Imperial(⟨res_num⟩) ); throw(f)   )
    }_q ; P
```

In this scenario the timeout operation is in parallel with the booking. The first operation that finishes raises the fault f that is caught by the handler of the scope q. The fault will kill the remaining operation and if the hotel response has not arrived yet (i.e. the value of res_num is still 0) then the fault tm is raised. P is executed otherwise.

A similar solution is not viable in BPEL: in case of timeout, the booking invocation is killed, and if an answer arrives, it is discarded. Thus one does not know whether the invocation succeeded or not, neither the reservation number in case of success.

In Jolie, the answer is used for error recovery. However, in case no answer is received from the booking service, the whole service engine gets stuck. In our approach instead the main session can continue its execution without delays.

It is difficult to apply the proposed solution when two or more solicits install handlers or require compensation. One may try to exploit the handler update primitive, but in this way compensations are executed inside the scope, thus they have to be terminated before execution can proceed. This problem, and other technical difficulties, justify the multiple solicit response primitive introduced in the next section.

4 Multiple Request-Response Communication Pattern

The request-response pattern allows one invocation to be sent and one answer to be received. For optimization reasons, it may be important to invoke many services in parallel, and only consider the first received answer (speculative parallelism).

[1] Clearly, because of network delay the answer may be received later than expected.

Table 5. Multiple request-response pattern rules

(MSR-SOLICIT)
$$\frac{z_1 = \overline{o_r}@l(\boldsymbol{y}, \boldsymbol{x}, P) \mapsto Q \quad w_{m+1} = \mathit{Wait}(o_r, \boldsymbol{y}, P) \mapsto Q}{\mathit{Wait}^+(z_1, \ldots, z_n \triangleright w_1, \ldots, w_m) \xrightarrow{\overline{o_r}(\boldsymbol{v})@l(\emptyset:\boldsymbol{v}/\boldsymbol{x})} \mathit{Wait}^+(z_2, \ldots, z_n \triangleright w_1, \ldots, w_m, w_{m+1})}$$

(MSR-RESPONSE)
$$\frac{\forall k \in \{1, \ldots, n\} : w_k = \mathit{Wait}(o_{r_k}, \boldsymbol{y}_k, P_k) \mapsto Q_k \;\; i \in \{1, \ldots, n\} \;\; J = \{1, \ldots, n\} \setminus \{i\}}{\mathit{Wait}^+(\triangleright w_1, \ldots, w_n) \xrightarrow{o_{r_i}(\boldsymbol{v})(\emptyset:\boldsymbol{v}/\boldsymbol{y_i})} Q_i | \prod_{j \in J} \mathit{Bubble}(\mathit{Wait}(o_{r_j}, \boldsymbol{y_j}, \boldsymbol{0}); P_j)}$$

(MSR-IGNORE FAULT)
$$\frac{n > 1 \;\; w_i = \mathit{Wait}(o_{r_i}, \boldsymbol{y_i}, P_i) \mapsto Q_i \;\; i \in \{1, \ldots, n\}}{\mathit{Wait}^+(\triangleright w_1, \ldots, w_n) \xrightarrow{o_{r_i}(f)(\emptyset:\emptyset)} \mathit{Wait}^+(\triangleright w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n)}$$

$$\mathit{Wait}^+(\triangleright \mathit{Wait}(o_r, \boldsymbol{y}, P) \mapsto Q) \equiv \mathit{Wait}(o_r, \boldsymbol{y}, P); Q$$

We model this communication pattern using a dedicated primitive that we call multiple solicit-response (MSR for short). A MSR consists of a list of solicit-responses, each one equipped with its own continuation. Formally, we define the syntax of the MSR primitive as $\mathit{MSR}\{z_1, \ldots, z_n\}$ where each z_i is a *solicit-response with continuation* written $z_i = \overline{o_{r_i}}@l_i(\boldsymbol{y}_i, \boldsymbol{x}_i, P_i) \mapsto Q_i$. Intuitively, the continuation Q_i is executed only when $\overline{o_{r_i}}@l_i(\boldsymbol{y}_i, \boldsymbol{x}_i, P_i)$ is the first to receive a successful answer.

Service behavior calculus - extension. We extend the service behavior calculus with the MSR primitive and with some auxiliary operators:

$P, Q ::= \ldots$		
	$\mathit{MSR}\{z_1, \ldots, z_n\}$	multiple solicit-response
	$\mathit{Wait}^+(z_1, \ldots, z_n \triangleright w_1, \ldots, w_m)$	multiple wait
$z ::= \overline{o_r}@l(\boldsymbol{y}, \boldsymbol{x}, P) \mapsto Q$		solicit with continuation
$w ::= \mathit{Wait}(o_r, \boldsymbol{y}, P) \mapsto Q$		wait with continuation

In a MSR the solicits are sent one after the other, and only when all the requests have been sent the MSR can receive a response. For this reason we introduce the multiple wait $\mathit{Wait}^+(z_1, \ldots, z_n \triangleright w_1, \ldots, w_m)$ that specifies the solicits that still have to be sent $z_1, \ldots, z_n$, and the ones that will wait for an answer $w_1, \ldots, w_n$. Thus, the MSR primitive $\mathit{MSR}\{z_1, \ldots, z_n\}$ above is a shortcut for $\mathit{Wait}^+(z_1, \ldots, z_n \triangleright)$. Moreover, we have that a multiple wait with only one waiting process is structurally equivalent to a standard wait. We formally define the behavior of the MSR primitive by extending the service behavior semantics with the rules presented in Table 5.

The multiple wait executes all the solicit-responses through rule MSR-SOLICIT. Once all the solicits have been sent, the multiple wait receives a successful answer through rule MSR-RESPONSE. It continues the execution with the corresponding continuation code, and kills all the other solicits by creating a bubble for each remaining waiting process. If a fault notification arrives as an answer, it is discarded by rule MSR-IGNORE FAULT if there is at least another available wait. If instead there is no other solicit waiting for an answer, the last fault received is raised (rule RECEIVE FAULT

described in Table 4). When an external fault arrives a bubble containing a dead solicit response is created for every solicit that has been sent, as specified by the function $killable$ that is extended in the following way:

$$killable(Wait^{+}(z_1,\ldots,z_n \triangleright w_1,\ldots,w_m), f) = \prod_{Wait(o_{r_j},\boldsymbol{y_j},P_j)\mapsto Q_j \in \{w_1,\ldots,w_m\}} Bubble(Wait(o_{r_j},\boldsymbol{y_j},\mathbf{0});P_j)$$

The MSR primitive is perfectly suited to capture speculative parallelism scenarios. Consider for instance the hotel reservation problem defined in the introduction. Suppose to use two booking services for making the hotel reservation, and that you would like to get the acknowledgment in 1 minute. If the booking services are located at A and B and if we use the $timeout$ service introduced before, this service could be defined as:

```
CLIENT :== msr {
    timeout_r@timeout(⟨60⟩,⟨ ⟩,0) ↦ throw(tm)
    book_r@H_1(⟨CC,dates⟩,⟨res_num⟩, annul@H_1(⟨res_num⟩) ) ↦ 0
    book_r@H_2(⟨CC,dates⟩,⟨res_num⟩, annul@H_2(⟨res_num⟩) ) ↦ 0 }
```

5 Related and Future Work

Among the most related approaches, Web-π [6] has no request-response pattern and its treatment of faults is rather different from ours. Orc [5] has a pruning primitive similar to our MSR. However, since Orc has no notion of fault, all the difficulties coming from error management do not emerge. Finally, the service oriented calculi CaSPiS [1] and COWS [7] include low-level mechanisms allowing the programmer to close ongoing conversations. However, our approach is different, since we aim at providing primitives which free the programmer from this burden.

As a future work, we plan to incorporate the primitives we propose in Jolie. Also we would like to study their expressive power: the implementation of MSR in terms of the existing primitives is not easy and we believe that a separation result could be proved.

References

1. Boreale, M., Bruni, R., De Nicola, R., Loreti, M.: Sessions and Pipelines for Structured Service Programming. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 19–38. Springer, Heidelberg (2008)
2. Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: On the interplay between fault handling and request-response service invocations. In: ACSD 2008, pp. 190–198. IEEE Press (2008)
3. Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: Dynamic error handling in service oriented applications. Fundamentae Informaticae 95(1), 73–102 (2009)
4. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: SOCK: A Calculus for Service Oriented Computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
5. Kitchin, D., Quark, A., Cook, W., Misra, J.: The Orc Programming Language. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS 2009. LNCS, vol. 5522, pp. 1–25. Springer, Heidelberg (2009)

6. Laneve, C., Zavattaro, G.: Foundations of web transactions. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 282–298. Springer, Heidelberg (2005)
7. Lapadula, A., Pugliese, R., Tiezzi, F.: A Calculus for Orchestration of Web Services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 33–47. Springer, Heidelberg (2007)
8. OASIS. Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
9. World Wide Web Consortium. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

Adaptation of Web Service Interactions Using Complex Event Processing Patterns

Yéhia Taher, Michael Parkin, Mike P. Papazoglou, and Willem-Jan van den Heuvel

European Research Institute for Service Science, Tilburg University, The Netherlands
{y.taher,m.s.parkin,mikep,wjheuvel}@uvt.nl

Abstract. Differences in Web Service interfaces can be classified as signature or protocol incompatibilities, and techniques exist to resolve one or the other of these issues but rarely both. This paper describes *complex event processing* approach to resolving both signature and protocol incompatibilities existing between Web Service interfaces. The solution uses a small set of operators that can be applied to incoming messages individually or in combination to modify the structure, type and number of messages sent to the destination. The paper describes how CEP-based adapters, deployable in CEP engines, can be generated from automata representations of the operators through a standard process and presents a proof-of-concept implementation.

1 Introduction

Web services allow the integration of distributed software through standard interface definition languages, transport mechanisms and aspects such as security and quality of service. Web Service interfaces (i.e., WSDL, BPEL, etc.) define the messages and protocol that should be used to communicate with the service [7]. However, if two services wish to interact successfully, they must both support the same messages and protocol through the implementation of compatible WSDL and/or BPEL documents. Unfortunately, this is difficult to achieve in practice; Web Services are often developed independently and follow different standards or approaches in constructing their interfaces and Web Service compositions will often use them in ways that were not foreseen in their original design and construction [3, 2]. Therefore, it is likely that most Web Services will be incompatible as services will not support the same interface.

This is a short paper describing a general approach to resolving differences between Web Services protocols through the use of Complex Event Processing (CEP [4]) technology. Specifically, we extend our previous work [10, 11] to show how a small set of general operators can be used to match the messages from one service with those of another. By using a *continuous query engine* running within a CEP platform, we demonstrate signature and protocol adaptation between Web Services in a proof-of-concept implementation.

This paper is structured as follows: Section 2 describes our CEP-based approach to signature and protocol adaptation; Section 3 introduces the operators used to resolve differences in Web Service protocols; Section 4 presents the

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 601–609, 2011.

CEP solution and a proof-of-concept implementation; Section 5 compares related work; Section 6 contains conclusions and our plans for future work.

2 Approach

Incompatibilities between Web Service protocols can be classified as either [2,3]:

1. Signature Incompatibilities arise due to the differences between services in expected message structure, content and semantics. In Web Services, XML schema provides defines a set of 'built-in' types to allow the construction of complex input and output message types from these primitives. This flexibility in constructing message types in XML often means that a message from one Web Service will not be recognized by another and, therefore, there is a requirement to provide some function that maps the schema of one message to another [6].

2. Protocol Incompatibilities are found when Web Services wish to interact but are incompatible because they support of different message exchange sequences. For example, if two services perform the same function, e.g., accept purchase orders, but Service A requires a single order containing one or more items while Service B expects an order message for each item, there is a mismatch in their communication protocols that must be resolved in order for them to interoperate. To solve these incompatibilities, there are two approaches: a) to force one of the parties to support the other's interface, or b) to build an adapter that receives messages, converts them to the correct sequence and/or maps them into a desired format and sends them to their destination. However, both of these solutions are unsatisfactory; imposing the development of an interface for each target service can lead to an organization having to maintain a different client for each service it uses, and the implementation of bespoke ad-hoc point-to-point adapters is costly and not-scalable.

Our solution is to automate the generation of adapters so the process is repeatable and scalable and remove the necessity to build costly bespoke adapters. Our approach for generating adapters is described in [9], which presents an algorithm for detecting signature and protocol incompatibilities between two Web Service protocol descriptions (i.e., interfaces) and a CEP-based mediation framework to perform protocol adaptation practically. This paper completes the mediation framework by showing how the incompatibilities found between two Web Service protocols, classified according to a set of basic transformation patterns by the algorithm in [9], can be transformed into configurable automata *operators* which are used to generate *adapters*.[1] In Section 3 we describe the operators required for each transformation pattern then in Section 4 show how they are used to generate CEP adapters and deployed to a CEP engine.

Complex Event Processing technology can discover relationships between events through the analysis and correlation of multiple events and triggers and

[1] Adapters are therefore the components that resolve sets of incompatibilities found between two services and are aggregations of predefined *operators* who's purpose is to resolve individual, specific incompatibilities.

take actions (e.g., generate new events) from these observations. CEP does this by, for example, modeling event hierarchies, detecting causality, membership and/or timing relationships between events and abstracting event-driven processes into higher-level concepts [4]. CEP platforms allow streams of data to run through them to detect conditions that match the *continuous computational queries* (CCQs, written in a Continuous Computation language, or CCL) as they occur. As a result, CEP has an advantage in performance and capacity compared to traditional approaches: CEP platforms typically handle many more events than databases and can process throughputs of between 1,000 to 100k messages per second with low latency. These features make a CEP platform an excellent foundation for situations that have real-time business implications.

In the context of Web Services, *events* occur when SOAP messages are sent and received. Therefore, CEP adaptation requires the platform to consume incoming messages, process them and send the result to its destination. However, a CCQ written for a particular adaptation problem is similar to the bespoke adapter solution described earlier. To offer a universal solution and a scalable method for Web Service protocol adaptation, we automate the generation and deployment of CCQs to transform incoming message(s) into the required output message format(s) using the predefined set of transformation operators.

3 Operators

[3] describes five basic transformation patterns that can reconcile protocol mismatches. We have developed an operator for each of these patterns that can be applied individually or in combination to incoming messages to achieve a transformation in both the structure, type and number of messages sent to the destination — i.e., to resolve both signature and protocol incompatibilities.

The operators developed for each of the transformation patterns are: *Matchmake*, which translates one message type to another, solving the *one-to-one* transformation; *Split*, a solution for the *one-to-many* pattern, which separates one message sent by the source into two or more messages to be received separately; the *Merge* operator is the opposite of the *Split* operator (i.e., it performs a *many-to-one* transformation) and combines two or more messages into a single message; the *Aggregate* operator is used when two or more of the same message from the source service interface correspond to one message at the target service and is a solution for the *one*$^{+}$*-to-one* transformation; finally, *Disaggregate* performs the opposite function to *Aggregate* operator.

Following [9], the operators are represented as *configurable automata*. Transitions between states represent both observable and non-observable actions. *Observable actions* describe the behavior of the operator vis-à-vis the service consumer and provider, i.e., an action is observable if it is a message consumption or transmission event. *Unobservable actions* describe the internal transitions of the operator, such as the transformation of a messages contents, and are performed transparently to the source and target services.

Transitions caused by observable actions are denoted as $<a,?/!m,a'>$, where a is the starting state and a' the end state following the consumption *(?)* or

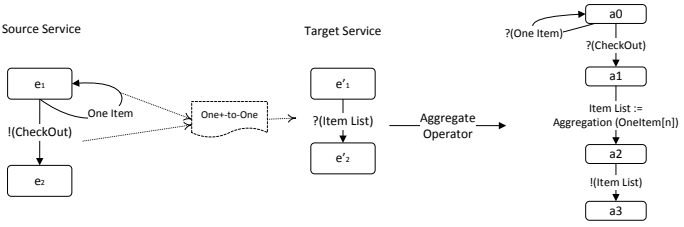


Fig. 1. The Aggregate Operator

transmission *(!)* of message *m*. An unobservable action is denoted as $<a, \psi, a'>$, where *a* and *a'* are the start and end states following internal action ψ.

For reasons of space it is not possible to describe all five operators in detail and we have chosen the *Aggregate* operator to illustrate how they work. Figure 1 shows the operator to resolve *one*$^+$*-to-one* incompatibilities between services, e.g., when a customer submits a purchase order for each item but the retailer expects a list of all items together. To resolve this incompatibility the aggregate operator consumes and stores *?OneItem* messages until it receives the message *(?CheckOut)* indicating all messages have been sent. The operator aggregates the stored messages into a list of items message using *ItemList=Aggregation(OneItem[n])* and forwards the new message using *!(ItemList)*.

4 CEP-Based Adaptation

4.1 General Principles

The adaptation of interactions between source and target services is specified using automata, therefore deploying them as CEP adapters requires their translation into continuous queries. To do this, we modeled message consumption and transmission actions as events. For each message type consumed or transmitted we create an input or output stream. A continuous query subscribes to the input stream of messages it wants to adapt and publishes the adapted message(s) to the corresponding output stream(s). For convenience, we name the input/output stream the same name as the message it consumes or transmits. This method allows a CEP engine to intercept messages exchanged between two services, to detect patterns of incompatibilities and implement corresponding adaptation solution, i.e., combinations of the operators encoded as continuous queries.

4.2 Conceptual Architecture

Figure 2 illustrates the conceptual architecture of the CEP implementation that translates the adapter specified in automata into continuous queries, i.e., via `Automata` → `Continuous Queries`. This includes the creation of input and output streams for the continuously running queries in the CEP engine, waiting for messages arriving through input streams.

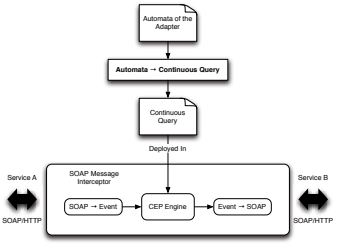


Fig. 2. Conceptual CEP Adaptation Architecture

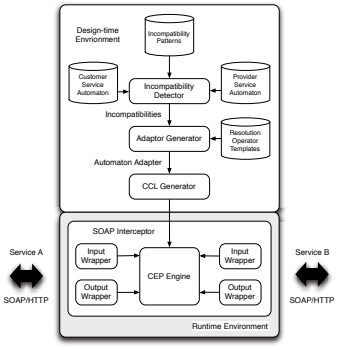


Fig. 3. Architectural Framework for the CEP Solution

In the second step, the `SOAP Message Interceptor`'s role is to control the exchange of messages between the two services. Upon receipt of a message, the Interceptor sends it to the input stream with the same name (through `SOAP` → `Event`). The message received is published as an event and is consumed by the query that subscribes to the input stream. The message(s) produced as a result of applying the operators is published to the corresponding output stream. Once on the output stream, the message is consumed by the SOAP message interceptor (through `Event` → `SOAP`) and sent to the target.

Figure 4 shows the transformation of the *Aggregation* automata to a continuous query. First, an input stream and a window to store messages arriving on the input stream are created for action *?(m1)* and an input stream is created for action *?(m2)*. The aggregation query is then specified: it subscribes to the window where messages from action *?(m1)* are stored and to the input stream for action *!(m2)* actions. After an *!(m2)* action, messages in the window with the same correlation criteria as new message are aggregated into a single message, the input messages are removed and the result is published to the output stream.

Figure 5 shows a concrete example where messages arriving through the input stream *Order_In* are stored in the window *Order_Win*. When the message arrives through *CheckOut_In* indicating order number #03203 is complete, messages in *Order_Win* with the same order number (#03203, the correlation criteria) are aggregated into a single message. The final message, containing *Item1* and *Item2*, is published to *Order_Out*.

4.3 Proof of Concept

This section illustrates the practical generation and deployment of CEP-based adapters using *model transformation*. It has two stages: the *design phase* models the adapter using operator automata through the use of an incompatibility detection process to produce a platform independent model, whilst the *transformation phase* takes the platform independent model to produce the adapter as a CCQ for a CEP engine, i.e, a platform specific model.

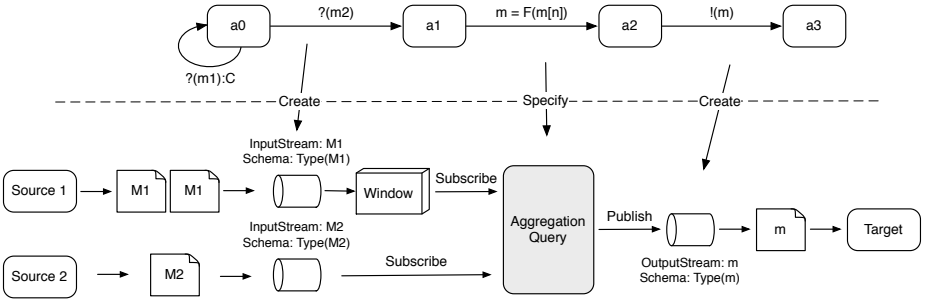


Fig. 4. Aggregate Translation

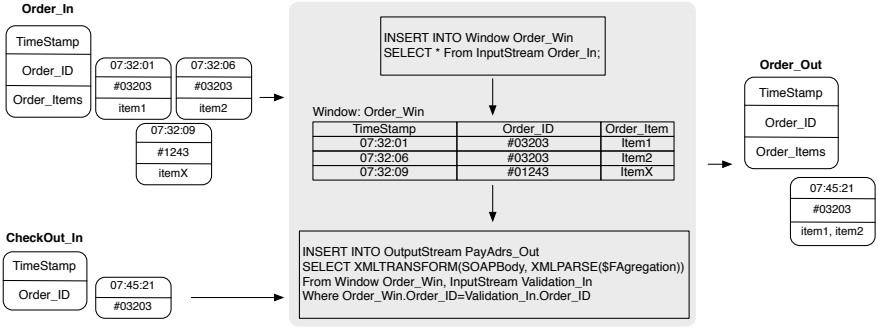


Fig. 5. Aggregate Flow

Figure 3 shows the framework for the automatic generation of adapters. If two incompatible services, A and B, wish to communicate, at design-time an adaptor can be generated for the runtime CEP engine by classifying the incompatibilities between two service interfaces (using the method described in [9]) and using them to construct the adapter using *resolution operator templates* (described in Section 3). The resulting adapter is converted into the CEP engine's continuous computation language (CCL) that is deployed at run-time within a SOAP message interceptor to provide a message serialization/deserialization capability.

The **Design Time Environment** is used to instantiate the template operators described in Section 3 so they can be used in a specific Web Service protocol adaptation. The design-time tools can be used to develop strategies for dealing with complex adaptation situations by allowing the composition of the template operators. In these cases, the designers of the adaptation must identify what adaptations are required between two services and use a graphical user interface (the *Design Tool* shown) to wrap the corresponding composition of operators in a map. Maps are exported to the CCQ code generation tool that includes a compiler to produce a CEP execution-time module (i.e., the CCQ) which is then loaded into the CEP execution engine.

The **Run Time Environment** contains a CEP platform with a continuous query engine and a set of SOAP message integration layers to allow it to send and receive messages to and from Web Services. The continuous query engine provides the capability for the system to receive, process, correlate and analyze SOAP messages against a CCQ. However, since Web services communicate through the use of SOAP messages, intermediate adapters are required to provide entry and exit points to the engine. These intermediate adapters are of two types: input and output wrappers. An input wrapper receives SOAP messages from the source's service interface and transforms it to the representation appropriate for the CEP engine and then sends it to the engine. Similarly, an output wrapper receives events produced by the engine and transforms it to a SOAP message before forwarding the message to the target service.

4.4 Demonstration and Experimentation

A demonstration of our prototype can be seen at: `http://www.youtube.com/watch?v=g05ciEPZ_Zc`.

5 Related Work

As [3] describes, there are many commercial tools to achieve Web Service *signature mediation* and solve signature incompatibilities, including: Microsoft's Biztalk mapper[2], Stylus Studio's XML Mapping tools[3], SAP's Exchange Infrastructure (XI) Mapping Editor[4] and Altova's MapForce[5]. Academic research also exists in resolving signature incompatibilities through the use of semantic web technology (i.e., OWL), such as that described in [5] that presents a "context-based mediation approach to [...] the semantic heterogeneities between composed Web services", and the Web Service Modeling Ontology (WSMO) specification [8] that provides a foundation for common descriptions of Web Service behavior and operations. This research does not attempt to resolve the associated problem of protocol incompatibility, however.

Active research is also being performed into the adaptation of web service protocols, although all work we have surveyed does not tackle both problems of signature and protocol incompatibility and all use different approaches to the CEP-based technique presented. For example, although [3] presents mediation patterns together with corresponding BPEL templates, a technique and engineering approach for semi-automatically identifying and resolving identifying protocol mismatches and a prototype implementation (the *Service Mediation Toolkit*), it does not solve the signature adaptation problem. Similarly, [2] "discusses the notion of *protocol compatibility* between Web Services" and [1] again only "focusses on the protocol mismatches, leaving data mismatches apart" —

[2] `http://www.microsoft.com/biztalk/en/us/default.aspx`
[3] `http://www.stylusstudio.com/xml_mapper.html`
[4] `http://www.sdn.sap.com/irj/sdn/nw-xi`
[5] `http://www.altova.com/mapforce/web-services-mapping.html`

i.e., they present solutions to protocol mismatches and do not tackle the associated problem of signature incompatibility. Our chosen approach solves both signature and protocol incompatibilities.

6 Conclusion

Web service incompatibilities are found in either their message signatures or protocols. This paper presents an CEP approach to adapt Web Service interactions and resolve these conflicts. Using predefined operators represented as configurable automata allows us to automatically CEP generate adapters capable of intercepting incoming messages sent between services and adapting their structure, type and number into the desired output message(s). Our future work will be in two areas: (i) performing extensive testing on real services, and (ii) developing tools to assist service designers to generate adapters.

Acknowledgment. The research leading to these results has received funding from the European Community's Seventh Framework Program [FP7/2007–2013] under grant agreement 215482 (S-CUBE). We thank Marie-Christine Fauvet, Djamal Benslimane and Marlon Dumas for their comments and contributions on earlier stages of this work.

References

1. Ardissono, L., Furnari, R., Petrone, G., Segnan, M.: Interaction Protocol Mediation in Web Service Composition. International Journal of Web Engineering and Technology 6(1), 4–32 (2010)
2. Dumas, M., Benatallah, B., Nezhad, H.R.M.: Web Service Protocols: Compatibility and Adaptation. IEEE Data Engineering Bulletin 31, 40–44 (2008)
3. Li, X., Fan, Y., Madnick, S., Sheng, Q.Z.: A Pattern-Based Approach to Protocol Mediation for Web Services Composition. Information & Software Technology 52(3), 304–323 (2010)
4. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman (2001)
5. Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F., Dustdar, S.: A Context-Based Mediation Approach to Compose Semantic Web Services. ACM Transactions on Internet Technology (TOIT) 8(1), 1–23 (2008)
6. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F.: Semi-Automated Adaptation of Service Interactions. In: Proceedings of the 16th International Conference on World Wide Web, pp. 993–1002 (2007)
7. Papazoglou, M.: Web Services: Principles & Technology. Pearson Education (2008)
8. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1(1), 77–106 (2005)
9. Taher, Y., Aït-Bachir, A., Fauvet, M.C., Benslimane, D.: Diagnosing Incompatibilities in Web Service Interactions for Automatic Generation of Adapters. In: Proceedings of the 23rd International Conference on Advanced Information Networking and Applications (AINA 2009), pp. 652–659 (2009)

10. Taher, Y., Marie-Christine, F., Dumas, M., Benslimane, D.: Using CEP TEchnology to Adapt Messages Exchanged by Web Services. In: Proceedings of the 17th International Conference on the World Wide Web (WWW 2008), Beijing, China, pp. 1231–1232 (April 2008)
11. Taher, Y., Nguyen, D.K., van den Heuvel, W.J., Ait-Bachir, A.: Enabling Interoperability for SOA-Based SaaS Applications Using Continuous Computational Language. In: Proceedings of the 3rd European ServiceWave Conference, Ghent, Belgium, pp. 222–224 (December 2010)

Employing Dynamic Object Offloading as a Design Breakthrough for SOA Adoption

Quirino Zagarese*, Gerardo Canfora, and Eugenio Zimeo
{quirino.zagarese,gerardo.canfora,eugenio.zimeo}@unisannio.it

Department of Engineering, University of Sannio

Abstract. In several application contexts, Web Services adoption is limited due to performance issues. Design methods and migration strategies from legacy systems often propose the adoption of coarse-grained interfaces to reduce the number of interactions between clients and servers. This is an important design concern since marshaling and transferring small parts of complex business objects might entail sensible delays, especially in high latency networks. Nevertheless, transferring large data in coarse-grained interactions might bring useless data on the client side, whereas a small part of the transferred object is actually used.

This paper presents a novel approach to extend existing Web services run-time supports with dynamic offloading capabilities based on an adaptive strategy that allows servers to learn clients behaviors at runtime. By exploiting this approach, service based applications can improve their performances, as experimental results show, without any invasive change to existing Web services and clients.

Keywords: web-services, performance, xml serialization, service design, adaptation.

1 Introduction

Web Services emerged as a breakthrough for cross-platform communication in heterogeneous environments thanks to the use of standards such as XML, SOAP [12] and WSDL[13]. However, performance issues slew down their adoption in environments where performance matters. This is mostly due to high latency and low bandwidth characterizing world-wide networks, something really tough to deal with when it comes to marshaling and transferring complex business objects. Meaningful examples of such objects can be found in the PLM (Product Life-cycle Management) area, where large data concerning structural and management aspects of products need to be shared among multiple systems [4], and in the telecommunication area, where providers sales and ordering systems expose interfaces to extract data on customers' orders [3]. In these complex and often multi-organisational environments, design an migration strategy to embrace the SOA paradigm can be very challenging.

* The work of this author is supported by Provincia di Benevento.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 610–617, 2011.

The need to minimize the overhead due to network latency often leads to the design of coarse-grained service interfaces. However, this approach can cause unnecessary data transfers, since each service invocation implies the serialization of a large XML document, even if the client needs only a small portion of such document. Over the years, many solutions have been proposed to minimize client-server interaction time, by compressing XML documents or by incrementally loading XML structures. The XMill[1] tool, Cheney's SAX events encoding [6] and Rosu's incremental dictionary-based compression[7] represent important results concerning the former approach. Incremental loading has been proposed in [2], where web services calls are inserted in XML documents in place of their results, as well as in [8], where a placeholder of the result of an invocation is propagated among services and lazily loaded only when needed. The latter approach seems more promising since it allows for reducing the amount of data transferred on the network by delivering to the caller only the data used by the application, whereas XML compression could still be used to further reduce the size of exchanged messages. However, the idea of incremental loading is in its infancy and current Web services technology does not yet provide tools for smart lazy-serialization of object attributes.

This paper tries to fill this gap by proposing an architecture for a middleware aimed at optimizing network-based interactions for data-intensive services. Optimization is achieved by applying a technique that combines eager and lazy serialization of the attributes of objects resulting from services invocations. We call such technique *Dynamic Object Offloading*.

The remainder of the paper is organized as follows. Section 2 introduces and characterizes the problem. Section 3 describes the architecture of the proposed middleware. Section 4 presents some preliminary experimental results. Section 5 discusses the results, concludes the paper and outlines future work.

2 Problem Characterization

Most systems that expose functions through web-services communicate with other systems through an inherently eager-loading approach: every time a new request is received, a result is computed, serialized to the network, and unserialized on the client side. A different approach consists of adopting a lazy-loading strategy. In such case, for each incoming request, a result is computed as well, but XML fragments get serialized to the client as soon as they are needed. The former approach may lead to unnecessarily transferred data, but the interaction is completed within a single request-response cycle. This means any network latency overhead contributes only once to the overall time needed to complete the client-server interaction. The latter approach ensures that only needed fragments get serialized to the client, but leads to multiple request-response cycles. Consider an object *Obj* containing n attributes whose sizes are defined in vector $S = \{s_1, s_2, \ldots, s_n\}$. If the web-services stack is based on an eager-loading strategy, the time needed by the client application in order to retrieve *Obj* can be computed as:

$$T_{retrieve} = L + \frac{\sum_{i=0}^{n} s_i}{Th}$$

where L represents SOAP transport latency and Th stands for SOAP transport throughput. In order to characterize $T_{retrieve}$ for a lazy-loading strategy, we introduce a binary vector $A = \{a_1, a_2, \ldots, a_n\}$ where the generic element a_i is defined as:

$$a_i = \begin{cases} 1 \text{ if } client \;\; uses \;\; n_i \\ 0 \; otherwise \end{cases}$$

As mentioned, for a lazy-loading strategy, multiple request-response cycles occur, since each access to a not-yet-loaded attribute triggers a new request. For such scenario, $T_{retrieve}$ can be defined as:

$$T_{retrieve} = L(1 + \sum_{i=0}^{n} a_i) + \frac{\sum_{i=0}^{n} a_i s_i}{Th}$$

To achieve a performance gain in terms of $T_{retrieve}$, the following must apply:

$$L \cdot \sum_{i=0}^{n} a_i + \frac{\sum_{i=0}^{n} a_i s_i}{Th} < \frac{\sum_{i=0}^{n} s_i}{Th} \tag{1}$$

If we assume that all attributes in *Obj* exhibit the same size S and define the number of accessed attributes as $N_{access} = \sum_{i=0}^{n} a_i$, then we can rewrite (1) as follows:

$$LN_{access} + \frac{N_{access} S}{Th} < \frac{NS}{Th}$$

where N is the total number of attributes in *Obj*. A performance gain is achieved if:

$$N_{access} < \frac{NS}{L \cdot Th + S}$$

So far, we have explored when a plain lazy-loading strategy performs better than a eager-loading one. One could combine the mentioned approaches in order to minimize both request-response cycles and downloaded fragments. In this case, N_{access} must be split into two terms: N_{eager} and N_{lazy}. The former represents the number of attributes downloaded during the first request-response cycle; the latter concerns the number of attributes downloaded on demand, by issuing further requests. The problem then consists of keeping the value of N_{lazy} as low as possible in order to minimize $T_{retrieve}$.

3 Middleware Architecture

In this section, we introduce an architecture for a middleware that minimizes $T_{retrieve}$ by combining eager and lazy-loading. We split XML documents returned by Simple Object Access Protocol (SOAP) Web Services into two sets

respectively having N_{eager} and N_{lazy} cardinalities: the former contains the eagerly loaded object attributes, the latter contains the lazy loaded ones. For each service request, the server decides which attributes are likely to be used by the client application, thus eagerly serializing them, and which are not, making them available for lazy access.

Choosing which attributes should be eagerly serialized cannot be evaluated in a static way, since clients behaviour is not likely to be known a priori. For this reason, we propose to monitor clients behaviour and characterize them by considering the following scenarios:

1. clients accessed an eagerly loaded property of the object
2. clients accessed a lazily loaded property of the object
3. clients did not access a lazily loaded property of the object
4. clients did not access an eagerly loaded property of the object.

The first and third scenarios describe desirable situations, as the system predicts the actual client behavior: no cache-misses are issued, nor any eager serialization is wasted. The second and forth scenarios concern negative situations; the former takes place when the server does not eagerly serialize an object attribute, but the client tries to use it; the latter happens when the server eagerly serializes an attribute, but the client does not use it.

Clients behaviors are stored in a *Knowledge Base* (KB) component which exposes two kinds of operations: "query", to decide if an object attribute should be eagerly serialized, and "update", to instruct it about clients behaviour.

Figure 1 describes a client-server interaction mediated by our proposed middleware. Gray components realize a standard architecture based on Web Services. *Service Interceptor* is a JAX-WS [5] compliant component, able to access service incoming and outgoing messages. It is responsible for analysing services invocations results and applying dynamic offloading, based on information obtained by querying KB. It implements the Interceptor architectural pattern that allows services to be added transparently to a framework and triggered automatically when certain events occur[11].

Data Access Layer (DAL) is a set of components that hide objects attributes loading strategy from the client-side and notify server about clients behaviour. *Lazy Data Access Service* (LDAS) enables lazy loading of objects attributes. It is responsible for updating the Knowledge Base, on the basis of notifications from DAL, and delivering offloaded attributes. It behaves like a dictionary where each entry is an *InvocationID-ObjectPropertyName* pair.

A sample invocation scenario can help better describing the whole architecture. When a service request is issued by a client (no. 1 on the diagram), Service Interceptor inspects the outgoing response message and queries the KB (2, 3) in order to decide which attributes of the returned object should be eagerly serialized to the client, after assigning an invocation id to them (5), and which ones should be offloaded to the LDAS (4). Offloading is performed asynchronously, in order to minimize response time.

After the client receives the result of the invocation, DAL keeps track of accesses to object properties. When the client requests an eagerly loaded attribute,

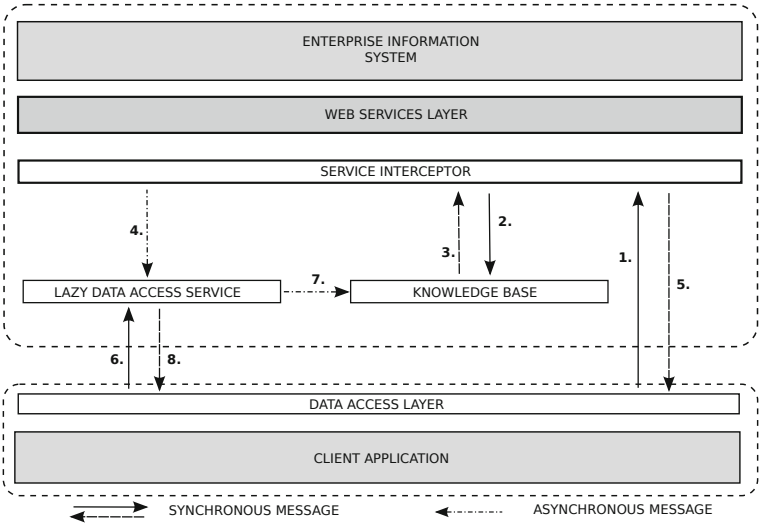


Fig. 1. Interaction phases in the proposed architecture

DAL simply saves such information and returns the attribute. When a request for an offloaded attribute is issued, DAL forwards a request (6), containing the id related to the invocation that generated the root object and accesses related data recorded since the last miss, to LDAS that, in turn, asynchronously updates KB (7) and serializes the requested attribute back to the client (8).

4 Preliminary Evaluation

We have implemented the described architecture in a prototype system based on the Apache CXF framework [10]. CXF is a JAX-WS compliant implementation and offers a flexible API for message interception.

The *LDAS* component has been implemented as a separate CXF Web service. The *KB* implements a profiling strategy, where for each attribute in the business object returned by the target service, the following probability is computed and updated:

$$P_{access} = \frac{\#attribute\_accesses}{\#service\_invocations}$$

We defined a target service that allows the client to retrieve a sample business object containing 100 fixed-size attributes. The client is emulated through a component that actually invokes the target service, accesses the attributes of the returned objects and possibly raises requests to *LDAS*. The access pattern is specified through a binary input vector *Acc*, where *Acc*[*i*] is defined as follows:

$$Acc[i] = \begin{cases} 1 \text{ if } client \;\; uses \;\; n_i \\ 0 \; otherwise \end{cases}$$

where n_i is the i^{th} attribute in the business object.

The evaluation process has been based on two cases. First, we compare eager loading, pure-lazy loading, random dynamic offloading and learning dynamic offloading for a client that never changes its behaviour. The eager loading approach makes the server always serialize the whole object; when pure lazy loading is used, the server always returns an empty object and the client can retrieve attributes by issuing new requests; with random dynamic offloading, the server randomly decides, for each attribute, whether it should be serialized; learning dynamic offloading implements a simple strategy where the probability to use an attribute is computed as the number of accesses to such attribute, divided by the number of invocations of the service, as outlined in section 3. In this phase, we make the client access a constant number of attributes; in the first round, the client constantly accesses 10 of the 100 attributes in the business object. Then we make the client access 20 of the 100 attributes and so on, until all of the attributes in the business object get accessed. Each round consists of 100 invocations of the target service. For each round we evaluate the average $T_{retrieve}$. The process has been repeated for three different attribute sizes: 1MB, 100KB, 10KB. These are realistic sizes for attributes in business objects characterizing the PLM area, such as object attributes encapsulating CAD files or images. By repeating measures for different attribute sizes, we are also able to outline how multiple request-response cycles impact $T_{retrieve}$, when network overhead dominates object size.

In the second case, we compare the mentioned strategies for a client that randomly decides to access or not a generic attribute. We evaluate the average $T_{retrieve}$ over 1000 invocations of the target service.

Table 1. Experimental results with static and random client behavior

	N_{acc}	10	20	30	40	50	60	70	80	90	100	*Rand.*
	Size	1MB										
$T_{retrieve[s]}$	Eager	119,89	119,89	119,89	119,89	119,89	119,89	119,89	119,89	119,89	119,89	119,89
	Lazy	12,53	25,02	37,50	49,98	62,47	74,95	87,44	99,92	112,40	124,89	62,47
	Rand.	66,21	72,85	79,39	85,23	91,44	97,85	103,28	110,15	115,97	122,36	91,25
	Learn.	12,04	24,03	36,02	48,00	59,99	71,98	83,97	95,96	107,95	119,94	94,78
	Size	100KB										
$T_{retrieve[s]}$	Eager	11,75	11,75	11,75	11,75	11,75	11,75	11,75	11,75	11,75	11,75	11,75
	Lazy	1,72	3,39	5,06	6,73	8,40	10,07	11,74	13,41	15,08	16,75	8,40
	Rand.	6,70	7,56	8,42	9,30	10,05	10,94	11,79	12,57	13,41	14,25	10,08
	Learn.	1,23	2,40	3,58	4,75	5,93	7,10	8,28	9,45	10,63	11,80	10,29
	Size	10KB										
$T_{retrieve[s]}$	Eager	1,22	1,22	1,22	1,22	1,22	1,22	1,22	1,22	1,22	1,22	1,22
	Lazy	0,67	1,28	1,90	2,52	3,14	3,75	4,37	4,99	5,60	6,22	3,14
	Rand.	0,93	1,25	1,54	1,88	2,19	2,48	2,78	3,13	3,39	3,72	2,18
	Learn.	0,17	0,29	0,42	0,54	0,66	0,78	0,90	1,03	1,15	1,27	2,06

Experimental results are reported in table 1 where each column contains the average $T_{retrieve}$ values for a round of 100 invocations, given a specific N_{access} value. The right most column contains the average $T_{retrieve}$ values obtained in the second case. Results show that a learning strategy outperforms all the other strategies in terms of $T_{retrieve}$, if the client application keeps a static behaviour and if it does not access all the attributes of the business object.

We expected that a pure lazy-loading strategy would outperform eager-loading. Surprisingly, a random offloading strategy outperforms eager-loading, as long as the client does not access more than 90% of the attributes. The table shows that a pure eager-loading strategy is nearly always the worst choice, if the business object exhibits coarse-grained attributes. As object granularity decreases, lazy-loading and random offloading become less competitive. Learning offloading still outperforms other strategies regardless of object granularity.

Results from the second case show that learning offloading is still a good compromise, also if the client exhibits an unpredictable behaviour. The results shown in the right most column of table 1, confirm again that whilst lazy-loading outperforms other strategies when the business object contains coarse-grained attributes, it becomes the worst choice as soon as granularity decreases. On the other hand, random and learning offloading are the best compromises.

5 Conclusion

Experimental results show that learning offloading can considerably reduce the amount of time needed by a client to retrieve a business object, or a useful subset of its attributes strictly needed to execute its business logic. Client predictability is a key factor: a client exhibiting a static behaviour leads to a remarkable reduction of $T_{retrieve}$. However, even when the client is characterized by a random behaviour, results show that learning offloading is still a good compromise between pure-eager and pure-lazy loading.

An important requirement to make correct decisions on eager or lazy serialization is to have a good idea of how a client is going to use the object returned by a service invocation. The decision should be based on a realistic expectation of the customer usage, which, in turn, depends upon the specific task to be performed in a given interaction. A strong basis of information on how the object attributes will be accessed will result in optimized decisions that will exhibit fewer failures in the field. In this context, usage profiles can be useful to actively gather information on how clients are actually using the objects they receive from services. We intend exploring usage profiles to characterize families of interaction between a set of services to perform a given task; usage profile information will then be used to guide predictive decisions on eager and lazy serialization of sets of attributes based on the task to be performed.

The attributes of an object do not live in isolation and, for any given usage profile, certain subsets of attributes are more likely to be used than others. Thus, once the usage profile is known, access to an object attribute can suggest which attributes are likely to be accessed in the near future; this information can be

used to implement predictive strategies for serialization. Of course, relationships among attributes are non-deterministic, and this calls for probabilistic models. Among these models, Bayesian networks are a promising tool to infer the probability that an attribute be accessed, given the accesses to other attributes, by modeling the conditional dependencies via a directed acyclic graph. We plan exploring learning procedures for Bayesian networks to learn the network that depicts the dependencies among attributes from usage profiles data.

Acknowledgements. The authors would like to thank C. Sementa, A. K. Hosseini and P. Cantiello for stimulating discussion on preliminary ideas behind this work [9].

References

1. Liefke, H., Sucio, D., Mill, X.: An Efficient Compressor for XML Data. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp. 153–164 (2000)
2. Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., Weber, R.: Active XML: Peer-to-peer data and Web services integration. In: VLDB (2002)
3. Amer-Yahia, S., Kotidis, Y.: A Web-service architecture for efficient XML. data ex-change. In: ICDE (2004)
4. Siemens PLM Software, Open product lifecycle data sharing using XML (2011)
5. JSR-000224 Java API for XML-Based Web Services 2.0, `http://jcp.org/aboutJava/communityprocess/final/jsr224`
6. Cheney, J.: Compressing XML with multiplexed hierarchical PPM models. In: Data Compression Conference, pp. 163–173 (2001), `http://citeseer.ist.psu.edu/cheney01compressing.html`
7. Rosu, M.C.: A-soap: Adaptive soap message processing and compression. In: Proceedings of the IEEE International Conference on Web Services, Salt Lake City, Utah, USA, pp. 200–207 (2007)
8. Tretola, G., Zimeo, E.: Extending Web Services Semantics to Support Asynchronous Invocations and Continuation. In: ICWS 2007, pp. 208–215 (2007)
9. Zagarese, Q., Sementa, C., Hosseini, A.K., Cantiello, P.: Improving the performance of Web Services by Dynamic Object Offloading. In: WIP Proceedings of PDP 2011, Ayia Napa, Cyprus (2011)
10. Apache CXF, `http://cxf.apache.org/`
11. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture. In: Patterns for Concurrent and Networked Objects, vol. 2, p. 101. John Wiley & Sons (2000)
12. SOAP Specifications, `http://www.w3.org/TR/soap/`
13. Web Service Definition Language, `http://www.w3.org/TR/wsdl`

A Survey of SOA Migration in Industry

Maryam Razavian and Patricia Lago

Department of Computer Science, VU University Amsterdam, The Netherlands*
{m.razavian,p.lago}@vu.nl

Abstract. Migration of legacy software to service-based systems is an increasingly important problem area. So far, many SOA migration approaches have been proposed in both industry and academia. There is, however, considerable difference between SOA migration approaches defined in academia and those emerged in industry. This difference pinpoints a potential gap between theory and practice. To bridge this gap, we conducted an industrial interview survey in seven leading SOA solution provider companies. Results have been analyzed with respect to migration activities, the available knowledge assets and the migration process. In addition, industrial approaches have been contrasted with academic ones, hence discussing differences and promising directions for industry-relevant research. As a result we found that, in fact, all companies converge to the same, one, common SOA migration approach. This suggests that, with experience, enterprises mature toward a similar approach to SOA migration.

1 Introduction

Migration of legacy systems to service-based systems enables enterprises to achieve advantages offered by SOA, while reusing the business functions embedded in the legacy systems. Enterprises nowadays have many software systems that are needed to be modernized because they are difficult to change and they cannot cope with everlasting requirements changes. Service-enabling the legacy systems allows enterprises to modernize their pre-existing business functions as added-value services, and therefore achieve SOA promises such as agility and flexibility. Hence, identification of migration strategies for service engineering is critical for migration of legacies, and SOA adoption in industrial setting.

So far, many SOA migration approaches have been proposed in both industry and academia with the ultimate goal of adoption in practice. There is, however, considerable difference between SOA migration approaches defined in academia and those emerged in industry. For example, while scientific approaches mainly take a reverse engineering perspective, industrial practitioners developed best practices in forward engineering from requirements to SOA technologies, where legacy code is not transformed but used as a reference. This difference pinpoints

* This research has been received funding from Jacquard (contract 638.001.206 SAPI-ENSA: Service-enAbling PreexIsting ENterprISe Assets); and FP7 contract 215483 (S-Cube). We would like to thank all architects that participated in this study.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 618–626, 2011.

a potential gap between theory and practice. One of the key causes of such a gap is that the approaches proposed in academia do not fully fit the main goals and needs of practice. To bridge this gap, it is necessary to understand the properties of migration approaches that are both feasible and beneficial for practice.

This paper provides deeper understanding of the types of migration approaches in industrial practice. To this end, we conducted an industrial interview survey in seven leading SOA solution provider companies. To the best of our knowledge, this is the first survey of this kind. With the objective of understanding the industrial migration approaches, we designed and executed the interviews. Each interview was analyzed considering the constituent conceptual elements of a migration process as proposed in [1], including the activities carried out, the available knowledge assets, and the overall organization of migration process. Furthermore, we looked for the best practices that companies have developed out of experience for successful legacy migration.

As a result we found that, in fact, all companies converge to the same, one, common SOA migration approach. This suggests that industrial migration approaches converge to a similar set of activities, process organization, and best practices, in other words, with experience enterprises mature toward a similar approach to SOA migration. In addition, we contrasted the industrial approaches with academic ones, which we identified from a previous Systematic Literature Review (SLR) on SOA migration [2]. Here we use the results of the SLR to discuss the differences and draw promising directions for industry-relevant research.

2 Results

To gain an understanding on industrial migration approaches, we needed to typify the approaches in a unified manner. For this purpose, we used the SOA Migration Framework (SOA-MF) introduced in our earlier work [1] (see Fig. 1.I). The analysis of the approaches revealed patterns common among various companies[1]. These are listed in four key findings presented in this section. Each finding is summarized in a Reflection Box, which is followed by detailed discussion of the finding. Furthermore, each finding is compared with the results of our previous study on academic SOA migration approaches (the SLR mentioned in Section 1). Major differences between industrial approaches and academic ones can *reflect* gaps between theory and practice.

2.1 Migration Activities

Reflection Box.1.

- **F1.1.** Different companies share the same set of activities for migration.
- **F1.2.** Industrial migration approaches converge to one, common, type of migration.

[1] Due to space constraints, our research methodology including the research questions, the study design, and data analysis method are made available at `http://www.few.vu.nl/~mrazavi/IndustrialSurveyAppendix.pdf`

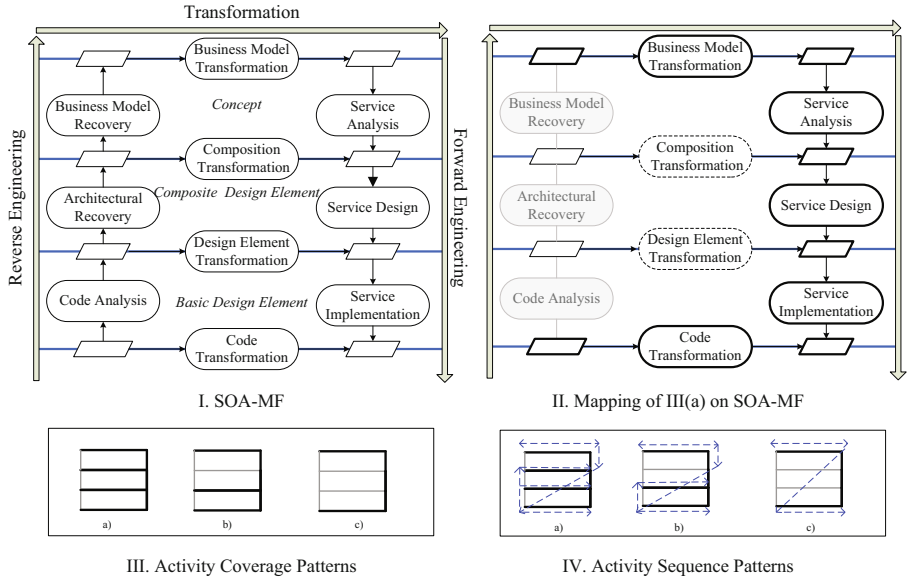


Fig. 1. Industrial Approaches Mappings

To answer *what is done* in industrial approaches, we identified the constituent activities of various approaches and mapped them on SOA-MF. Fig. 1.III, represents the schematic forms of those mappings. Mappings revealed two main findings: a) industrial approaches share the same set of activities for migration and b) industrial approaches are convergent to a subset of those activities. The two findings are further discussed in the following.

Finding F1.1. Various companies, independent from the company type (i.e., consultancy vs. in-house) and migration application domain, share the same set of activities for migration. This is evident from Fig. 1.III, where the activities correspond to three graphically similar coverage patterns. It should be noted that the similarity among coverage patterns, thanks to expressiveness of SOA-MF, indicates the conceptual similarity of constituent activities and artifacts of the migration approaches. According to [2], SOA migration approaches with similar set of activities constitute a migration family. Similarly, the three similar approaches identified in the interviewed companies belong to the same family.

Contrast with theory. While the industrial approaches are all members of one family, the SLR revealed that the academic approaches belong to eight very different families. By covering different sets of activities each of these eight families provide a very different view on what SOA migration entails. For instance, one family reverse engineers the legacy code and transforms the extracted code segments to services, another family only covers the forward engineering subprocess. Considering the industrial approaches, all the approaches are categorized into (only) one of the eight families. Interestingly, the size of that family,

called industrial family, is the smallest as compared to the others (i.e. 3% of academic approaches). Thus, one could conclude that 97% of the academic approaches do not fit in industrial family. This may indicate that academic research might be digging into aspects (like sub-processes and techniques) that are less relevant for industry. On the contrary, by looking at the characteristics of the industrial family research could better focus on the open research questions pertaining such family and hence have a better chance to close the gap between academic research results and industry needs.

Finding F1.2. By further analyzing the activities of the industrial approaches, we found that those common among all approaches, called *core activities*, are the ones shown in Fig. 1.II with bold boxes. The *variable* activities, i.e., those not common to all approaches, pertain to the coverage of the two transformation activities shown in Fig. 1.II by dashed line boxes. Furthermore, we observed that the core activities are those performed more frequently and systematically, while the variable activities are carried out less frequently and in an ad-hoc manner. More precisely, the limitations posed by legacy systems makes the variable activities less frequent. Several of the interview participants mentioned that, transformations that require decomposing the legacy systems are rarely carried out because they are not feasible as legacy systems are mainly monolithic. Furthermore, we observed that core activities are mainly supported by the state-of-the-practice methodologies and techniques such as SOMA [3]. The variable activities, however, are mainly carried out using local best practices. Consequently, we argue that, due to higher feasibility of the core activities and support of well-established methodologies and techniques, the industrial migration approaches are characterized by core activities.

Contrast with theory. None of the migration approaches in the SLR fully covers the core activities. I.e., none of the academic approaches comprehensively supports the type of migration that is both feasible and beneficial in industrial setting. This indicates an important gap between the migration activities emerged from practice and the ones researched in academia.

2.2 Sequencing of Migration Activities

Reflection Box.2.

- **F2.** In the industrial migration approaches the To-Be situation initiates and drives the migration.

By providing the mappings on SOA-MF, previous section addressed what activities are covered in the industrial migration approaches. Here we focus on what is the sequencing of those activities. There are two main types of sequencing of activities in the migration approaches, namely arc-shaped and bowl-shaped [4]. In summary, in arc-shaped approaches migration is driven by As-Is situation, while it is the To-Be situation that drives the bowl-shaped ones. All the industrial approaches elicited by our study were bowl-shaped.

This categorization of approaches is based on the graphical representation resulted from mapping their sequencing of activities on SOA-MF (e.g. Fig. 1.IV). The sequencing of activities in an arc-shaped approach starts from the reverse engineering sub-process. In this category, the As-Is situation initiates and drives the migration. Unlike the arc-shaped category, the bowl-shaped one starts from forward engineering and the To-Be situation is the main driver of migration.

Finding F2. The bowl-shaped sequencing of activities in industrial approaches implies the following: in all of the migration approaches the To-Be situation, characterized by requirements or properties of the target service-based system, drives and shapes the migration. To shape the migration process, first the To-Be situation is defined within the forward engineering sub-process; further, the To-Be situation is compared with the As-Is and as such, the legacy elements are selected and re-shaped to services. A question that arises is why industries perform migration in a bowl-shaped manner. Some of the participants, in one way or another, stated that in order to reach the migration goals they need to have the To-Be situation as the primary shaping force behind migration. As such, we conclude that to ensure achieving the migration goals, companies shape their migration decisions primarily by the To-Be situation.

Contrast with theory. Unlike the industrial migration approaches, the academic ones are mainly arc-shaped. In the SLR only 30 % of the primary studies are categorized as bowl-shaped approaches and the rest are arc-shaped. As such, 70% of the approaches do not support To-Be driven migration, which is considered as the best practice among the practitioners. This highlights promising opportunities for research to focus on how to support To-Be driven migration.

2.3 Legacy Understanding through Personalization

Reflection Box.3.

- **F3.1.** The industrial migration approaches do not use reverse-engineering techniques to understand the legacy systems.
- **F3.2.** The required knowledge is elicited from the stakeholders who own the knowledge.

Understanding the legacy systems plays an important role in SOA migration as it enhances extracting the best candidates among existing legacies for migration to SOA. In traditional software engineering, this understanding is gathered by extracting the representation of the legacy systems using reverse engineering techniques. As shown in Fig. 1.III, we observed that in the industry-defined approaches none covers the reverse-engineering subprocess. This observation resulted in two key findings discussed in the following.

Finding F3.1. To gain the required understanding of the legacy system, the industrial approaches do not use reverse engineering techniques. This is due to the following two reasons: *a)* the knowledge about the pre-existing system mainly resides in the stakeholders' minds (e.g. maintainer, developer, and architect). As such, the stakeholders know what functionalities are supported, and where they

are located in the legacy system. As a result, reverse engineering of the pre-existing system is not favorable considering the little Return On Investment (ROI) it brings.
b) the legacy systems are usually comprised of a set of heterogeneous systems that are implemented in different programming languages ranging from COBOL to Java. As a result, for reverse engineering of the code different tools are needed and this implies a considerable amount of costs.

Contrast with theory. To understand the legacy systems, more than 60% of the approaches in the SLR use reverse engineering techniques. Those approaches extract the representations of the legacy systems using techniques such as code analysis and architectural recovery. Only one of the academic approaches (out of 39), supports the legacy understanding without reverse engineering techniques (i.e. using structured interviews)[5]. This indicates an important gap between theory and practice since reverse engineering is not favorable in practice.

Finding F3.2. We further observed that the industrial migration approaches elicit the relevant knowledge by directly asking the stakeholders, who own, developed, or maintained that system. More precisely, knowledge about the legacy system mainly remains tacit in stakeholders minds. As such, understanding is achieved by person-to-person knowledge elicitation. We argue that, this type of knowledge elicitation is in-line with *personalization* knowledge management strategy [6]. Personalization deals with exchanging tacit type of knowledge. Using personalization, the legacy understanding is gained by knowing 'who knows what' and consequently sharing the tacit knowledge about the legacy systems in that regard.

Contrast with theory. In the SLR, all the approaches focus on capturing the knowledge by documenting it. As such, they are in-line with codification strategy addressing explicit documentation of the knowledge. The results of this study, however, suggests the importance of personalization. As such, research is needed to improve elicitation techniques, especially targeted for SOA migration, supporting personalization strategy.

2.4 Service Extraction by Defining the Ideal Services

Reflection Box.4.
- **F4.1.** The main driver in extraction of the legacy assets for migration is the portrait of ideal service.
- **F4.2.** Approaches emerged out of more experience portray the ideal services in more detail.

Finding F4.1. The migration approaches, inherently, embrace trade-off analysis between the level of reuse of legacy elements and characteristics of the ideal services. We observed that, in this trade-off analysis, the industrial approaches assign considerably higher weight to the later rather than the former. To do so, first they determine the ideal services during the forward engineering sub-process. Later, those ideal services are re-shaped in a way that the reuse of

pre-existing assets are realized. This way, the portrait of the ideal service is the main driver of service extraction. That is, the services identified from the pre-existing capabilities would likely be substantially different in the absence of that portrait of the ideal service. This is in-line with our other finding that all the migration approaches are bowl-shaped meaning that the To-Be candidate services guide the analysis and transformation of the As-Is legacy elements.

Contrast with theory. A characteristic of the bowl-shaped approaches is having the ideal services (To-Be situation) as the main driver in service extraction. As such, this finding points out the same gap between theory and practice as discussed in finding F.2, namely inadequate support of To-Be driven migration.

Finding F4.2. We further observed that, industrial approaches vary in the level of detail in which they portray their ideal services. Some of the approaches only define the capability of the desired services at conceptual level (e.g. order business service), while some others also provide the design of such services along with its associated service contract (e.g. order software service design). Some of the approaches externalize the constraints which each service should meet, while some others do not explicitly consider any constraints. Interestingly, we observed that the companies with more experience in providing service-based solutions tend to define the ideal services more detailed compared to the ones with less experience. Hence, we argue that the extent to which the ideal service is codified is an indicator of the maturity of the migration approach.

Contrast with theory. Detailed description of the ideal services is a best practice that companies have developed with experience. Interestingly, we could not trace back this best practice to the academic approaches.

3 Discussion

In software engineering as an applied science, research in principle should serve the final purpose of being applied in practice. The extent to which this principle is supported by research, however, has been subject of debate for decades, and remains a still unsolved problem. The premier conference on software engineering featured in 2011 a panel on "What Industry Wants from Research" discussing the current gaps between theory and practice, and how to address them. All panel members in one way or another hinted the following cause of such gap: what research proposes does not fit the fundamental problems, goals, strategies and weaknesses of practice. We argue that, this paper is a step towards filling the theory and practice gap as it sheds light on how migration is performed in practice and further contrasting it with how academic research addressed the migration problem. By identifying the characteristics which make these approaches favorable for practice, we could identify directions for future research that have better chance of adoption by practitioners.

I) Migration approaches fitting core activities. Getting back to finding F1, we argue that core activities can act as a frame of mind confining the migration approaches that are more aligned with practice. From that perspective,

one would see that, for instance, the approaches addressing wrapping the applications as a whole are more in-line with practitioners concerns, compared to the ones addressing the automatic recovery of the legacy architecture. Hence, this frame of mind pinpoints the types of industry-relevant research in SOA migration methodologies and techniques.

II) To-Be driven migration approaches. As noted in finding F2, inadequate support for the bowl-shaped approaches in academia highlights promising opportunities for research to focus on how to support To-Be driven migration. For instance, future research can focus on addressing the following challenge of the practitioners: how to systematically elicit and capture the migration drivers and how to shape the migration process using those drivers.

III) Legacy understanding without reverse-engineering. Although reverse engineering is not covered in industrial migration approaches (see finding F3), elicitation of the knowledge about the legacy system is crucial for a successful migration. In this regard, research can benefit practice by providing methods, techniques, or guidelines that facilitate elicitation of migration-relevant knowledge from different sources of such knowledge.

IV) Legacy evaluation from multiple perspectives. As noted, companies evaluate and extract the legacy assets for migration to SOA by depicting their ideal services. This is, however, done in an ad-hoc manner, which may hinder successful service extraction. An immediate concern calling for further research is how to systematically evaluate pre-existing legacy assets based on different aspects of the ideal services.

4 Conclusions

This paper explored the types of migration approaches employed by leading SOA solution providers in practice. Results show that by supporting similar set of activities, process organization, and best practices, industrial migration approaches do converge to one, common, type of migration. As such, this paper suggests that the industrial approaches mature towards a similar approach to SOA migration. Further findings (removed for sake of space) show that industrial approaches, strictly follow incremental migration.

In spite of what academics think, practitioners still face difficulties in consolidating to a successful yet cost-effective migration approach. The many available methods often prove to be abstract or commercial to be applicable. By contrasting the industrial migration approaches and the academic ones, this paper emphasizes important gaps between theory and practice and consequently sketches the promising industry-relevant research directions. Those research directions enable finding solutions to problems that industrial practice confronts in real-world migration cases and is tailored to individual needs.

When a company wants to devise or select a specific approach for migration of its pre-existing assets to services there are many issues that need to be resolved. In this study we identified the type of industrial migration approaches that

is feasible in practice. What issues, goals, assumptions, and decisions explicitly make that specific type of migration favorable in practice, though, is yet unclear. We are carrying out follow-up studies to identify the goals, assumptions and issues that shape the migration decision making process.

References

1. Razavian, M., Lago, P.: Towards a Conceptual Framework for Legacy to SOA Migration. In: Fifth International Workshop on Engineering Service-Oriented Applications (WESOA 2009), pp. 445–455 (2010)
2. Razavian, M., Lago, P.: A Frame of Reference for SOA Migration. In: Di Nitto, E., Yahyapour, R. (eds.) ServiceWave 2010. LNCS, vol. 6481, pp. 150–162. Springer, Heidelberg (2010)
3. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Gariapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. IBM Syst. J. 47, 377–396 (2008)
4. Razavian, M., Lago, P.: A Dashboard for SOA Migration (under submission, 2011)
5. Lewis, G., Smith, D.B.: Developing Realistic Approaches for the Migration of Legacy Components to Service-Oriented Architecture Environments. In: Draheim, D., Weber, G. (eds.) TEAA 2006. LNCS, vol. 4473, pp. 226–240. Springer, Heidelberg (2007)
6. Hansen, M.T., Nohria, N., Tierney, T.: What's your strategy for managing knowledge? Harvard Business Review 77(2) (1999)

Forms-Based Service Composition

Ingo Weber, Hye-Young Paik, and Boualem Benatallah

School of Computer Science and Engineering
University of New South Wales
Sydney, NSW, Australia, 2052
{ingo.weber,hpaik,boualem}@cse.unsw.edu.au

Abstract. In many cases, it is not cost effective to automate business processes which affect a small number of people and/or change frequently. We present a novel approach for enabling domain experts to model and deploy such processes from their respective domain as Web service compositions. The approach is based on user-editable service naming, a graphical composition language where Web services are represented as forms, a targeted restriction of control flow expressivity, automated process verification mechanisms, and code generation for executing orchestrations. A Web-based service composition prototype implements this approach, including a WS-BPEL code generator.

1 Introduction

Business process management (BPM) refers to a discipline and software suites that automate, improve, and optimize business processes to enhance productivity [12]. Despite BPM's success, the reality is that today many processes are in fact *not* automated. First, among other reasons, BPM products are not suitably equipped to deal with processes that are ad-hoc [14]. Second, there are costs and high skills involved in implementing automated processes. This affects primarily the "long tail of processes" [9], i.e. processes that are less structured, that do not affect many people uniformly, or that are not critical.

In recent reports and studies [10,11], the split between BPM technology and its value for end-users is acknowledged. The reports include recommendations on increasing the relevance of BPM for end-users, allowing process changes by non-technical personnel, and reducing the complexity of BPM technology.

Motivated by the need for user-friendly BPM technology, the goal of this work is to devise an approach to support domain experts in their long-tail process automation needs. We focus on processes that can be implemented as Web service compositions. As a user group, we target business domain experts, i.e., non-IT professionals. We believe that these often have a good understanding of the processes they participate in; and that they are able to abstract from single process instances to the bigger picture of the process model containing alternatives and exceptions. An example is hiring a new employee, where HR recruiters have a good understanding of the default process and under which circumstances they may deviate from it.

The traditional approaches to BPM for process automation have inherited from programming. We believe this causes difficulties for the targeted users. The following lists the problems and requirements that are relevant to our goal:

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 627–635, 2011.

- Programming requires writing code as abstract artifacts, symbolic or textual [5]. It is hard for untrained users to match their tasks to the abstractions.
- The so-called *selection barrier* [6] refers to the fact that often the users do not know how to express what they want the computer to do.
- Immediate program verification is needed to give feedback to the user [5].
- The system needs to understand high-level instructions of the user and translate them to a formal representation [5].

Our goal is to create a language and system for forms-based service composition, to allow domain experts to address their idiosyncratic, long-tail process automation needs themselves.Since we want to include processes where parts are executed conditionally, we propose a scripting approach to design process models. We focus on a graphical representation based on forms for designing processes. In the approach, services can be described using names that are meaningful to the user, and independent of the services' technical names. We use these names during service discovery and in the process design.

Also, we aim at keeping the complexity of process modeling low, in order to make the approach applicable to domain experts. As such, we include features to verify the correct combination of the modeled control and data flows through automated verification techniques. A code generator can automatically translate the models to an executable language. The complexity is further limited through a targeted restriction of control flow expressivity. However, to enable fast execution, we include an automatic parallelization technique in our code generation. We note that the approach outlined above is very different to other service composition or workflow tools (e.g., JOpera, Kepler, Taverna)[1], which tend to support highly complex modelling and programming capacity, and demand higher level of assumed knowledge from their users. We conduct a preliminary case study with use cases from the financial domain: data analysis processes such as finding a correlation between news and stock price changes. The use cases are taken from our industry partner, Sirca[2].

In summary, the contributions of this paper are the following:

- A forms-based composition method, where (i) forms are linked to Web services; and (ii) compositions can be modeled in a restricted, yet powerful generic language.
- Immediate automated process verification for reducing the burden on the user to build a correct composition.
- Automatic code generation with parallelization, to generate executable orchestrations from the forms-based composition language.

We also present a prototype and show how the approach can be enacted.[3]

[1] `www.jopera.org/`, `kepler-project.org/`, `www.taverna.org.uk/`, respectively

[2] `http://www.sirca.org.au`

[3] A full technical report and a demonstration video of the prototype can be found at `http://www.cse.unsw.edu.au/~FormSys/FormSys/`. The tool has also been demonstrated at the 9th Intl. Conference on Business Process Management (BPM) in August 2011, without publication.

2 Forms-Based Service Composition Approach

In this section, we explain how services are created and managed in a repository, how they are represented for domain experts, and how domain experts can then model processes graphically. We start by introducing a running example.

2.1 Use Case: News and Financial Data Analysis Process

Research and development within Sirca on the possible utilization of available datasets led to the implementation of numerous Web services [13]. The types of Web services range from query/search, data cleaning, to complex statistical analysis. Currently, each Web service is invoked by a simple user interface based on Web forms, and the services operate independently. One such example is described in Fig. 1, where each step represents a Web service.

1. Find news data: *e.g., news data on the company 'BHP'*
2. Find performance data: *e.g., hourly stock price summary for code 'BHP.AX'*
3. Merge datasets: *e.g., merge the result data sets from the first two steps*
4. Perform statistical analysis: *e.g., which news were possibly influential on the price*
5. Visualize dataset: *e.g., influential news and the prices*

Fig. 1. Financial data analysis from Sirca, for an Australian mining company 'BHP'

Repeating the above process by operating the Web forms involves around 30 mouse clicks, as well as entering the same information repeatedly at multiple steps. Once the processing is complete, the exact parameters that resulted in a given graph are lost. The set of changing parameters and the details of which service to use with which parameters differs between analysts and their tasks at hand. Therefore, while being repetitive, the processes are required to be flexibly executable or adaptable by the analyst.

Automating such processes is of interest to (i) reduce the required amount of user interaction, and (ii) retain the parameters that led to some visualization. The latter is important, because comparable graphs can be required periodically. With this motivating example in mind, we present our approach next.

2.2 Forms as Service Interface Representations

In our repository, every service is collectively represented by a WSDL document, a user-editable name, an icon, and forms as graphical representations of input and output messages. While WSDL needs to be present in the repository, it is completely hidden from the domain expert designing processes.

Graphical Representations: The service is a computational entity that performs some function, which is represented with an icon. For example, Figure 2 shows the icon for the "Find News Data" service.

Fig. 2. Find News Data service as icon

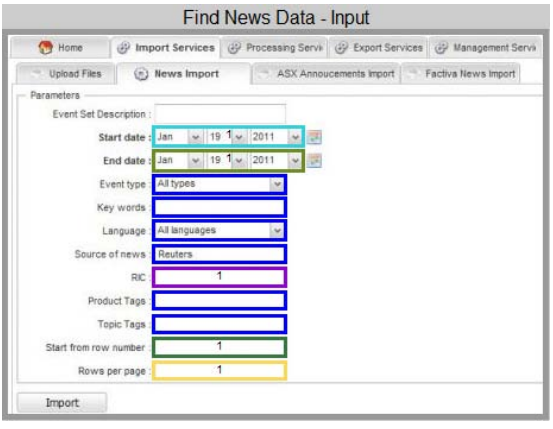


Fig. 3. Graphical representation of Input Message for Find News Data

The technical information about a service is stored as standard WSDL. In fact, a service in our approach corresponds to a WSDL operation. An invocable WSDL operation has an input and an optional output message[4]. The input and output messages for a service are represented as forms – reflecting the service's running user interface. Fig. 3 shows an example of an input message as a form. The names of form fields, corresponding to the service's input/output parameters, as well as the names of services themselves can be set to names meaningful to the users, which are used during search and composition of services. The form representation is also useful when the domain experts specify data mappings between messages.

Each data field from the message which is to used in process designs has a box associated to it, somewhere in the form. How the boxes correspond to data fields in the message has to be marked up manually for now, to enable automatic execution of designed processes. By default, the form could be rendered from the XML Schema type that belongs to the respective message. However, given our focus on domain experts, we believe that the form representation should be something the user is already familiar with. Hence, a screenshot of the UI through which the user commonly accesses this service makes a good representation.

Using Service Names Meaningful to Users: Besides the technical information from WSDL and the graphical representations, the services in our repository are also given non-technical names. These names are created by users, at the time when entering a service into the repository. In the process modeling tool, the user can assign different names to services, if they prefer them for the processes at hand. For instance, while some user called a service "Find News Data", another user may refer to it as "Import News Data".

[4] For simplicity, we currently neglect fault messages and exception handling, as well as certain XML Schema constructs and certain WSDL features.

2.3 Forms-Based Control Flow Modeling

Using the rich service descriptions outlined in the previous section, modeling an executable process becomes a matter of drag-and-drop and clicking. We treat control flow and data flow as two separate, but not independent, layers. The control flow serves as an abstract process description: which services should be executed, under which conditions and in which order? The data flow adds more detail, by specifying how the input and output message fields of the various services interact.

In order to retain the focus on domain experts, the control flow modeling is restricted: services are arranged into a single sequence, and may be subject to some condition. If the condition evaluates to true in a process instance, the associated services are executed; if it evaluates to false, they are skipped in this instance. The conditions are free text, and, through code generation, are turned into questions to users starting process instances. The above restrictions have strong impact on expressiveness[5]. However, anecdotal evidence from experience with industry contacts suggests that forcing the occasional user of our system to understand the particularities of the semantics of an expressive language alienates most targeted domain experts. Therefore, we try to keep the control flow modeling as simple as possible – see Fig. 4 for a screenshot from our tool.

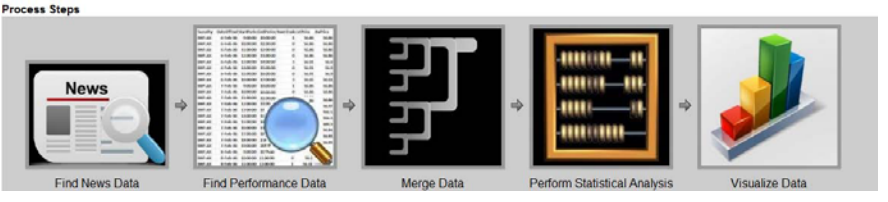


Fig. 4. Graphical Process Modeling of the Running Example

2.4 Forms-Based Data Flow Modeling

The data flow modeling works roughly as follows in our approach. Each service has an input and possibly an output message. A message consists of a set of fields, and has a form as graphical representation – c.f. Fig. 3. Data fields from one message can be mapped to data fields of another message in our approach. For instance, in our running example, the outputs of the two import services can be mapped to the merge service's input; the from/to dates in one data import service can be mapped to the date range of the other import service; etc.

Data fields of input messages fall into one of three categories, with respect to processes: user-static (i.e., always the same value for some user), process-static (i.e., always the same value for some process model), and process-instance-specific data (i.e., likely to be different for each instance of a process).

[5] In the technical report[3] , we discuss the expressiveness issue of our approach in detail with regards to workflow patterns.

In our method, the user can define certain kinds of mappings between / assignments to fields of messages of different Web services:

- specifying that an output field of one service corresponds to the input field of another (**output-input mapping**);
- specifying that two (or more) input fields of separate services will get the same value from the process-specific user input (**input-input mapping**);
- specifying a static value for an input field, including `null` (**static assignment**).

2.5 Process Verification

Implicitly, a data flow graph is created from output-input mappings (the directed edges of the graph). There can be contradictions within the data flow, or between the data flow and control flow of a process. Our approach includes an automatic verification for a set of problems that may appear. With the verification technique, the modeler can be kept from modeling, e.g., (illegal) loops in the data flow. More details, including a (semi-)formal treatment of the problems and solutions (based on well-known graph algorithms), are in the technical report[3].

2.6 Code Generation for Process Execution

The goal of modeling a process in our approach is to automate the execution of repetitive tasks. In order to determine the necessary input for the process, our solution combines all inputs for all services, and removes any field which is the target of a mapping or static assignment. The result forms a message with the consolidated input data format to start the process. For this message, we generate a Web form, where the user can enter the information and trigger an instance of the process. Analogously, the outputs of all services are consolidated to one output message of the process, for which again a Web form is created.

When desired by the user, our approach can parallelize steps in the process based on the data flow, by ignoring additional constraints from the control flow. This is achieved by mapping the problem onto graph algorithms, as before. The resulting, non-redundant process is then translated to WS-BPEL directly. Details can be found in the technical report[3].

3 Related Work

Here we present related work in brief; an exhaustive discussion of related work is included in the technical report[3].

Mashups have similar goals to our system. Topics such as data harvesting and visualization, composition of existing data and UI, and custom views or UIs for existing services are common in mashups [18]. While this may facilitate certain processes, the predominant composition paradigms in mashups are event-based synchronization [19] and data flow between components, not control flow over

process activities [4]. While there is some overlap between mashup approaches and ours, we see them as largely complementary.

End-User Process Modelling: Todor Stoitsev [15] investigates using Task Management (Outlook plugin) for "process modelling by example": the system tracks how people split up larger tasks into subtasks, and delegate some subtasks to others; this can be used as input to a workflow design tool. [8] describes a technique for constructing process models with formal execution semantics from informal models (e.g., Powerpoint drawings). The technique stops at producing BPMN, but possibly could be extended to generate executable models. However, the missing aspects to enable that (e.g., service selection, data flow) are not explored. [3] describes BPEL4UI / MarcoFlow: a language and tool for enabling BPEL designers to incorporate distributed UI composition in BPEL processes. Mashup-like UI components are synchronized between each other (for a single user), with UI components at other users, and the process. Microsoft InfoPath[6] essentially is a code-free software engineering tool. However, it is still for users familiar with programming, e.g., who know how databases work or what Web service are. *PICTURE* is a domain-specific modeling method and notation for public administration [1]. The key differences to our work are: PICTURE targets capturing the processes, and does not support creating executable processes; and PICTURE is domain-specific, whereas our tool is generic.

End-User Programming: A field of research that has a similar goal to ours, although in a different domain, is *end-user programming (EUP)*. EUP is the umbrella term for approaches that "make limited forms of programming sufficiently understandable and pleasant that end users will be willing and able to program" [2]. An approach in EUP that we consider closely related work is *CoScripter* [7], which primarily focuses on personal processes in the scope of browsing and using Web applications. The user can record, play and publish/share such browser processes on a public Wiki. The processes are stored in a simple end-user understandable language, using natural language keywords such as "go to <URL>" and "click on <link>". In CoScripter, all steps in a script need to be standard operations in a browser. While closely related to our work in terms of the understandability of process steps and the user focus, it does not support Web service invocation or conditional execution.

Own work: The work presented herein is also related to our own work. The tool in its current form has been presented (without publication)[3]. The tool and approach are significant extensions of earlier work for processes made up of PDF forms [17]. The PDF form-filling services are, in turn, the result of a separate work [16], with which FormSys Process Designer shares some database tables. The idea to relate Web services and their messages to forms stems from the earlier work, but representing arbitrary WSDL Web services through forms is a novel contribution of this paper.

[6] `http://office.microsoft.com/en-us/infopath/`

4 Conclusion, Discussion and Future Work

We have presented a forms-based service composition approach which allows domain experts with little technical knowledge to encode idiosyncratic, repetitive business processes themselves from design to execution.

Our preliminary evaluation revealed the following as desirable: user-editable input/output forms of the process; conditional data mappings; and process simulation in the design environment. For our use cases, the number of data mappings stayed reasonably small and therefore using colors to represent them was sufficient. However, if too many colors are required, they can eventually become confusing. These findings guide part of our immediate future work; in particular, a richer language for data mapping is under investigation, to enable more complex mappings and conditions over data fields. The tool will be further tested by our industry partner, and user studies will be conducted.

Acknowledgements. This work has been supported by a grant from the Smart Services CRC[7]. We thank Maurice Peat, Fethi Rabhi, Kader Lattab, and Angel Lagares Lemos for their valuable feedback.

References

1. Becker, J., Algermissen, L., Pfeiffer, D., Räckers, M.: Bausteinbasierte Modellierung von Prozesslandschaften mit der PICTURE-Methode am Beispiel der Universitätsverwaltung Münster. Wirtschaftsinformatik 49, 267–279 (2007)
2. Cypher, A., Dontcheva, M., Lau, T., Nichols, J. (eds.): No Code Required - Giving Users Tools to Transform the Web. Morgan Kaufmann (2010)
3. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: From People to Services to UI: Distributed Orchestration of User Interfaces. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 310–326. Springer, Heidelberg (2010)
4. Di Lorenzo, G., Hacid, H., Paik, H.-Y., Benatallah, B.: Data Iintegration in Mashups. SIGMOD Rec. 38(1), 59–66 (2009)
5. Harel, D.: Can Programming Be Liberated, Period? Computer 41, 28–37 (2008)
6. Ko, A.J., Myers, B.A., Aung, H.H.: Six learning barriers in end-user programming systems. In: VLHCC 2004, pp. 199–206 (2004)
7. Leshed, G., Haber, E., Matthews, T., Lau, T.: CoScripter: Automating & Sharing How-To Knowledge in the Enterprise. CHI Letters: Human Factors in Computing Systems 10(1), 1719–1728 (2008)
8. Mukherjee, D., Dhoolia, P., Sinha, S., Rembert, A.J., Gowri Nanda, M.: From Informal Process Diagrams to Formal Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 145–161. Springer, Heidelberg (2010)
9. Oracle White Paper. State of the Business Process Management Market (August 2008), `http://tinyurl.com/3c4u436` (accessed November 20, 2009)
10. Pettey, C., Goasdu, L.: Gartner Reveals Five Business Process Management Predictions for 2010 and Beyond. Gartner Press Release (January 13, 2010), `http://www.gartner.com/it/page.jsp?id=1278415` (accessed September 2, 2010)

[7] `http://www.smartservicescrc.com.au`

11. Reijers, H.A., van Wijk, S., Mutschler, B., Leurs, M.: BPM in Practice: Who Is Doing What? In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 45–60. Springer, Heidelberg (2010)
12. Richardson, C., Vollmer, K., Clair, C.L., Moore, C., Vitti, R.: Business Process Management Suites, Q3 2009 – The Need For Increased Business Agility Drives BPM Adoption. Forrester TechRadar For BP & A Pros (August 13, 2009)
13. Robertson, C., Rabhi, F., Peat, M.: Consumer Information Systems and Relationship Management: Design, Implementation and Use. In: A Service-Oriented Approach towards Real Time Financial News Analysis. IGI Global (2011)
14. Schurter, T.: BPM State of the Nation 2009. bpm.com, `http://www.bpm.com/bpm-state-of-the-nation-2009.html` (accessed November 25, 2009)
15. Stoitsev, T.: End-User Driven Business Process Composition. PhD thesis, TU Darmstadt, Fachbereich Informatik, Telekooperation (2009)
16. Weber, I., Paik, H., Benatallah, B., Gong, Z., Zheng, L., Vorwerk, C.: FormSys: Form-processing Web Services. In: WWW 2010: Proceedings of the 19th International World Wide Web Conference, Demo Track (2010)
17. Weber, I., Paik, H.-Y., Benatallah, B., Vorwerk, C., Gong, Z., Zheng, L., Kim, S.W.: Managing Long-Tail Processes Using FormSys. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 702–703. Springer, Heidelberg (2010)
18. Wong, J., Hong, J.: What Do We "Mashup" When We Make Mashups? In: WEUSE 2008, pp. 35–39 (May 2008)
19. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12(5), 44–52 (2008)

Contractually Compliant Service Compositions*

Enrique Martínez, Gregorio Díaz, and M. Emilia Cambronero

Department of Computer Science
University of Castilla - La Mancha, Spain
{emartinez,gregorio,emicp}@dsi.uclm.es

Abstract. In the field of service-oriented computing, an e-contract is used to regulate the acceptable behaviours of the services taking part in a composition. *C-O Diagrams* are a visual model for the specification of deontic e-contracts, including reparations, conditional clauses and real-time restrictions. In this work we define a set of satisfaction rules based on timed automata to see whether a composition is compliant with the contract specification, providing the model with the mathematical rigour necessary for formal verification.

Keywords: Contracts, deontic logic, formal verification, visual models, timed automata.

1 Introduction

An *e-contract* is defined as a contract regulating business-to-business interactions over the Internet. Its purpose is to ensure that the partners taking part in the business process (possibly Web Services) comply with certain obligations, permissions and prohibitions by means of specifying a set of clauses. Moreover, these clauses can include the conditions required to be applied, the time boundaries to be satisfied, and references to secondary contracts (reparations).

In [4] we have already presented a visual model for the specification of e-contracts called *C-O Diagrams*. The approach followed is inspired by the formal language $\mathcal{CL}$ [5], in which a contract is expressed as a composition of obligations, permissions and prohibitions over actions, and reparations for obligations and prohibitions can be defined. The main contribution of this work is the definition of a set of satisfaction rules for *C-O Diagrams* based on timed automata [1]. These rules allow us to formally check if all the possible behaviours of a service composition are compliant with the e-contract specified by a *C-O Diagram*. Let us note that in this work we does not focus on how the behaviours of a service composition are translated into timed automata, as many other works in the literature are related to this aspect (e.g. [2,3]).

The rest of the work is structured as follows: Section 2 is a brief description of *C-O Diagrams* and its syntax. Section 3 defines the set of satisfaction rules. Finally, in Section 4, we present the conclusions and future work.

* Partially supported by the Spanish government (cofinanced by FEDER founds) with the project TIN2009-14312-C02-02 and the JCCLM regional project PEII09-0232-7745. The first author is supported by the European Social Fund and the JCCLM.

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 636–644, 2011.

2 C-O Diagrams Description and Syntax

In Fig. 1 we show the basic element of *C-O Diagrams*. It is called a **box** and it is divided into four fields. On the left-hand side of the box we specify the conditions and restrictions. The *guard* **g** specifies the conditions under which the contract clause must be taken into account (boolean expression). The *time restriction* **tr** specifies the time frame during which the contract clause must be satisfied (deadlines, timeouts, etc.). The *propositional content* **P**, on the center, is the main field of the box, and it is used to specify normative aspects (obligations, permissions and prohibitions) that are applied over actions, and/or the specification of the actions themselves. The last field of these boxes, on the right-hand side, is the *reparation* **R**. This reparation, if specified by the contract clause, is a reference to another contract that must be satisfied in case the main norm is not satisfied (a *prohibition* is violated or an *obligation* is not fulfilled, there is no reparation for *permission*), considering the clause eventually satisfied if this reparation is satisfied. Each box has also a name and an agent. The *name* is useful both to describe the clause and to reference the box from other clauses, so it must be unique. The *agent* indicates who is the performer of the action.

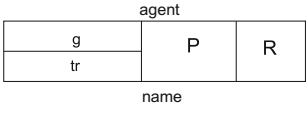


Fig. 1. Box structure

These basic elements of *C-O Diagrams* can be refined by using AND/OR/SEQ refinements, as shown in Fig. 2. The aim of these refinements is to capture the hierarchical clause structure followed by most contracts. An **AND-refinement** means that all the subclauses must be satisfied in order to satisfied the parent clause. An **OR-refinement** means that it is only necessary to satisfy one of the subclauses in order to satisfy the parent clause, so as soon as one of its subclauses is fulfilled, we conclude that the parent clause is fulfilled as well. A **SEQ-refinement** means that the norm specified in the target box (*SubClause2* in Fig. 2) must be fulfilled after satisfying the norm specified in the source box (*SubClause1* in Fig. 2). By using these structures we can build a hierarchical tree with the clauses defined by a contract, where the leaf clauses correspond to the atomic clauses, that is, to the clauses that cannot be divided into subclauses. There is another structure that can be used to model **repetition**. This structure

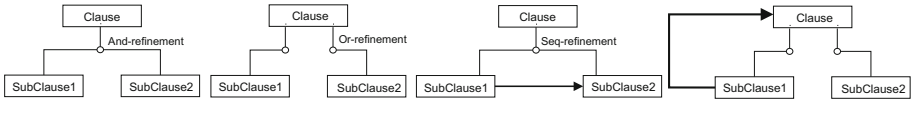


Fig. 2. AND/OR/SEQ refinements and repetition in *C-O Diagrams*

is represented as an arrow going from a subclause to one of its ancestor clauses (or to itself), meaning the repetitive application of all the subclauses of the target clause after satisfying the source subclause. For example, in the right-hand side of Fig. 2, we have an **OR-refinement** with an arrow going from *SubClause1* to *Clause*. It means that after satisfying *SubClause1* we apply *Clause* again, but not after satisfying *SubClause2*.

We have given here an abridged description of *C-O Diagrams*. A more detail description can be found in [4], including a qualitative and quantitative evaluation underlining the advantages of having a visual model for the specification of e-contracts, and a discussion on related work.

Definition 1. *(C-O Diagrams Syntax) We consider a finite set of real-valued variables $\mathcal{C}$ standing for clocks, a finite set of non-negative integer-valued variables $\mathcal{V}$, a finite alphabet Σ for atomic actions, a finite set of identifiers $\mathcal{A}$ for agents, and another finite set of identifiers $\mathcal{N}$ for names. The greek letter ϵ means that and expression is not given, i.e., it is empty.*

We use C to denote the contract modelled by a C-O Diagram. The diagram is defined by the following EBNF grammar:

$$
\begin{aligned}
C &:= (agent, name, g, tr, O(C_2), R) \mid \\
&\quad (agent, name, g, tr, P(C_2), \epsilon) \mid \\
&\quad (agent, name, g, tr, F(C_2), R) \mid \\
&\quad (\epsilon, name, g, tr, C_1, \epsilon) \\
C_1 &:= C\,(And\ C)^+ \mid C\,(Or\ C)^+ \mid C\,(Seq\ C)^+ \\
C_2 &:= a \mid C_3\,(And\ C_3)^+ \mid C_3\,(Or\ C_3)^+ \mid C_3\,(Seq\ C_3)^+ \\
C_3 &:= (\epsilon, name, \epsilon, \epsilon, C_2, \epsilon) \\
R &:= C \mid \epsilon
\end{aligned}
$$

where $a \in \Sigma$, $agent \in \mathcal{A}$ and $name \in \mathcal{N}$. Guard g is ϵ or a conjunctive formula of atomic constraints of the form: $v \sim n$ or $v - w \sim n$, for $v, w \in \mathcal{V}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$, whereas timed restriction tr is ϵ or a conjunctive formula of atomic constraints of the form: $x \sim n$ or $x - y \sim n$, for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. O, P and F are the deontic operators corresponding to obligation, permission and prohibition, respectively, where $O(C_2)$ states the obligation of performing C_2, $F(C_2)$ states prohibition of performing C_2, and $P(C_2)$ states the permission of performing C_2. And, Or and Seq are the operators corresponding to the refinements we have in C-O Diagrams, AND-refinement, OR-refinement and SEQ-refinement, respectively.

The simplest contract we can have in *C-O Diagrams* is that composed of only one box including the elements *agent* and *name*. Optionally, we can specify a guard g and a time restriction tr. We also have a deontic operator (O, P or F) applied over an atomic action a, and in the case of obligations and prohibitions it is possible to specify another contract C as a reparation.

We use C_1 to define a more complex contract where we combine different deontic norms by means of any of the different refinements we have in *C-O Diagrams*. In the box where we have the refinement into C_1 we cannot specify an agent nor a reparation because these elements are always related to a single deontic norm, but we still can specify a guard g and a time restriction tr that affect all the deontic norms we combine.

Once we write a deontic operator in a box of our diagram, we have two possibilities as we can see in the specification of C_2: we can just write a simple action a in the box, being the deontic operator applied only over it, or we can refine this box in order to apply the deontic operator over a compound action. In this case we have that the subboxes (C_3) cannot define a new deontic operator as it has already been defined in the parent box (affecting all the subboxes).

3 C-O Diagrams Satisfaction Rules

In this section we define a set of satisfaction rules for *C-O Diagrams* based on timed automata. The purpose of this definition is providing a formal mechanism to check if a service composition behaviour (specified by a timed automaton) is compliant with respect to a contract (specified by a *C-O Diagram*). To define this satisfaction rules we follow the *C-O Diagrams* syntax given in Definition 1. The satisfiability of a contract is defined based on the states of a timed labelled transition system associated to a timed automaton. Basically, a timed automaton (TA) [1] is a tuple (N, n_0, E, I), where N is a finite set of locations (nodes), $n_0 \in N$ is the initial location, E is the set of edges, and I is a function that assigns invariant conditions (which could be empty) to locations. We write $n \xrightarrow{g,a,r}_{s} n'$ to denote $(n, g, a, s, r, n') \in E$, where $n, n' \in N$, g is a guard, a is an action, r is a set of clocks we want to reset, and s is a set of variable assignments. The semantics of a timed automaton is defined as a timed labelled transition system $(Q, q_0, \rightarrow)$, where Q is a set of states, $q_0 \in Q$ is the initial state, and $\rightarrow$ is the set of transitions. Due to the lack of space, refer to [2] for a complete definition of timed automaton and its semantics.

The *C-O Diagrams* satisfaction rules consist of a set of rules where the satisfiability of a contract is defined based on the states of the timed labelled transition system associated to a timed automaton. To define this set of rules we follow the *C-O Diagrams* syntax given in Definition 1.

Definition 2. (*C-O Diagrams* Satisfaction Rules: Part I)

Let $\mathcal{A} = (N, n_0, E, I)$ be a timed automaton, with the associated timed labelled transition system $(Q, q_0, \rightarrow)$ and $q \in Q$. Given a *C-O Diagram* C, one can define $(\mathcal{A}, q) \models C$ ($\mathcal{A}$ in state q satisfies contract C) as follows:

(1) $(\mathcal{A}, q) \models (agent, name, g, tr, O(a), R)$ **iff** $\forall \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$:
- The **main clause holds**, that is, $\exists i \in [1, j-1]$ such that $q_i \xrightarrow{a}_{s} q_{i+1}$ with $n_i \xrightarrow{g\prime,a,r}_{s} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$
- The **main clause does not hold** but **reparation holds**, that is, $R \neq \epsilon$ and $(\mathcal{A}, q_{i+1}) \models R$ for the first $i \in [1, j-1]$ such that $q_i \xrightarrow{d}_{s} q_{i+1}$ with $(n_i, u) \xrightarrow{d} (n_i, u + d)$ and $(u + d) \notin tr$

(2) $(\mathcal{A}, q) \models (agent, name, g, tr, P(a), \epsilon)$ **iff** $\exists \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$ where **the main clause holds**, that is, $\exists i \in [1, j-1]$ such that $q_i \xrightarrow{a}_{s} q_{i+1}$ with $n_i \xrightarrow{g\prime,a,r}_{s} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$

(3) $(\mathcal{A}, q) \models (agent, name, g, tr, F(a), R)$ **iff** $\forall \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$:
– The **main clause holds**, that is, $\nexists i \in [1, j-1]$ such that $q_i \xrightarrow[s]{a} q_{i+1}$ with $n_i \xrightarrow[s]{g\prime,a,r} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$,
– The **main clause does not hold** but **reparation holds**, that is, $R \neq \epsilon$ and $(\mathcal{A}, q_{i+1}) \models R$ for the first $i \in [1, j-1]$ such that $q_i \xrightarrow[s]{a} q_{i+1}$ with $n_i \xrightarrow[s]{g\prime,a,r} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$

Lines (1)–(3) correspond to the satisfaction rules of applying an obligation, a permission or a prohibition over an atomic action a. In the case of obligation, for all the possible paths in our automaton we must have the performance of a by the specified agent (denoted by $agent(a)$) and fulfilling also any condition or time restriction we have specified (denoted by $(g \wedge tr) \in g\prime$). If the obliged action is not performed in the expected time frame, we have the alternative possibility of satisfying reparation R from the moment at which timed restriction is not fulfilled anymore (denoted by $(u + d) \notin tr$). In permission we consider that the performance of a is only necessary in one of the paths. This interpretation of permission is because we think that an automaton satisfying a contract must offer the possibility of performing a permitted action in at least one of its paths. Prohibition is the opposite of permission, so we cannot have a path where we perform the forbidden action a, but in case we perform the action we still have the possibility of satisfying reparation R after that.

Definition 2. (*C-O Diagrams* Satisfaction rules: Part II)

(4) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_1, name_1, g_1, tr_1, O(a), R_1)\, Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq\, (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$
iff $\forall \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$:
– The **first main clause holds**, that is, $\exists i \in [1, j-1]$ such that $q_i \xrightarrow[s]{a} q_{i+1}$ with $n_i \xrightarrow[s]{g\prime,a,r} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$, **and remaining sequence holds**, that is,
$(\mathcal{A}, q_{i+1}) \models (\epsilon, name, g, tr, (agent_2, name_2, g_2, tr_2, C_2, R_2)\, Seq$
$\ldots Seq\ (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$
– The **first main clause does not hold** but its **reparation and remaining sequence holds**, that is, $R \neq \epsilon$ and
$(\mathcal{A}, q_{i+1}) \models (\epsilon, name, g, tr, R_1\, Seq\, (agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq$
$(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ for the first $i \in [1, j-1]$ such that $q_i \xrightarrow[s]{d} q_{i+1}$
with $(n_i, u) \xrightarrow{d} (n_i, u + d)$ and $(u + d) \notin tr$

(5) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_1, name_1, g_1, tr_1, P(a), R_1)\, Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq\, (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$
iff $\exists \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$ where the **first main clause holds**, that is, $\exists i \in [1, j-1]$ such that $q_i \xrightarrow[s]{a} q_{i+1}$ with $n_i \xrightarrow[s]{g\prime,a,r} n_{i+1}$ where
$(g \wedge tr) \in g\prime$ and $agent(a)$, and **remaining sequence holds**, that is,
$(\mathcal{A}, q_{i+1}) \models (\epsilon, name, g, tr, (agent_2, name_2, g_2, tr_2, C_2, R_2)\, Seq \ldots Seq$
$(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$

(6) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_1, name_1, g_1, tr_1, F(a), R_1)\, Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq\, (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$
iff $\forall \langle q_1 \longrightarrow q_2 \longrightarrow \ldots \longrightarrow q_j \rangle$ for $q = q_1$:
– The **first main clause holds**, that is, $\nexists i \in [1, j-1]$ such that $q_i \xrightarrow[s]{a} q_{i+1}$ with $n_i \xrightarrow[s]{g\prime,a,r} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$, **and remaining sequence holds**, that is,
$(\mathcal{A}, q_{i+1}) \models (\epsilon, name, g, tr, (agent_2, name_2, g_2, tr_2, C_2, R_2)\, Seq$
$\ldots Seq\ (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$
– The **first main clause does not hold** but its **reparation and remaining sequence holds**, that is, $R \neq \epsilon$ and
$(\mathcal{A}, q_{i+1}) \models (\epsilon, name, g, tr, R_1\, Seq\, (agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq$

$(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ for the first $i \in [1, j-1]$ such that $q_i \xrightarrow{a}_s q_{i+1}$ with $n_i \xrightarrow[s]{g\prime, a, r} n_{i+1}$ where $(g \wedge tr) \in g\prime$ and $agent(a)$

Lines (4)–(6) correspond to the satisfaction rules for a contract consisting of a SEQ-refinement when the first subcontract of the sequence is just a deontic operator applied over an atomic action. The satisfaction of the contract consists of the satisfaction of the first subcontract of the sequence and, after that, the satisfaction of the rest of the sequence starting from a location we reach after satisfying the first subcontract. Alternatively, if reparation R of the first subcontract is not empty, we can satisfy a SEQ-refinement having this reparation as its first subcontract.

Definition 2. (*C-O Diagrams* Satisfaction Rules: Part III)

(7) $(\mathcal{A}, q) \models (agent, name, g, tr, \mathcal{D}((\epsilon, name_1, g_1, tr_1, C_1, \epsilon)\, \mathcal{REF}$
$(\epsilon, name_2, g_2, tr_2, C_2, \epsilon)\, \mathcal{REF} \ldots \mathcal{REF}\, (\epsilon, name_k, g_k, tr_k, C_k, \epsilon)), R)$ **iff**
$(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent, name_1, g_1, tr_1, \mathcal{D}(C_1), R)\, \mathcal{REF}$
$(agent, name_2, g_2, tr_2, \mathcal{D}(C_2), R)\, \mathcal{REF} \ldots \mathcal{REF}$
$(agent, name_k, g_k, tr_k, \mathcal{D}(C_k), R), \epsilon)$

(8) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_1, name_1, g_1, tr_1, C_1, R_1)\, \mathcal{RE}$
$(agent_2, name_2, g_2, tr_2, C_2, R_2)\, \mathcal{RE} \ldots \mathcal{RE}$
$(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ **iff**
$(\mathcal{A}, q) \models (agent_1, name_1, g \wedge g_1, tr \wedge tr_1, C_1, R_1) \diamond$
$(\mathcal{A}, q) \models (agent_2, name_2, g \wedge g_2, tr \wedge tr_2, C_2, R_2) \diamond \ldots \diamond$
$(\mathcal{A}, q) \models (agent_k, name_k, g \wedge g_k, tr \wedge tr_k, C_k, R_k)$

(9) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_1, name_1, g_1, tr_1, \mathcal{D}($
$(\epsilon, name_{11}, g_{11}, tr_{11}, C_{11}, \epsilon)\, \mathcal{REF}\, (\epsilon, name_{12}, g_{12}, tr_{12}, C_{12}, \epsilon)\, \mathcal{REF} \ldots \mathcal{REF}$
$(\epsilon, name_{1m}, g_{1m}, tr_{1m}, C_{1m}, \epsilon)), R_1)\, Seq\, (agent_2, name_2, g_2, tr_2, C_2, R_2)$
$Seq \ldots Seq\, (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ **iff**
$(\mathcal{A}, q) \models (\epsilon, name, g, tr, (\epsilon, name_1, g_1, tr_1, (agent_1, name_{11}, g_{11}, tr_{11},$
$\mathcal{D}(C_{11}), R_1)\, \mathcal{REF}\, (agent_1, name_{12}, g_{12}, tr_{12}, \mathcal{REF}(C_{12}), R_1)$
$\mathcal{REF} \ldots \mathcal{REF}\, (agent_1, name_{1m}, g_{1m}, tr_{1m}, \mathcal{D}(C_{1m}), R_1), \epsilon)\, Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2) Seq \ldots Seq\, (agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$

In line (7) we have that $\mathcal{D} \in \{ O, P, F \}$ and $\mathcal{REF} \in \{ And, Or, Seq \}$, so it corresponds to the satisfaction rule of applying a deontic norm over an AND-refinement, an OR-refinement or a SEQ-refinement of subcontracts. For all the deontic operators we just propagate them into each one of the subcontracts, as well as reparation R and *agent*, and the satisfaction of the main contract consists of the satisfaction of the refinement $\mathcal{REF}$ of these new subcontracts.

In line (8) we have that $\mathcal{RE} \in \{And, OR\}$ and $\diamond \in \{ \wedge, \vee \}$. It corresponds to the satisfaction rule for an AND-refinement or an OR-refinement of subcontracts with no deontic operator applied over the refinements (they will be specified in the subcontracts). In these cases, the satisfaction of the main contract consists of the conjunction ($\wedge$ for AND-refinement) or disjunction ($\vee$ for OR-refinement) of the satisfaction of each one of the subcontracts, propagating any condition or time restriction in the main contract into these subcontracts (denoted by $g \wedge g_k$ and $tr \wedge tr_k$).

Line (9) corresponds to the satisfaction rule for a SEQ-refinement when the first element of the sequence is a deontic norm $\mathcal{D} \in \{ O, P, F \}$ applied over another refinement of subcontracts $\mathcal{REF} \in \{ And, Or, Seq \}$. In all these cases we propagate the deontic operator into each one of the subcontracts, as well as

reparation R and *agent*, so the satisfaction of the main contract consists of the satisfaction of the SEQ-refinement when the first element of the sequence is the refinement $\mathcal{RE}\mathcal{F}$ we have before but now combining these new subcontracts.

Definition 2. (*C-O Diagrams* Satisfaction Rules: Part IV)

(10) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (\epsilon, name_1, g_1, tr_1, (agent_{11}, name_{11}, g_{11}, tr_{11}, C_{11}, R_{11})$
$\mathcal{RE}\,(agent_{12}, name_{12}, g_{12}, tr_{12}, C_{12}, R_{12})\,\mathcal{RE}\ldots\mathcal{RE}$
$(agent_{1m}, name_{1m}, g_{1m}, tr_{1m}, C_{1m}, R_{1m}), \epsilon)\; Seq\,(agent_2, name_2, g_2, tr_2, C_2, R_2)$
$Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ **iff**
$(\mathcal{A}, q) \models (\epsilon, name, g, tr, ((agent_{11}, name_{11}, g_{11} \wedge g_1, tr_{11} \wedge tr_1, C_{11}, R_{11})\; Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2)\; Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k))$
$\mathcal{RE}\,((agent_{12}, name_{12}, g_{12} \wedge g_1, tr_{12} \wedge tr_1, C_{12}, R_{12})\; Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2)\; Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k))$
$\mathcal{RE}\ldots\mathcal{RE}\,((agent_{1m}, name_{1m}, g_{1m} \wedge g_1, tr_{1m} \wedge tr_1, C_{1m}, R_{1m})\; Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2)\; Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k)), \epsilon)$

(11) $(\mathcal{A}, q) \models (\epsilon, name, g, tr, (\epsilon, name_1, g_1, tr_1, (agent_{11}, name_{11}, g_{11}, tr_{11}, C_{11}, R_{11})$
$Seq\,(agent_{12}, name_{12}, g_{12}, tr_{12}, C_{12}, R_{12})\; Seq\ldots Seq$
$(agent_{1m}, name_{1m}, g_{1m}, tr_{1m}, C_{1m}, R_{1m}), \epsilon)\; Seq\,(agent_2, name_2, g_2, tr_2, C_2, R_2)$
$Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$ **iff**
$(\mathcal{A}, q) \models (\epsilon, name, g, tr, (agent_{11}, name_{11}, g_{11} \wedge g_1, tr_{11} \wedge tr_1, C_{11}, R_{11})\; Seq$
$(agent_{12}, name_{12}, g_{12} \wedge g_1, tr_{12} \wedge tr_1, C_{12}, R_{12})\; Seq\ldots Seq$
$(agent_{1m}, name_{1m}, g_{1m} \wedge g_1, tr_{1m} \wedge tr_1, C_{1m}, R_{1m})\; Seq$
$(agent_2, name_2, g_2, tr_2, C_2, R_2)\; Seq\ldots Seq\,(agent_k, name_k, g_k, tr_k, C_k, R_k), \epsilon)$

Lines (10) corresponds to the satisfaction rule for a SEQ-refinement when the first element of the sequence is a refinement $\mathcal{RE} \in \{And, OR\}$ of subcontracts, with no deontic operator applied over it. We consider SEQ-refinement as distributive over AND-refinement and OR-refinement, so the satisfaction of this contract consists of the satisfaction of the contract where we have applied this property, propagating any condition or time restriction in the AND-refinement or the OR-refinement into their subcontracts.

Finally, line (11) corresponds to the satisfaction rule for a SEQ-refinement when the first element of the sequence is another SEQ-refinement of subcontracts, with no deontic operator applied over it. We consider SEQ-refinement to be associative, so the satisfaction of this contract consists of the satisfaction of the contract where we have applied this property to have only one SEQ-refinement, propagating any condition or time restriction in the internal SEQ-refinement into their subcontracts.

After defining these satisfaction rules, the definition of the algorithm checking whether a timed automaton $\mathcal{A} = (N, n_0, E, I)$ with associated timed labelled transition system $(Q, q_0, \rightarrow)$ satisfies a *C-O Diagram* C is quite straightforward. It returns "YES" if $(\mathcal{A}, q_0) \models C$, otherwise it returns "NO".

Example 1. Let us consider the **Second_Update** *C-O Diagram* of the *Software Provision System* case study presented in [4]. It models a contract we denote as C. According to the *C-O Diagrams* syntax, this diagram can be written as:

$(\epsilon, Second\_Update, \epsilon, \epsilon, (Software\,provider, Sends\_Update2, \epsilon, tr_4, O(a_4), R_4)$
$Seq\,(\epsilon, Client\_Second\_Behavior, \epsilon, \epsilon, (Client, Second\_Payment, \epsilon, \epsilon, O(a_5), \epsilon)$
$And\,(Client, Second\_Changes, \epsilon, \epsilon, F(a_6), R_6), \epsilon), \epsilon)$

where we use tr_4 to denote the temporal restriction we have in $Sends\_Update2$, R_4 is the reparation we define for this clause, and R_6 is the reparation we define for clause $Second\_Changes$, stating the obligation of performing r_{6a} or r_{6b}.

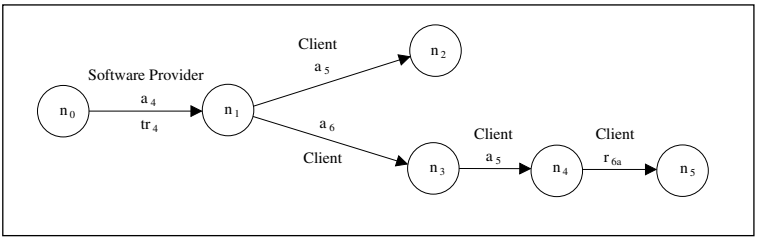


Fig. 3. Automaton $\mathcal{A}$ of *Example 1*

We want to check if the timed automaton $\mathcal{A}$ shown in Fig. 3 satisfies this contract C starting from n_0. First, we have a SEQ-refinement where the first subcontract states the obligation of performing a_4. Then, we apply rule **(4)** and we see that the initial obligation is fulfilled, performing $Software\,provider$ a_4 within time frame tr_4 in the edge from n_0 to n_1. Next, we have an AND-refinement between an obligation and a prohibition, so applying rule **(8)** we have to check the satisfaction of the obligation and the satisfaction of the prohibition from n_1. According to rule **(1)**, we have to see that a_5 is performed by *Client* in every possible path. An edge performing a_5 exists in all cases, so the obligation is satisfied. According to rule **(3)**, we have to see that a_6 is not performed by *Client* in any possible path. For path $n_1 \longrightarrow n_2$ it is fulfilled, but for path $n_1 \longrightarrow n_3$ we have that a_6 is performed. Nevertheless, as there is a reparation R_6 defined, we have to see if this reparation is fulfilled. We can see that, after applying rules **(7)** and **(8)**, it is enough to satisfy the obligation of one of the two actions (r_{6a} or r_{6b}) in order to fulfill the reparation. We have that the edge from n_4 to n_5 performs r_{6a}, so the reparation is fulfilled and the subcontract is eventually fulfilled. Therefore, we conclude that contract C is satisfied by $\mathcal{A}$.

4 Conclusions and Future Work

In this paper we have defined a set of satisfaction rules for *C-O Diagrams* based on timed automata to check whether a service composition is compliant with a contract specified by a *C-O Diagram*. We have also shown an example about how to apply this approach.

As future work, we are planning to apply this model to several case studies in order to check its usefulness in different fields and the evaluation of its computational complexity. We are also working on the definition of a transformation that automatically generates a timed automaton compliant with the contract specified by a *C-O Diagram*. This can be useful for several purposes, for example to check the correctness of the contract specification.

References

1. Alur, R., Dill, D.L.: Automata For Modeling Real-Time Systems. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, Springer, Heidelberg (1990)
2. Cambronero, M.E., Valero, V., Díaz, G., Martínez, E.: Web Services Choreographies Verification. Technical Report DIAB-09-04-1, University of Castilla-La Mancha (2009)
3. Lomuscio, A., Qu, H., Solanki, M.: Towards verifying contract regulated service composition. In: Proceedings of IEEE International Conference on Web Services (ICWS 2008), pp. 254–261 (2008)
4. Martínez, E., Díaz, G., Cambronero, M.E., Schneider, G.: A Model for Visual Specification of e-Contracts. In: Proceedings of the 7th IEEE 2010 International Conference on Services Computing (SCC 2010), pp. 1–8 (2010)
5. Prisacariu, C., Schneider, G.: A Formal Language for Electronic Contracts. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 174–189. Springer, Heidelberg (2007)

Profit Sharing in Service Composition

Shigeo Matsubara

Department of Social Informatics, Kyoto University,
Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan
matsubara@i.kyoto-u.ac.jp
http://www.ai.soc.i.kyoto-u.ac.jp/~matsubara

Abstract. Component services are often provided by different organizations, which needs to determine how to divide the profit obtained for the composite service to the component service providers. Previous studies have mainly focused on the process of aggregating multiple component services into a composite service. However, the process of the profit sharing has not yet discussed sufficiently. This problem can be formalized as a coalition game in the game theory. However, its flexibility of defining the policy of utilizing the services causes a problem. This paper shows that the existing profit sharing methods, more precisely, neither the equal division method nor the division method based on the Shapley value cannot satisfy the following two desiderata; (1) the sufficient level of service provision is attained, and (2) component services are not broken up more than is necessary. Moreover, we examine what factors make difficult to attain the sufficient level of service provision, and give a discussion toward mitigating this problem.

Keywords: Game theory, Shapley value, Incentive problem, Web services.

1 Introduction

An essential part of service-oriented computing technologies is service composition, i.e., a user's task is accomplished by aggregating multiple component services into a single composite service. Service composition has been actively studied [4,1] but the automated composition is still far from being achieved. We consider one of the reasons exists in lacking a sufficient discussion about an incentive problem, although recently incentive issues are getting more attention among the service-oriented computing researchers [2,5].

It seems that previous studies naively assumed that services are voluntarily provided and discussed how to find appropriate services among a huge amount of services. Here, voluntarily does not mean that services are provided with no charge but means that service providers do not strategically restrict the use of their services. However, as shown below, service providers may be motivated to impose a restriction on the use of their services.

A simple method of pricing a composite service is that the service providers set the prices of their own services individually, calculate the sum of each component

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 645–652, 2011.

service included in the composite service, and charge it to the user as a price of the composite service. The providers, however, are often difficult to set optimal prices that maximize their profit. This is because the providers are difficult to know in advance how their services are combined with other services.

A solution to overcome this drawback is to first set the price of the composite service that is equivalent to the user's valuation and then divide the profit to the component service providers included in the composite service. This paper focuses on how to solve a profit sharing problem.

Profit sharing problems are formalized as coalitional game in the game theory [3]. The coalitional game theory assumes that the members included in a coalition are disjoint with each other. That is, it assumes that any combinations of the component services are available if these component services are provided. However, such an assumption does not hold in the domain of web services. The service provider of s_1 can set the term of use in that utilizing s_1 in a composite service $s_1s_2s_3$ is allowed but utilizing s_1 in another composite service s_1s_3 is not allowed.[1]

Thus, we have to examine how this flexibility affects the profit sharing problem. More precisely, this paper examines whether the requirements for profit sharing in Web services are satisfied in the existing profit sharing methods such as equal division, profit sharing based on the Shapley value. We show that these methods cannot attain the sufficient level of service provision, examine what factors deteriorate the level of service provision, and finally give a discussion toward mitigating this problem.

2 Model

In this section we formalize the problem of profit sharing. n service provider agents $S = \{s_1, s_2, \cdots, s_n\}$ and service user agents exist. To keep the discussion simple, we assume that a service provider agent provides a single component service. Thus, the set S can be viewed as the set of component services as well as the set of service provider agents. A composite service can be composed from any subset S_i of S if it is allowed.

The users have their valuation $v : 2^n \rightarrow \Re$ for component services and composite services, which is called as characteristic function, where, $\Re$ is a set of real values and $v(\emptyset) = 0$. This paper assumes that the valuation values are given for all the combinations of component services. In composing services by sequencing component service s_i and component service s_j, it may happen that the type of output in s_i does not compatible with the type of input in s_j. In such a case, we just let the valuation of the composite service to zero.

This paper assumes that zero monotonicity for the characteristic function. Zero monotonicity means that the value of a composite service gets larger as the number of component services included in the composite service increases. The assumption of zero monotonicity does not mean that users should choose the

[1] In this paper, $s_1s_2s_3$ represents a composite service, while s_1,s_2,s_3 represent a set of component services.

services including more components of services. Users have a budget constraint and it may happen that users choose a low-price service, although its quality of service is not high.

The relevant characteristics of a user agent are summarized in that user's type $\theta_i = (v_i, b_i)$ where v_i is the characteristic function and b_i is the budget constraint. We assume that the distribution function of the user's type is known. From this assumption the profit for any combinations of the composite services can be calculated if the prices for each component service and each composite service are given. In addition, this paper does not consider the user's strategic behavior.

As mentioned above, a characteristic of web service is that service providers can flexibly set the term of use of their services. $s_1 s_2$ represents a composite service that each component service is also available, while $[s_1 s_2]$ represents a composite service that each component service is not available.

3 Desiderata for Profit Sharing Methods

Defining appropriate guiding principles is important in developing, maintaining, and utilizing the SOA. Granularity and composability have been discussed as design principles. Incentive issues, however, have not been examined sufficiently. Incentive issues become very significant especially in applying the SOA concept to inter-organizational services domain.

This paper considers the two requirements for profit sharing methods: (1) a variety of component services are provided to meet the user's requirements, (2) the unit of component services is appropriate, that is, a unit service is not too fine-grained. For example, we consider the service is too fine-grained if a dictionary service is divided to services of having indexes A, B, C, and so on. If a unit of service is too fine-grained, the computational burden of calculating service composition appears.

In the rest of this section, we explain why under-provision of services should be considered. It happens that more than one Web services provided by different providers collude with each other and provide only a composite service, that is, do not provide each component service to users. Satisfying the sufficient provision of component services means that component services are provided in the manner that users are allowed to combine these services to other services to achieve the user's task.

Here, note that we do not intend to discuss the one-time usage of the services by service users. Consider $user1$ and $user2$ exist. If the times of utilizing the services are different, the identity of $user1$ might be equal to the identity of $user2$. $user1$ wants to use composite service $s_1 s_2 s_3$ and $user2$ wants to use composite service $s_1 s_3$. Here, if component service s_1 allows users to use s_1 as $s_1 s_2 s_3$ but does not allow users to use s_1 as $s_1 s_3$, under-provision of services occurs. From the viewpoint of $user1$ it is sufficient if $user1$ can use composite service $s_1 s_2 s_3$. This can be dealt with in the ordinal cooperative game theory, i.e., the profit sharing methods such as the Shapley values can be applied. However, if $user2$ exists, whether using $s_1 s_3$ is allowed or not should be considered simultaneously.

Discussion on this paper is different from that about coalitional rationality in the cooperative game theory. For example, the collusion by s_1 and s_2 does not mean that s_1 and s_2 deviate the coalition of $s_1s_2s_3$ and create a new coalition including only s_1 and s_2. s_1 and s_2 will not reject to form a coalition of $s_1s_2s_3$ if it brings larger profit to s_1 and s_2. Here, collusion means that s_1 and s_2 reject to be included in the composite service s_1s_3 nor s_2s_3.

This discussion is caused from the fact that the granularity of the service is not clear. In the profit distribution problem in the game theory, the unit of player(agent) has no ambiguity. For example, in dealing with the problem of corporate alliance, a player represents a company.

We introduce the term of a level of service provision for the later discussions. A level of service provision means to what extent users are allowed to arbitrary combine these services with each other to accomplish their tasks.

4 Drawbacks in the Existing Profit Sharing Methods

This section proves that equal-division method and profit sharing method based on the Shapley value do not satisfy the requirements for the profit sharing in Web services.

4.1 Profit Sharing by Equal Division

A simple method for profit sharing is to share the profit for the composite service equally among the component service providers. This method is simple but has a drawback. Consider the following example. Two component services of s_1, s_2 and a composite service s_1s_2 exist. The characteristic functions are $v(s_1) = 8$, $v(s_2) = 2$, $v(s_1s_2) = 12$. Here, if the profit is equally divided, s_1 and s_2 earn $12/2 = 6$, respectively. However, if the cost for providing component service of s_1 is 7, s_1 suffers a loss. On the other hand, if s_1 provides its service as a single service, its profit becomes 8, i.e., s_1 can make a money. Therefore, even if users want to use the composite service of s_1s_2, s_1 will prevent it.

4.2 Profit Sharing by Using the Shapley Value

Next, we consider to use the Shapley value. The Shapley value has been studied as a profit/cost sharing method [3]. The Shapley value represents the marginal contribution of component service s_i, i.e., to what extent the user's valuation increases by introducing s_i to the existing service $S_i - s_i$.

$$\phi_{s_i} = \sum_{S_i : s_i \in S_i \subset S} \frac{(\#S_i - 1)!(n - \#S_i)!}{n!} \{v(S_i) - v(S_i - s_i)\}$$

Here, $\#S_i$ represents the number of component services included in S_i and $v(\emptyset) = 0$.

The Shapley value satisfies the desirable properties such as Pareto optimality, the null player property, the equal treatment property, and additivity. However,

it causes a problem if we apply it to the profit sharing problem among Web services because the unit of services is not obvious.

Consider the following example. Three component services of s_1,s_2,s_3 exist and the characteristic function is given as $v(s_1) = v(s_2) = v(s_3) = 0$, $v(s_1s_2) = 5$, $v(s_2s_3) = 3$, $v(s_1s_3) = 3$, $v(s_1s_2s_3) = 10$. The Shapley values are calculated as follows. $\phi_{s_1} = 11/3$, $\phi_{s_2} = 11/3$, and $\phi_{s_3} = 8/3$.

Next, consider the case that services providers of s_1 and s_2 do not provide s_1 and s_2 as a component service but only provide a composite service of $[s_1s_2]$. Here, users are not allowed to use the composite services of $[s_1s_3]$ nor $[s_2s_3]$. The characteristic function is given as $v([s_1s_2]) = 5$, $v(c) = 0$, $v([s_1s_2]s_3) = 10$. In this case, the Shapley values are calculated as $\phi_{[s_1s_2]} = 15/2$ and $\phi_{s_3} = 5/2$.

Compared the two cases, s_1 and s_2 can earn $11/3 + 11/3 = 22/3$ if s_1 and s_2 are provided as component services, while they can earn $15/2$ if s_1 and s_2 do not provide s_1 nor s_2 as a component service. The profit in the latter case is larger than that in the former case, which means that users cannot use the composite service of $[s_1s_3]$ nor $[s_2s_3]$. This means that a virtue of service computing that a variety of services are provided and users can arbitrarily combine these services to satisfy their demands is spoiled. The next section examines the conditions that providers choose a strategy of not providing a component service even if they can do it.

5 Analysis of the Levels of Service Provision

This section discusses the profit sharing based on the Shapley values. We draw the conditions in the case including only three component services, and then, we analyzes what factors affect the levels of service provision.

5.1 Case of Thee Component Services

This section discusses the case of three component services available. If s_1,s_2,s_3 are provided by each provider, the Shapley values are calculated as follows.

$s_1 : \frac{1}{6}(v(s_1) + v(s_1) + v(s_1s_2) - v(s_2) + v(s_1s_3) - v(s_3) + v(s_1s_2s_3) - v(s_2s_3) + v(s_1s_2s_3) - v(s_2s_3))$
$s_2 : \frac{1}{6}(v(s_2) + v(s_2) + v(s_1s_2) - v(s_1) + v(s_2s_3) - v(s_3) + v(s_1s_2s_3) - v(s_1s_3) + v(s_1s_2s_3) - v(s_1s_3))$

Therefore, s_1 and s_2 obtain the following profit in total.

$$\frac{1}{6}(v(s_1) + v(s_2) - 2v(s_3) + 2v(s_1s_2) - v(s_2s_3) - v(s_1s_3) + 4v(s_1s_2s_3)) \quad (1)$$

Next, if s_1 and s_2 collude with each other and $[s_1s_2]$ is provided as a component service, the profit share is calculated as follows.

$$\frac{1}{2}(v(s_1s_2) + v(s_1s_2s_3) - v(s_3)) \quad (2)$$

By calculating the difference between the expressions of (1) and (2), we obtain the followings.

$$\frac{1}{6}(v(s_1) + v(s_2) + v(s_3) - v(s_1s_2) - v(s_2s_3) - v(s_1s_3) + v(s_1s_2s_3))$$

Therefore, if the following inequality holds, the composite services of s_1s_3 and s_2s_3 are available for users.

$$v(s_1s_2s_3) + v(s_1) + v(s_2) + v(s_3) > v(s_1s_2) + v(s_2s_3) + v(s_1s_3)$$

Whether component services are provided so that any combinations of these component services are allowed can be affected by the following two strategic behaviors. One is to eliminate the contribution of other composite service (s_1s_3, s_2s_3 in the above example) and increase the own contribution of (s_1s_2) relatively by providing it as a single service. Another is to increase the profit share by providing the component service without any restrictions and increasing the number of share holders.

When the user's valuation considerably increases if the three services of s_1, s_2, s_3 are used as a set, the share of s_1 and s_2 increase by individually having the share. On the other hand, if the user's valuation for $s_1s_2s_3$ is not so larger than s_1s_2, s_1s_3, s_2s_3, s_1 and s_2 can increase their share by preventing each of s_1 and s_2 as a component service.

If the number of the component services becomes more than three, the problem becomes more complicated. This is because the collusion by the three providers as well as the collusion by the two providers may occur. Here, we have a question on what conditions the levels of service provision is spoiled. The next section examines how the number of providers affects the levels of service provision.

5.2 Effects of the Number of Providers

Suppose that a composite service of X including k component services and X is provided as a composite service, i.e., a subset of component services in X are not provided to users. In addition, to make analysis tractable, we assume that the composite services are homogeneous, which means that the characteristic function is symmetric to any component services. Here, we can say that the value of the characteristic function can be determined by the number of component services i included in the composite service. We designate the value per component service as $v_i(v_1 < v_2 < \cdots < v_n)$ in the case that i component services are included.

Here, if component services of X and y exist, the characteristic function can be represented as follows.

$$v(Xy) = (k+1)v_{k+1}, v(X) = kv_k, v(y) = v_1$$

The share of X is given as follows.

$$\frac{1}{2}(kv_k + (k+1)v_{k+1} - v_1)$$

On the other hand, if the component services included in X are provided individually, the profit share of each service can be given by dividing the profit $(k+1)v_{k+1}$ for providing the composite service Xy equally to $(k+1)$ providers because we assumes that the component services are homogeneous. Thus, the increase of the profit share of each component service by providing X as a component service can be calculated as follows.

$$\frac{1}{2}(v_k - \frac{k-1}{k}v_{k+1} - \frac{1}{k}v_1)$$

This expression tells that the increase becomes less than zero if k increases. Service providers can claim that the set of their services is more valuable by colluding each other and preventing users from using any combinations of these services. However, the number of k further increases, the profit has to be shared by the more providers, which results in reducing the profit share increase by the collusion.

6 Toward Problem Solving

To overcome the drawbacks that applying the profit sharing method based on the Shapley values to the service computing domain spoils the sufficient service provision, we propose the direction of dealing with the problem. The idea is to give the same share as that obtained by forming a collusion if the collusion gives a larger amount of profit, which discourage providers to collude with each other.

In the example in section 4.2, we showed that s_1 and s_2 have an incentive to collude with each other and provide s_1s_2 as a component service. Here, if s_1 and s_2 are paid 15/2, they do not have an incentive not to provide s_1 or s_2 as a component service, which enables users to utilize composite services of s_1s_3 or s_2s_3. The profit share of 15/2 between s_1 and s_2 are determined by calculating the Shapley values, although s_1 and s_2 share equally 15/4 each because we assume services are homogeneous. The share of component service s_3 decreases to 5/2. However, its share is larger than zero that is the share in providing s_3 without combining other services.

As mentioned above, if the number of component services is more than three, a various type of collusions may exist. Here, we have a good news that we do not have to examine a case of including quite a many providers because the number of component services becomes large, the providers are less motivated to collude with each other. Carrying out more detained analysis is included in our future work.

So far, we have discussed how to maintain the sufficient level of service provision. However, satisfying only the sufficient level of service provision is not sufficient in the context of incentive. This is because providers may have an incentive to break a service into many of too fine-grained component service as explained in Section 3. For example, a provider of operating dictionary service can break the dictionary service into sub services of having only index-a entries, index-b entries, etc. if it brings the provider larger profit.

A method to solve this problem is to deal with a set of services as a component service if these services are provided by a single provider. This prevents too fine-grained services to be provided.

7 Concluding Remarks

This paper pointed out that neither the adding-up method nor the equal-division method have drawback in sharing the profit among the providers in the service-oriented computing domain. Next, we introduced the concept of a level of service provision, which means to what extent users are allowed to arbitrary combine these services with each other to accomplish their tasks. We showed that the profit sharing method based on the Shapley values, which has been studied in the game theory, cannot attain the sufficient level of service provision. The analysis showed that the level of service provision gets worse if the number of the component services included in the corresponding composite service is small. In addition, we gave a direction to solve this problem.

The discussion shows a possibility of solving the problem of service-oriented computing by introducing the game theory. On the other hands, service-oriented computing might be possible to affect the studies of the Shapley values. So far, the research efforts have been invested in clarifying the axioms of the Shapley values in the game theory and in dealing with the computational complexity of calculating the Shapley values for a large-scale problem in computer science. Here, the flexibility of defining the policy of service utilization causes a new type of collusion. Further investigation about this problem is included in our future work.

Acknowledgments. This research was partially supported by a Grant-in-Aid for Scientific Research (B) (22300052, 2010-2012) and a Grant-in-Aid for Scientific Research (A) (21240014, 2009-2011) from Japan Society for the Promotion of Science (JSPS).

References

1. Ben Hassine, A., Matsubara, S., Ishida, T.: A Constraint-Based Approach to Horizontal Web Service Composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 130–143. Springer, Heidelberg (2006)
2. Cheng, H.K., Tang, Q.C., Zhao, J.L.: Web services and service-oriented application provisioning: An analytical study of application service strategies. IEEE Transactions on Engineering Management 53(4), 520–533 (2006)
3. Shapley, L.S.: A value for n-person games. Annals of Mathematical Studies 28, 307–317 (1953)
4. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using shop2. Journal of Web Semantics 1(4), 377–396 (2004)
5. Zheng, X., Martin, P., Powley, W., Brohman, K.: Applying bargaining game theory to web services negotiation. In: Proceedings of the 2010 IEEE International Conference on Services Computing (SCC 2010), pp. 218–225 (2010)

A Predictive Business Agility Model for Service Oriented Architectures

Mamoun Hirzalla[1,2], Peter Bahrs[2], Jane Cleland-Huang[1], Craig S. Miller[1], and Rob High[2]

[1] School of Computing, DePaul University, 243 S. Wabash Ave, Chicago, IL 60604
{mhirzall,jhuang,cmiller}@cs.depaul.edu
[2] IBM, 11501 Burnet Road, Austin, TX, USA
{mamoun.hirzalla,bahrs,highr}@us.ibm.com

Abstract. Service-Oriented Architecture (SOA) is now considered a mainstream option for delivering solutions that promise business agility benefits. Unfortunately, there is currently no quantitative approach for predicting the expected agility of a SOA system under development. In this paper we present an empirically validated Predicted Business Agility Index (PBAI) which is designed to measure the expected business agility of a SOA deployment. The PBAI is constructed through statistically analyzing the relationship between 150 technical attributes and the attainment of business agility in 39 SOA deployments. 37 of the technical attributes, classified into three areas of architecture, business process management, and impact analysis are determined to be the primary contributors to achieving business agility. The PBAI is evaluated using a leave-one-out cross validation experiment of the SOA projects in our study.

Keywords: Business Agility, SOA, Metrics, Architecture, Impact Analysis.

1 Introduction

The introduction of SOA into the enterprise has resulted in the proliferation of enterprise level solutions that significantly leverage services to solve very large scale problems [1]. As a result, it is now almost impossible to conduct business on the web without using some type of service. According to High et al [2], SOA solutions facilitate increased agility and resilience within an organization. They enable the alignment of information systems with business processes to allow the organization to respond more quickly to changing market conditions. Dove [3] defines business agility as the ability to "manage and apply knowledge effectively, so that an organization has the potential to thrive in a continuously changing and unpredictable business environment." The development of a SOA solution represents a non-trivial investment in human resources, capital and time. It is often undertaken with the expectations that it will position the organization to respond more adeptly to changing market conditions. Unfortunately, SOA projects that do not consider the impact of

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 653–660, 2011.

various architectural, process, and governance-related decisions upon future business agility may fail to achieve these desired benefits [4]. Therefore, we have developed a business agility predictor model, which is designed to evaluate a SOA deployment currently under development in order to (i) predict the extent to which it is likely to achieve business agility in the future, (ii) identify deficiencies, and (iii) suggest corrective measures. One of the benefits of our approach is that it can be used during the planning and development phase by technical IT stakeholders.

The business agility predictor model was constructed in several stages. First, a Business Agility Index (BAI) was developed to measure after-the-fact attainment of business agility in each of the studied SOA deployments. In the second stage of model construction, we developed a Predicted BAI (PBAI) which is designed for use in early stages of a SOA project to predict future business agility based on the degree to which various attributes are present in the project. The PBAI was developed using a version of the Goal-Question-Metric (GQM) approach known as GQM-MEDIA (GQM/MEDEA) [5] to identify technical attributes of the project which were believed to potentially impact business agility. These attributes were later grouped into meaningful categories, and a statistical analysis was performed to construct the PBAI model, and to validate it against the BAI.

In the remainder of this paper, section 2 describes the BAI and how it was constructed and validated, while sections 3 and 4 describe construction, validation of the PBAI model and threats to validity. Finally, in sections 5 and 6, we review related work and provide a summary of our contributions.

2 The Business Agility Index (BAI)

In our previous work [6], we analyzed 5 major SOA practices that were comprised of 35 attributes that had the potential to impact business agility. The 35 attributes were factored out of more than 150 original attributes which were collected through an IBM Academy of Technology virtual conference and through personal interviews with personnel from non-IBM projects. Data collection was driven by 215 questions, of which 80 were designed for purposes of this study and produced the 150 attributes that were considered for the study. A total of 39 projects were ultimately included in the data analysis. Of these, 30 projects were executed by IBM professionals, while 9 were non-IBM projects.

To construct the Business Agility Index (BAI), various experts from IBM with significant SOA experience devised eight equally weighted true/false questions, referred to from now on as the BAI. The BAI index was computed by assigning one point to each question answered positively for a project. The BAI scale therefore ranged from 0 (no indication of business agility) to 8 (strong indication of business agility).

As part of the data collection process, each study participant was responsible for engaging an extensive set of business stakeholders for their project in providing a simple yes/no answer to the more direct question: “Did this project achieve business agility?” They were also responsible for answering the eight BAI questions. Reported results in Table 2 show that 18 of the projects were classified as business agile, while

14 were classified as non business agile. The remaining 7 projects were unclassified. Results show that 12 of the business agile projects achieved BAI scores of 5 or higher, while 6 achieved scores of 4 or lower. Furthermore, 13 of the non-business agile projects achieved BAI scores of 4 or lower. Consequently, for the SOA projects included in our study, 100% of projects receiving high BAI scores were in fact perceived as business agile. In contrast, 70% of projects receiving low BAI scores were classified as non-business agile.

3 Predicted Business Agility Index (PBAI)

Although the BAI is able to largely differentiate between business and non-business agile projects, its usefulness is limited because it requires business expertise and can only be assessed after the fact. In contrast, the PBAI model incorporates a concrete set of factors and attributes that are easily collectable by technical project personnel during early phases of a project, and which have the capability of accurately differentiating between projects which are likely to attain business agility and those which are not.

Building the PBAI therefore involved first identifying a very broad set of 150-candidate SOA attributes that were collected as a result of executing the data collection questions. Factor analysis was later used to identify 37 attributes that accounted for the differences in attained business agility as measured by the BAI. The 37 attributes were grouped into meaningful **composite factors**. This grouping was necessary because the relatively small sample size of 39 projects and the problem of

Table 1. Business Agility Index Questions

1.	As a result of deploying this SOA solution, is your organization able to achieve better business outcomes and respond faster to customer changing *requirements* and demands? *(Yes, No)*
2.	As a result of deploying this SOA solution, is your organization able to address potential ad-hoc business situations? *(Yes, No)*
3.	As a result of deploying this SOA solution, is your organization able to adapt better to dynamic business situations once they were identified? *(Yes, No)*
4.	As a result of deploying this SOA solution, is your organization able to create solutions that address market requests and competitors faster and more efficiently? *(Yes, No)*
5.	As a result of deploying this SOA solution, is your organization able to get a better view of their business conditions and react faster to events that have the potential to cause disruption of their business? (Yes, No)
6.	As a result of deploying this SOA solution, is your organization able to adapt better to changing situations and enabled efficient routing of business needs with minimal interruptions? (Yes, No)
7.	As a result of deploying this SOA solution, is your organization able to view trusted and useful data? (Yes, No)
8.	As a result of deploying this SOA solution, is your organization able is able to create services faster to the marketplace? (Yes, No)

incomplete data made it infeasible to analyze all of the attributes individually. We developed a set of five **composite metrics** described as architecture (SOA), impact analysis (IA), Business Process Modeling (BPM), loose coupling (LC), and governance (Gov), each of which aggregated metric results from its associated individual attributes. For example, the *Governance Metric* evaluated the extent to which a SOA project exhibited governance practices, and was computed by summing the associated individual attributes. In the following section we describe the statistical analysis that evaluates the extent to which each composite factor contributes to attained business agility. These findings were used to build the PBAI model.

Table 2. BAI Assessment of 39 SOA Deployments

	Business Agility Achieved?			
BAI	No	Yes	Unclassified	Total
0	6	0	1	7
1	0	1	2	3
2	1	1	1	3
3	1	3	1	5
4	6	1	2	9
5	0	1	0	1
6	0	5	0	5
7	0	4	0	4
8	0	2	0	2
Total	14	18	7	39

3.1 The Business Agility Predictor Model

The overall sample provided sufficient representation of both business agile and non-business agile SOA deployments: 38% of projects we studied scored below the midpoint of the BAI scale, while the remaining 62% performed above it. The SOA, Impact Analysis, BPM, loose coupling, Governance scores, and the BAI index were computed for each project. The projects that claimed business agility had a mean value for the SOA score 102.19% higher than the projects that did not claim to achieve business agility. A less significant result was reported for BPM, Impact Analysis and Loose Coupling scores with values 48.27%, 56.16% and 9.50% respectively. The Governance factor showed a difference of -6.39% in the mean between projects that achieved business agility versus those that did not.

3.2 Building the Predictor Model

To construct the PBAI we utilized multiple linear regression analysis to investigate the relationship between composite factors (i.e. the independent variables) and the business agility index (i.e. the dependent variable. The descriptive statistics for the identified factors were generated and examined. Projects with missing data were simply dropped from the analysis, reducing the number of analyzed projects from 39 to 32. Regression assumptions such as normality and collinearity were checked and found to be appropriate for the analysis. Correlation analysis between each of the

factors and the dependent variable BAI showed positive correlations for each factor. Correlation values revealed that SOA, IA and BPM are the most significantly correlated factors to achieving business agility in SOA solutions with correlation values of (.67, p = .000), (.52, p=.002), (.50, p=.004) respectively. Loose coupling is also significant (p=.019), however, to a lesser degree than the other factors. SOA Governance, on the other hand, is not significant (p=.305) based on the collected data.

The results of multiple regression showed that our independent variables produced an adjusted R^2 of .64 ($F(3,31) = 19.46$, $p = .000$) for the prediction of achieving business agility. All of the tested predictors turned out to be significant ($p < .05$) except for the Governance Score (p =.309) and Loose Coupling (p=.133)). Based on the results, the regression analysis indicated that SOA, BPM and IA are reliable factors for predicting BAI, while Gov and LC were not. We consequently excluded Gov and LC from our model. Using our model parameters, we write the predicted business agility index PBAI calculation equation as follows:

$$PBAI = 0.65\ SOA + 0.41\ BPM + 0.44\ IA - 1.98$$

where SOA is the SOA Score, BPM is BPM Score and IA is Impact Analysis Score.

3.3 Analysis

A closer look at the individual elements contributing to the SOA Score reveals that inherent support of a given architecture for rules, events, task automation, alerts, monitoring and analytics is essential to achieving the desired business agility improvements. Given that business agility is primarily concerned with the continuous sensing and adapting to new conditions, the significance of this factor seems logical.

Similarly it is not surprising to find that *impact analysis and BPM capabilities* contribute to business agility. Impact analysis ensures that SLAs are monitored proactively and it therefore contributes directly to the business agility goals of responding to ad-hoc changes. Furthermore, the optimized business processes and proper alignment with IT that happen as a result BPM's best practices ensures that business processes are well thought out and not created randomly. BPM attributes such as modeling, monitoring of key performance indicators and the use of rule engines to externalize business rules add a significant amount of flexibility to SOA solutions. It is worth noting that monitoring SLAs or KPIs is not restricted to SOA solutions. Any IT solution can be architected to incorporate aspects of dynamic and predictive impact analysis components as well as BPM best practices.

As previously discussed, our analysis could not conclusively identify loose coupling as a significant factor. However, our results show a positive correlation between loose coupling and the BAI. Given the significance levels reported in regression analysis results, we chose not to include this factor in the overall predictive model. The Governance score predictor was found to be insignificant. The governance score includes a mix of capabilities including establishing and tracking project management and architect roles, advertising and sun-setting services, and tracking requirements and requests for change. One reason that the factor may not have been identified as significant in our model is because all projects that scored

high in impact analysis also tended to score high on the governance factor implying the adoption of strong governance practices. This overlap will be explored further in our future work.

We were not able to draw conclusions concerning reuse due to the missing data with respect to re-use questions. As a result of dropping projects with missing data, it is possible that re-use was not given full consideration as a predictive factor.

4 Validating the PBAI

We conducted a standard leave-one out cross-validation experiment based on the 32 existing projects. The 32 projects were the final set of projects used after excluding projects that did not meet the project selection profile from our original set of 39 projects. In each experimental run, one project was set aside for testing purposes, while the remaining 31 projects were used to repeat the entire regression analysis and construct a new PBAI equation. The PBAI equation was then used to compute a PBAI score for the test project. This process was repeated 32 times, until each project had been tested.

We evaluated the stability of the PBAI equation over the 32 computations. In all of the experiments, the generated PBAI model included the three factors of SOA, BPM, and IA, and excluded Gov, and LC. Furthermore, in 90% of the cases the identified factors remained the same, and differences in coefficient values were found to be minor. In the remaining 10% of cases, the LC factor nudged the BPM factor (i.e. LC factor $p < 0.05$ while BPM factor $p > 0.05$) .

4.1 Threats to Validity

The primary threat to validity in our study arises from the fact that only 39 SOA deployments were included in our study; however it is not trivial to increase the number of projects studied as each SOA project involved many hours of hands-on data collection and a significant investment in time. Nevertheless, the relatively small sample size did impact our ability to explore certain factors such as the re-use factor, for which there was a problem with missing data. Additional threat to validity was introduced through the fact that 30 of the 39 projects are IBM related. However, these projects represented a wide range of complexity, duration, industry participation and scope, and the leave-one-out cross-validation experiment demonstrated that the results were at least applicable across this broad sample of projects.

5 Related Work

Most techniques for measuring business agility are domain specific and are unrelated to SOA deployments. Several authors have argued that the vagueness of the agility concept makes it extremely difficult to measure using regular quantitative methods [7], [8], [9]. Tsourveloudis et al. [7] used fuzzy logic to measure agility in the manufacturing sector. Their approach measures operational characteristics such as

change in quality, versatility, and product variety, rather than measuring the indirect results of agility such as better profits, time to market, or customer satisfaction. The authors associate specific attributes with more general areas of agility infrastructure. Infrastructure agility parameters and their variations are used to compute an overall agility score. In some respects, Tsourveloudis et al's approach is similar to ours, as both methods group attributes or factors into large categories and then evaluate an organization with respect to those larger categories.

Lin et al. [9], [10] also used fuzzy logic by developing the fuzzy agility evaluation (FAE) framework and its associated fuzzy agility index (FAI). The FAE used a survey to collect and analyze agility drivers such as IT integration, competence, team building, technology, quality, change, partnership, market, education and welfare. The framework included steps to analyze and synthesize the answers to the agility drivers and provide associated weightings that are used to establish FAI thresholds and map scores to different agility levels.

While some of these methods are effective for measuring business agility in specific domains or even for general IT solutions, they do not address the problem of measuring business agility in SOA deployments. In contrast, our approach is designed specifically for use with SOA deployments and incorporates a mixture of factors which are general to all IT solutions as well as factors specific to SOA projects.

6 Conclusions and Future Work

This paper has presented a PBAI model for predicting whether a SOA project under development is likely to attain business agility. The model was developed through analyzing data from over 32 successful SOA deployments, and has been shown experimentally to serve as a relatively strong predictor of business agility. The empirical validation reported in this paper substantiates many of the ideas that have previously been anecdotally claimed as best practices, while challenging other broadly accepted ideas. For example, governance is broadly touted as a best practice for achieving business agility, but our analysis did not identify it as a significant factor, r =.187. On the other hand, stronger correlations were reported for SOA, IA, BPM, and LC with r values of .675, .52, .50 and .41 respectively. In future work, we plan to extend the validation steps of the model to include a detailed comparison of BAI and PBAI values to assess the predictability capabilities of the resulting model. Additional work may involve the inclusion of additional dimensions to our BAI and PBAI factors.

Acknowledgments. We thank our IBM colleagues who participated in the discussions while working on the 3rd SOA Best Practices conference. We also thank the many participants who invested many hours in the data collection process.

References

[1] Newcomer, E.: Understanding SOA with Web services. Addison-Wesley, Upper Saddle River NJ (2005)
[2] High, R., Kinder, S., Graham, S.: IBM SOA Foundation: An architectural introduction and overview - version 1.0 (November 2005), http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/#download (accessed: March 27, 2011)
[3] Dove, R.: Response ability: the language, structure, and culture of the agile enterprise. J. Wiley, New York (2001)
[4] Fiammante, M.: Dynamic SOA and BPM: best practices for business process management and SOA agility. IBM Press/Pearson, Upper Saddle River NJ (2010)
[5] Briand, L.C., Morasca, S., Basili, V.R.: An operational process for goal-driven definition of measures. IEEE Transactions on Software Engineering, 1106–1125 (2002)
[6] Hirzalla, M., Bahrs, P., Huang, J., Miller, C., High, R.: An Analysis of Business Agility Indicators in SOA Deployments. In: Int'l Conference on Software Engineering Research and Practice (SERP), Las Vegas, USA (2011)
[7] Tsourveloudis, N.C., Valavanis, K.P.: On the measurement of enterprise agility. Journal of Intelligent and Robotic Systems 33(3), 329–342
[8] Lim, S.L., Ishikawa, F., Platon, E., Cox, K.: Towards Agile Service-oriented Business Systems: A Directive-oriented Pattern Analysis Approach. In: 2008 IEEE International Conference on Services Computing, Honolulu, HI, USA, pp. 231–238 (2008)
[9] Lin, C.T., Chiu, H., Tseng, Y.H.: Agility evaluation using fuzzy logic. International Journal of Production Economics 101(2), 353–368 (2006)
[10] Lin, C.-T., Chiu, H., Chu, P.-Y.: Agility Index in the supply chain. International Journal of Production Economics 100(2), 285–299 (2006)

Personal-Hosting RESTful Web Services for Social Network Based Recommendation

Youliang Zhong, Weiliang Zhao, and Jian Yang

Department of Computing, Macquarie University,
North Ryde, NSW 2109, Australia
{youliang.zhong,weiliang.zhao,jian.yang}@mq.edu.au

Abstract. Recommender systems have been widely used in information filtering. However the existing recommendation methods do not work effectively in the situations when a group of people want to share information and make recommendations within a social network. In this paper we propose a personal-hosting web services architecture *ph-REST* for social network based recommendation, in which every user is represented by a dedicated RESTful web services engine that collaborates with others over a social structure formed by *co-peers with common interests*. The proposed architecture explores the potential of applying service and Cloud computing to personal and social information sharing and assimilation.

Keywords: Personal-hosting web services, Social network based recommendation, Recommendation methods.

1 Introduction

With the surge in the popularity of Web 2.0 technologies, people routinely use social networking, collaboration tools, and wikis to search, accumulate and acquire new knowledge, as well as to share the acquisitions with their friends and colleagues. Recommender systems have been widely accepted in information sharing and assimilation. However the current filtering methods have limitations in many situations where most items have few user ratings, and the users prefer to share their information within peer groups and make decisions as they wish.

Considering a scenario of research literature search, when a researcher wants to expand her collections in a particular area, she will ask her colleagues to give recommendations, and her colleagues may continue with the requests to their social associates and pass the results back to the researcher. This is usually a relay process for getting recommendations through social networks. To follow the human behaviour of social networking, it is necessary to build a truly distributed architecture with corresponding filtering methods by exploiting the 'social association' structure in recommendation process.

Web services [2] would be one of the promising technologies to achieve the above goal. Especially the recently popular RESTful web services architecture

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 661–668, 2011.

[6] has demonstrated its strengths in the applications where the services are consumed by a large number of clients simply through HTTP protocols.

In this paper we propose a personal-hosting web services architecture *ph-REST* with a relay-based recommendation method for social network based recommendation. In the architecture every user is represented by a dedicated RESTful web services engine referred to *REST-engine*, which collaborates with other users through web services provision and consumption. A recommendation process starts with a requesting user's collection of items and associated ratings. Then the user's friends or peers can forward the request to their friends or peers and so forth. All the peers produce and adjust recommendations, and return the recommendation results backwards to the requesting user. The main contributions of the paper are as follows:

- **An architecture of personal-hosting RESTful web services.** This paper proposes a personal-hosting RESTful web services architecture *ph-REST*, in which every user is equipped with a dedicated web services engine that plays both roles of service provider and service consumer. The proposed architecture shows a great potential of applying Service and Cloud computing to social network based recommendation.
- **A relay-based model for social network based recommendation.** This paper introduces a relay-based recommendation model by explicitly utilizing social association, which is based on a dynamically formed social structure. Having recommendations be adjusted by direct and indirect peers in the social structure broadens the range of recommending peers therefore increases the quality of recommendation results.

The rest of the paper is organized as follows. We firstly present a motivating example and the *ph-REST* structure in Section 2, then discuss the social network based recommendation model in Section 3. Prototype and experiments are discussed in Section 4, and the related work is reviewed in Section 5. Concluding remarks are provided in Section 6.

2 Personal-Hosting Web Services Architecture

2.1 A Motivating Example

Fig 1 illustrates how recommendations are produced followed "social association" in a social network by a relay mechanism. In the picture, each user maintains a collection of items and associated ratings. For instance, user *Alex* possesses a list of item "*a, c, e*", with associated ratings "*8, 7, 3*", he also has three friends: *John, Peter, Eddy. Alex* has commonly rated item *c* with *John*, and *e* with *Peter*, but nothing common with *Eddy.* We call an item like *c* or *e* as co-rated-item (CRI), and friends who have CRIs as co-peers. Consequently, *Alex* and *John* are co-peers, and so are *Alex* and *Peter.*

When user *Alex* wants to expand his collection. He sends a request to his co-peers *John* and *Peter*; similarly *John* can forward the request to his co-peer

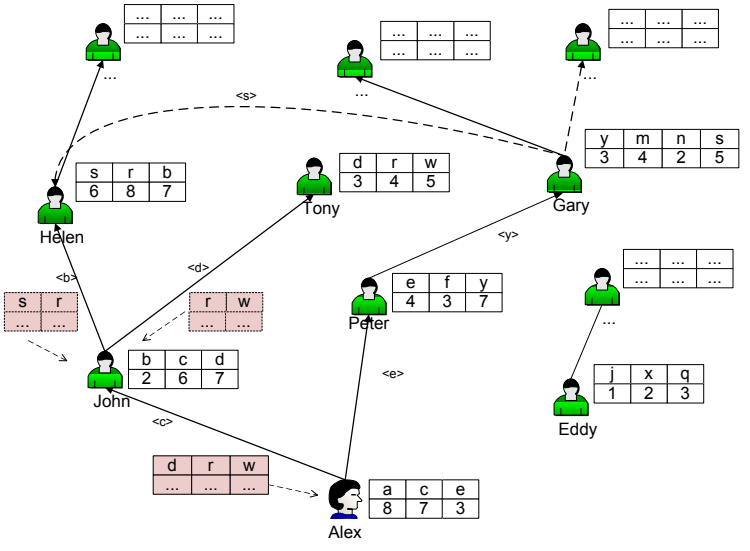


Fig. 1. A motivating example of social recommendation

Helen and *Tony*, and so forth. For a given request, the initial requestor and all its successive co-peers will form a social structure, and every user in the structure will produce recommendations on his own. Finally, after getting responses from his co-peers *John* and *Peter*, *Alex* aggregates and filters the recommended items to create a final recommendation list for himself.

2.2 ph-REST: Personal-Hosting RESTful Web Services

A personal-hosting web services architecture *ph-REST* is proposed, in which every user is represented by a dedicated web services engine (REST-engine). The communication between REST-engines follows a Producer-Consumer pattern [14]. Figure 2 shows the components of a REST-engine. Of each engine, there are *HTTP Server* and *HTTP Client* for communicating with other engines, *REST Router* handles recommendation requests and responses. In a *Recommendation application*, *Relay Manager* maintains co-peer relationship and communicates with recommendation components: *Prediction* and *Filtering*, which work with *Resource Manager* through to various data sources mapped to actual data in an embedded database.

3 Social Network Based Recommendation

3.1 Notation

From the motivating example, we introduce a co-peer graph CPG(q, V, A) as a labeled directed acyclic graph, where q is a recommendation request, V a set of vertices and A a set of directed arcs over V, such that,

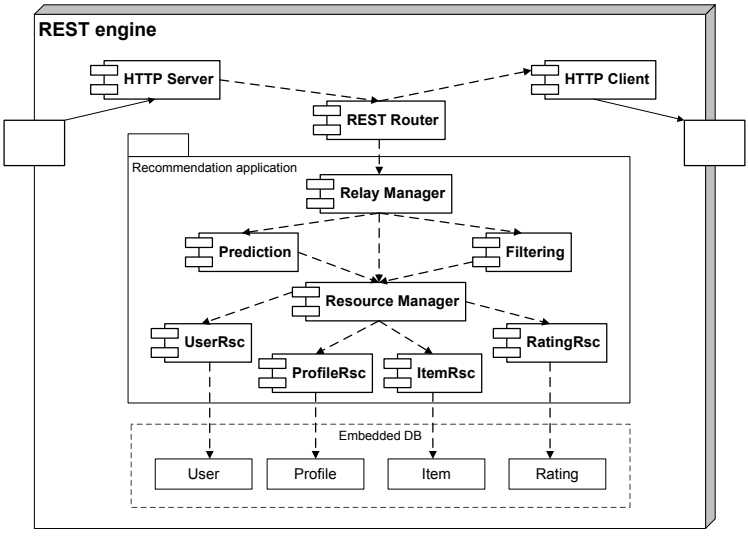


Fig. 2. The components of REST-engine

- For any $v \in V$, it represents a user that possesses a pair of tuples of items and ratings. $T(v) = \{t_i \mid i = 1..n\}$ is a set of items possessed by v, and $R(v) = \{v(t_i) \mid i = 1..n\}$ the associated ratings over $T(v)$.
- q is a recommendation request made by a root node $v_0 \in V$. The request q includes a user's preferences such as items and ratings. In Fig (1), *Alex* is the root node, he sends out a recommendation request with his possession of items (*a, c, e*) and ratings (*8, 7, 3*).
- For a pair of vertices $u, v \in V$, the set intersection of $T(u) \cap T(v)$ is denoted as C_u^v. If $C_u^v \neq \emptyset$, then C_u^v is referred to co-rated-items (CRI) between v and u, and u and v become co-peers. Consequently, a directed arc $a(u, v) \in A$ from u to v is established in the graph.
- For $v \in V$, a co-peer u that sends a request to v is called an inbound co-peer of v, or $Icp(v) = \{u\}$. And those co-peers that receive a request from v are called outbound co-peers of v, denoted as $Ocp(v)$. $Ocp(v) \cap Icp(v) = \emptyset$. Furthermore, of the outbound co-peers of v, a group of them that commonly rate on a particular item $t_i (t_i \notin T(v))$ is denoted as $Ocp_i(v)$.
- For $v \in V$, if $Ocp(v) = \emptyset$, then v is called a leaf peer in the graph. The set of all leaf peers is referred to as L.

3.2 Prediction Formulas

Every user in a co-peer graph may carry out three tasks when participating a recommendation process: (1) makes predictive ratings of potential items, which are rated by a user but not its inbound-co-peer. (2) aggregates the recommendations replied from its outbound co-peers, (3) selects highly recommended items based on predictive ratings or/and other criteria for its inbound co-peer.

Considering a pair of co-peers u and v, with co-rated-items C_u^v, and u is the inbound co-peer of v, or v is an outbound co-peer of u. For an item $t_i \in P_u^v$, where $P_u^v = T(v) \backslash T(u)$ indicating the "potential items" from v for u. We want to predict a rating for the item t_i, denoted as $r_u^v(t_i)$. Let us have $r_u^v(t_i) = v(t_i) + b$, where $v(t_i)$ is the rating made by v, and b an adjustment constant. According to Minimum Mean Square Error principle, we have $E = \sum_{t_j \in C_u^v} (v(t_j) + b - u(t_j))^2$, and the predictive $r_u^v(t_i)$ can be obtained by minimizing E. Formally,

$$\begin{gathered} r_u^v(t_i) = v(t_i) + b, \\ b = \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} (u(t_j) - v(t_j)), \\ so,\ r_u^v(t_i) = \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} (v(t_i) - v(t_j)) + \overline{R}_v(u), \\ where\ \overline{R}_v(u) = \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} u(t_j). \end{gathered} \tag{1}$$

From the standing point of an inbound co-peer say user u, it may associate with multiple outbound co-peers which have made individual predictions on a same potential item say t_i. The user then needs to aggregate all the predictive ratings of t_i. To this end, we use the following formula (2) for simplicity of representation, where $r_u(t_i)$ is the aggregated rating of t_i for u.

$$\begin{gathered} r_u(t_i) = \frac{1}{|Ocp_i(u)|} \sum_{v \in Ocp_i(u)} r_u^v(t_i). \\ T(u) = T(u) \cup T^r(u),\ where\ T^r(u) = \cup_{v \in Ocp(u)} P_u^v, \\ R(u) = R(u) \cup \{r_u(t_j) \mid t_j \in T^r(u)\}. \end{gathered} \tag{2}$$

3.3 Web Services Operations

In *ph-REST*, every user is represented by a REST-engine that provides three public operations: *init_request*, *relay_request* and *receive_recommendation*, and two internal methods: *wait_and_make* and *make_recommendation*. Both *init_request* and *relay_request* are used for receiving and relaying recommendation requests, and *receive_recommendation* for propagating recommendation responses. *wait_and_make* is designed to control recommendation process, and lastly *make_recommendation* to actually produce recommendations.

Through a full cycle of a replaying and adjusting process, a root node v_0 in a co-peer graph will get a set of recommended items and associated ratings $< T^r(v_0),\ R^r(v_0) >$, which will be aggregated and filtered by the root node.

$$T^r(v_0) = \cup_{u,v \in V}^{v \in Ocp(u)} \widetilde{P}_u^v,\ and\ R^r(v_0) = \{r_{v_0}(t_j) \mid t_j \in T^r(v_0)\}. \tag{3}$$

4 Prototype and Evaluation

4.1 Prototype of ph-REST

A prototype system of *ph-REST* has been developed by using *Restlet*, one of the most popular RESTful web services framework. The experiment data were

taken from ISI Web of Knowledge database [11], consisting of 500 articles, each bibliographic item of an article included title, author, publish year, abstract and keywords. 30 users were created with totally 635 ratings.

Besides the general considerations of web services deployment, several special issues have been addressed in the prototype system. Particularly, REST-engines need to control recommendation propagation under conditions. To this end, each request is relayed with control information including the endorsed co-peers, the maximum relay depth, and the longest period of waiting time. Doing so, the system ensures that the recommendation requests and responses are propagated through a well-controlled and tree-structured service chain.

4.2 Performance of Recommendation Model

To measure the effectiveness of recommendation results, we adopt commonly used NMAE [7] and Coverage [3] metrics, as well as our own defined metric peMAE as follows (formula 4, 5 and 6). In these formulas, the superscript "e" stands for items or ratings originally made by a user, $r_v(.)$ for the predictive ratings made by co-peers. $T^r(v)$ represents the recommended items and $T_l^r(v)$ the highly relevant items from $T^r(v)$.

$$NMAE = \frac{1}{|V|} \sum_{v \in V} \frac{M_v}{R^e_{max}(v) - R^e_{min}(v)}, \; M_v = \frac{1}{|T^e(v)|} \sum_{t_j \in T^e(v)} |r_v(t_j) - v^e(t_j)|. \quad (4)$$

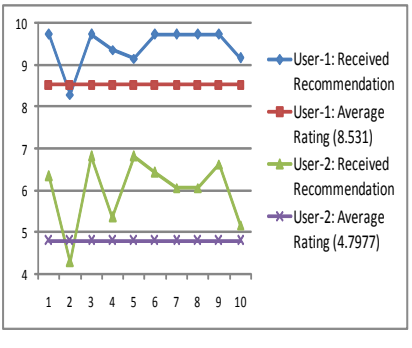
$$Coverage = \frac{1}{|V|} \sum_{v \in V} \frac{|T_l^r(v)|}{|T^r(v)|}. \quad (5)$$

$$peMAE_v = \frac{1}{|T^r(v)|} \sum_{t_j \in T^r(v)} r_v(t_j) - \frac{1}{|T^e(v)|} \sum_{t_i \in T^e(v)} v^e(t_i). \quad (6)$$

Figure 3 shows our experiment results. The left table lists a set of NMAE, Coverage (%) and peMAE values. The average NMAE was about 0.2 and average peMAE marginally over 0.1, that indicates the recommendation results are fairly accurate. The average Coverage (%) was about 60%, that actually reflects the

User	NMAE	Coverage (%)	peMAE
2	0.3447	0.621	0.2631
3	0.1629	0.457	0.0752
4	0.2533	0.7547	0.1078
...			
28	0.3882	0.5235	0.2148
29	0.388	0.5023	0.2622
30	0.184	0.4742	0.0996
AVG	**0.1994**	**0.5922**	**0.1143**

a-Table: NMAE-Coverage-peMAE



b-Figure: Personalized recommendation

Fig. 3. Performance of recommendation relay model

nature of recommendation in social networks where novelty and serendipity usually get much more concerns than content relevance [9,10].

To exemplify how the proposed method achieves personalized recommendation, a set of commonly recommended items were selected from the recommendation results for two users. The average rating of user-1 was 8.5310 and that of user-2 was 4.7977. The right part of Figure 3 shows that these two users with different rating styles received individual recommendations that matched their personal modus operandi, even if the recommended items were the same.

5 Related Work

Generally speaking, two types of filtering approaches are widely used in existing recommender systems: content-based filtering (CN) and collaborative filtering (CF) [1]. While CN recommends items to a user based on the similarities between potential items and existing ones rated by the user [3,4], CF based on the ratings assigned by other users with "similar taste" [15,17]. Our prediction formulas leverage both *Slope One* method [12] and peer relationship so that recommendations can be relayed through the co-peers in social networks.

Few researches on recommendation methods deploy web services [19,20,16], all of them followed a centralized filtering approach. In contrast, our approach utilizes personal-hosting web services so that recommendations are produced and adjusted by every user in a social network.

While most researches on distributed recommendation focus on complexity, scalability and privacy-protection, few ones emphasize on distributed data sources or computation mechanism [18,5,8,13]. Nearly all these researches collect raw data from users, and may execute processing by using a global dataset. Our model does not rely on a global dataset nor a centralized process. Equipped with a personal-hosting web services engine, every user maintains its collection of data and produces recommendations on its own.

6 Conclusion

In this paper we propose a novel architecture *ph-REST* for social network based recommendation, in which every user is represented by a dedicated personal-hosting RESTful web services engine. The *ph-REST* architecture, the web services deployment and the prediction algorithms are discussed in the paper. The proposed architecture shows a great potential of using Service and Cloud computing in social network based recommendation.

Co-rated items and co-peer relationship are very basic relationships in social networking, there should be more meaningful factors that affect recommendation results, such as the depth of social relationship, the size of co-peers, and the centrality of peers. Future work can take these options into account during recommendation process, as well as the qualitative aspects of social networks such as trust and credibility.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the State-of-the-Art and possible extensions. IEEE Trans. on Knowl. and Data Eng. 17(6), 734–749 (2005)
2. Alonso, G.: Web services: concepts, architectures and applications. Springer, Heidelberg (2004)
3. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press, New York (1999)
4. Belkin, N.J., Croft, W.B.: Information filtering and information retrieval: two sides of the same coin? Communications of the ACM 35(12), 29–38 (1992)
5. Berkovsky, S., Busetta, P., Eytani, Y., Kuflik, T., Ricci, F.: Collaborative Filtering over Distributed Environment. In: DASUM Workshop, Citeseer (2005)
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine (2000)
7. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A Constant Time Collaborative Filtering Algorithm. Information Retrieval 4(2), 133–151 (2001)
8. Gong, S.J., Ye, H.W., Su, P.: A Peer-to-Peer based distributed collaborative filtering architecture. In: International Joint Conference On Artificial Intelligence, pp. 305–307 (2009)
9. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS) 22(1), 5–53 (2004)
10. Iaquinta, L., de Gemmis, M., Lops, P., Semeraro, G., Filannino, M., Molino, P.: Introducing serendipity in a content-based recommender system. In: International Conference on Hybrid Intelligent Systems, pp. 168–173. IEEE (2008)
11. ISI-WoK (2010), `http://wokinfo.com/`
12. Lemire, D., Maclachlan, A.: Slope one predictors for online Rating-Based collaborative filtering. Society for Industrial Mathematics (2005)
13. Liu, Z., Qu, W., Li, H., Xie, C.: A hybrid collaborative filtering recommendation mechanism for p2p networks. Future Gener. Comput. Syst. 26, 1409–1417 (2010)
14. Milanovic, N.: Service engineering design patterns. In: International Workshop Service-Oriented System Engineering, pp. 19–26. IEEE (2006)
15. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: an open architecture for collaborative filtering of netnews. In: ACM Conference on Computer Supported Cooperative Work, pp. 175–186. ACM (1994)
16. Sen, S., Geyer, W., Muller, M., Moore, M., Brownholtz, B., Wilcox, E., Millen, D.R.: FeedMe: a collaborative alert filtering system. In: Anniversary Conference on Computer Supported Cooperative Work, pp. 89–98. ACM (2006)
17. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating word-of-mouth. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 210–217. ACM Press/Addison-Wesley Publishing Co. (1995)
18. Tveit, A.: Peer-to-peer based recommendations for mobile commerce. In: International Workshop on Mobile Commerce, pp. 26–29. ACM, Rome (2001)
19. Xu, J., Zhang, L.J., Lu, H., Li, Y.: The development and prospect of personalized TV program recommendation systems. In: International Symposium on Multimedia Software Engineering, pp. 82–89. IEEE (2002)
20. Zadel, M., Fujinaga, I.: Web Services for Music Information Retrieval (2004)

Work as a Service

Daniel V. Oppenheim, Lav R. Varshney, and Yi-Min Chee

IBM Thomas J. Watson Research Center, Hawthorne NY 10532, USA
{music,lrvarshn,ymchee}@us.ibm.com

Abstract. Improving work within and among enterprises is of pressing importance. We take a services-oriented view of both doing and coordinating work by treating *work as a service*. We discuss how large work engagements can be decomposed into a set of smaller interconnected service requests and conversely how larger engagements can be built up from smaller ones. Encapsulating units of work into service requests enables assignment to any organization qualified to service the work, and naturally lends itself to ongoing optimization of the overall engagement.

A service request contains two distinct parts: coordination information for coordinating work and payload information for doing work. Coordination information deals with business concerns such as risk, cost, schedule, and value co-creation. On the other hand, payload information defines the deliverables and provides what is needed to do the work, such as designs or use-cases. This general two-part decomposition leads to a paradigm of work as a two-way information flow between service systems, rather than as a business process that needs to be implemented or integrated between two organizations.

Treating work as information flow allows us to leverage extant understanding of information systems and facilitates information technology support for work using mainstream service-oriented architectures (SOA). Significant benefits from this approach include agility in setting up large engagements to be carried out by distributed organizations, visibility into operations without violating providers' privacy or requiring changes to internal processes, responsiveness to unpredictability and change, and ongoing optimizations over competing business objectives.

Keywords: Work, encapsulation, service, decoupling, information flow.

1 Introduction

Differences among labor pools globally, rapid proliferation of capacious information technology infrastructures, and increased churn due to a millennial generation that is project-based rather than jobs-based [6] has disrupted the nature of work in many institutions, causing increased decentralization of workforces and increased leverage of communities, networks, and ecosystems of people and of firms to do work. These business, technological, and social trends have intensified interest in general ways of structuring the coordination and doing of work.

The fundamental problem of doing work is to transform inputs into outputs to meet specified requirements. For human tasks, work systems can be individuals

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 669–678, 2011.

or groups that may be distributed within or among organizations. But most work required by businesses is complex. The problem then becomes how to translate a business need, perhaps expressed as a service request, into an optimal decomposition of units of work and how to optimally *coordinate* its execution.

The basic problem of coordinating work is to decompose a service request into units that can be assigned to a set of work systems, provide the necessary inputs and requirements to the work systems, and aggregate their outputs, while continuously responding to changing conditions. Often there are dependencies among work assigned to different work systems. The work systems may have conflicting local objectives and may perform work with differing costs, schedules, and reliabilities. Optimal coordination must take these factors into account.

In this paper we treat *work as a service* (WaaS). Doing of work is encapsulated as a service request and coordination of work involves routing service requests to work systems. Within the WaaS paradigm, large work engagements can be decomposed into a set of smaller interconnected service requests and conversely larger work engagements can be built up from smaller ones.

An encapsulated service request contains two distinct parts: coordination information for coordinating work and payload information for doing work. Coordination information deals with business concerns such as risk, cost, schedule, and value. Payload information defines deliverables and provides what is needed to do the work, such as designs or use-cases. This general two-part decomposition leads to a paradigm of work as a two-way information flow between work systems, rather than as a business process that needs to be implemented or integrated between two organizations.

Encapsulation eliminates search inefficiencies on inputs, requirements, and formats for work systems. But more importantly, it enables assignment of work requests to any qualified work system, leading naturally to ongoing optimization of the overall engagement in response to unpredictable system dynamics. Coordination becomes a problem of dynamically routing information flow.

By treating work as information flow, several patterns of and organizational structures for doing work can be treated in a common framework.

Since work is encapsulated as service requests, mainstream service-oriented architectures (SOA) can be used to provide information technology support.

As demonstrated in the sequel, significant benefits from this approach include agility in setting up large engagements to be carried out by distributed work systems, visibility into operations without violating providers' privacy or requiring changes to internal processes, responsiveness to unpredictability and change, and ongoing optimizations over competing system-level business objectives.

2 The Changing Nature of Work and Workforce

The combined force of the trends described earlier has been the emergence of new models of work including: globally dispersed teams in firms [18], outsourcing, crowdsourcing [20], information factories [8], virtual enterprises [13], cross-enterprise collaborations [17], open source, social production [5], and asset reuse [2].

With these new models, there is greater division of labor and workforce specialization, but traditional coordination mechanisms such as mutual adjustment through informal communication [14] are no longer effective [11]. The trade-off between specialization benefits and coordination costs are well known [4], but formal mechanisms may reduce these costs without reducing benefits from specialization. The goal of the WaaS paradigm is precisely this.

As will be evident, encapsulation of work makes it procedurally equivalent to plug in any work system, whether a crowd, partner organization or combination of several work systems. This is in contrast to business process management (BPM) approaches, where recombining the doing of work requires a new business process to connect the pieces together, a provably complex undertaking [15]. Further BPM models do not lend themselves to many optimizations [25], whereas the information flow paradigm herein readily supports optimization.

Notwithstanding, the work for a single encapsulated work request may be carried out using BPM. Further, it is possible to use fulfillment of work requests as signals to transition between states in business process models.

3 Work as a Service (WaaS) Encapsulation

In this section, we describe the essential aspects of the WaaS encapsulation and the resultant information flow paradigm.

As depicted schematically in Fig. 1(a), a work engagement consists of essentially three parts:

1. A *requestor*, which is a service system that requests work to be done, provides inputs, and specifies the requirements.
2. A *provider*, which is a service system charged with fulfilling the work request to meet requirements.
3. An *encapsulated service request*, which captures the interaction between the requestor and provider, the two-way information flow among them.

There are three aspects of a work task that must be established between the requestor and the provider. First, a so-called service level agreement (SLA) must be reached, which specifies business-level properties, such as cost and schedule. Second, as work is ongoing, there may be monitoring by the requestor of partial results and checkpoints achieved by the provider, so as to have appropriate visibility. Third, at the conclusion of a work engagement, the provider sends deliverables to the requestor who either confirms or rejects the delivered work. Note that all three involve two-way flows of information.

These two-way communications that arise should all be captured in the encapsulated work request. We define a general two-part decomposition of work into business concerns and domain concerns. These two parts are called *coordination information* and *payload information*, respectively, and are depicted schematically in Fig. 1(b). Coordination mechanisms can restrict attention to the former part whereas work systems can restrict attention to the latter.

(a) (b)

Fig. 1. (a) Work as an encapsulated service request with requestor R and provider P. (b) The encapsulated request partitioned into coordination and payload information.

In detailing the contents of coordination and payload information, it is easiest to first consider an *atomic service request*: an encapsulation of a unit of work so small that it cannot be broken into pieces. We will later see how to combine and recombine atomic service requests into *molecular service requests*.

3.1 Coordination Information

What information about an atomic service request is needed for coordination? Since the goal of coordination is to satisfy concerns, we enumerate several possible business concerns that arise in work. Note that coordination may be done by the requestors and providers themselves or by an external agent.

A first consideration is schedule: how long will it take for an atomic piece of work to be done. Note that this varies across different work systems and is also potentially stochastic. A second consideration is cost: how much money will the provider charge the requestor to do work and whether there are bonuses or penalties associated with speed or quality [3]. Although the encapsulation formalism is eminently amenable to outcomes-based pricing [21] rather than effort-based pricing, the cost may still have some variability as well. A third consideration is quality: how good will the deliverable be with respect to the requirements. One way to certify the quality of a work system is through the use of CMMI level—higher levels imply more stringent process and quality control.

Each of these concerns can be captured as a mapping from the Cartesian product of the possible set of work systems and the possible set of work tasks to the space of random variables that represent time, cost, or quality. As milestones are reached and partial results are achieved, the probability distributions can be updated with new information that is furnished by the provider. Exogenous perturbations to the system such as natural disasters might also change these distributions, as can changes in requirements imposed by the requestor.

In reality, schedule, cost, and quality are very much intertwined. For example, loosening schedules may reduce costs. These competing business concerns can be balanced through the notion of *value*, which is what should be optimized. The value-dominant logic brings this point out even further [23], and indeed we feel that the encapsulated service request is the seat of value co-creation.

For molecular service requests, coordination information needs to also contain the interdependencies among its atomic constituents. This includes not only things like the fact that one piece of work needs to be done before another, but also inertia effects and other factors, cf. [26].

3.2 Payload Information

We now ask what information is needed by a work system to do work. Broadly, payload information should include the inputs that are to be transformed into outputs and the requirements that specify what is to be done. This should be the minimal sufficient information for doing work.

More specifically, for software development, payload information may include APIs, architectural diagrams, and requirements documents; for engine design, it may include specifications of mechanical, hydraulic, and electrical interfaces, performance requirements and CAD language. The WaaS encapsulation is designed to be general, supporting the needs of any specific domain.

As the lifecycle of the work request proceeds, payload information is updated based on partial results and milestones. Technological developments that impact the doing of work, or changes in requirements would can evolve payload information. Fixes for errors made in execution may also be incorporated.

3.3 Information Flow

The WaaS paradigm may be interpreted as an information flow description. One can think of the encapsulated service request as a multidimensional variable that captures the current state of things, including the value being generated. As things happen, information flows to the service request for it to be updated. Updates to both payload and coordination information happen continuously, capturing both business and domain concerns.

Coordination mechanisms can also be thought of in informational terms as routing. Essentially, coordination involves connecting requestor and provider together to interact through an encapsulated service request. Moreover, as in Sec. 4, large work engagements can be constructed by routing several service requests within a work ecosystem. Coordination must then consider governance issues such as accountability, responsibility, and decision-making rights. There may be different organizational structures for coordination. See Sec. 5.

4 Patterns and Structures

Now we discuss the structural building blocks that enable composition and decomposition of encapsulated service requests to form large work engagements. Several canonical patterns emerge, which are tied to various organizational structures that arise in businesses; cf. Malone's notions of flow, sharing, and fit [12, p. 140]. This demonstrates that the WaaS paradigm applies to its targeted problem space of complex work within or between organizations.

First consider delegating work from one work system to another, as in Fig. 2. This may be done, e.g. if the original provider is overloaded. The original provider becomes a requestor (delegator) for the downstream provider. Payload information and the interdependency portion of coordination information are copied

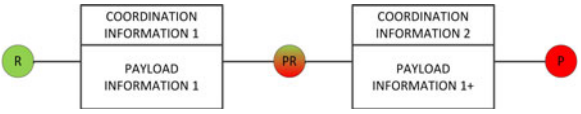


Fig. 2. Delegation of work by re-routing an encapsulated service request. The original provider becomes a requestor for the downstream provider.

essentially unchanged to the new re-routed service request, perhaps adding information useful to the new provider. The remaining coordination information, however, is written anew to capture the business concerns of the delegator and to hide the business concerns of the original requestor, which are not of direct relevance to the new provider. The delegator remains accountable and responsible to the original requestor, but the new provider is only responsible and accountable to the delegator. Note that the original request may specify that when delegated some coordination information must also flow downstream.

Another possible pattern of work is to tear a molecular service request into pieces and re-route them to several producers. Payload information is partitioned into (possibly overlapping) pieces with minimal sufficient information required to do the newly reconstituted work. The interdependency portion of the coordination information is also partitioned, and remaining coordination information is written anew. Recursive hierarchical tearing can also be done.

Tearing should be done for specialization gains, or when work systems can be leveraged in parallel. For example, if a general service provider has several sourcing channels such as a crowd and a factory, different pieces might be routed to different places. Note that tearing requires PR to monitor and eventually integrate or aggregate the completed work so as to be able to respond to R.

A third pattern is to merge several service requests (from one or more requestors) into one. The payload information of the merged request is simply the union of the payloads of the requests being merged. Coordination interdependencies must be combined with any new interdependencies that arise. Remaining coordination information is written afresh, typically meeting the minimum specifications of the original requests. Hierarchical merging can also occur.

Economies of scale are a prime motivator for merging. For example consider several requests to perform environmental testing for electronics where a single cold room could be used simultaneously for all the requests.

Another kind of re-routing that can arise is to withdraw a service request from one provider and assign it to another. This could happen if changing conditions prevent the original provider from completing the service request.

The basic operations described above can be used in combination to generate other structures. As an example, consider a coordination hub [16], a centralized authority charged with coordinating work to derive maximal value, by taking work from several requesters, tearing and merging, and then delegating to several providers, while responding to changing conditions. A hub can be thought of as

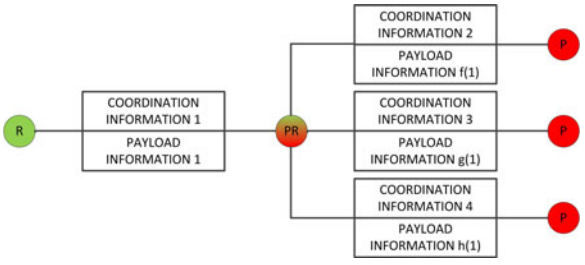


Fig. 3. Tearing and delegating work by re-routing encapsulated service requests. The original provider aggregates and becomes a requestor for downstream providers.

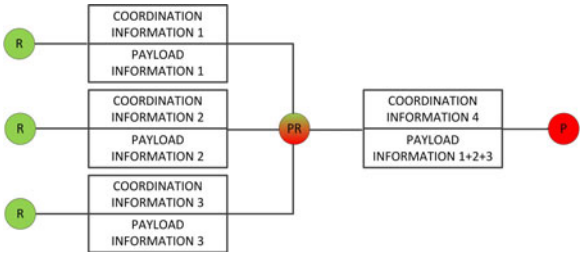


Fig. 4. Merging and delegating work by re-routing encapsulated service requests. The original provider becomes a requestor for the downstream provider.

a kind of intermediate delegator. Cross-enterprise collaboration, when several organizations partner to do work, may take the form of a hub [3], as may robust supply chain collaboration [19].

Formal proof aside, it should be clear that arbitrary topologies can be constructed using the basic building blocks and operations we have defined.

5 Coordination and Governance

We have discussed how service requests can be combined and recombined in various patterns, but it is still unclear how to initially make the plan for doing so or in response to change. That is the purpose of a coordination mechanism.

Initial coordination involves appropriately tearing and merging requests, and then determining routes for work to assign it to qualified producers; unlike communication networks the source and destination are not pre-specified.

Over time as more information becomes available, the work plan may need to be modified and requests re-routed, whether due to environmental events, or updates in the work lifecycle itself or in interdependent tasks and systems.

Coordination mechanisms may be implemented with an automatic program in SOA built on a protocol like WS-coordination, a human program manager or program executive, or a governance council in cross-enterprise collaboration.

Any of these, however, require both access to the coordination information in the encapsulated work requests and the ability to make re-routing decisions, which are matters of governance and organization. By requiring coordination information to be freshly written, the WaaS paradigm naturally limits information to the requester and provider who clearly need it. But when required, governance policies may provide a window to other work systems.

Many possible visibility and governance policies exist, as in Fig. 5. In a centralized hub with complete visibility, responsibility, and decision-making authority [17], a globally optimal coordination scheme can be used [3]. If there is hierarchical authority, as in a globally integrated enterprise [18], visibility and decision-making authority is restricted to one level depth in the tree and coordination mechanisms must respect this. In a fully decentralized governance structure, each pair of service systems is responsible for their own coordination.

Due to differing visibility and authority for re-routing, these various forms of coordination have different abilities to react when conditions change.

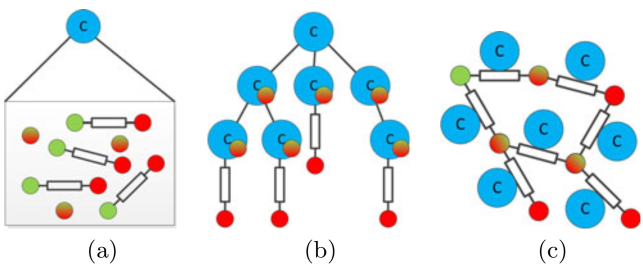


Fig. 5. Coordination, with coordinators C. (a) central coordinator with full visibility and authority into all encapsulated service requests. (b) globally integrated enterprise with hierarchical visibility and authority. (c) decentralized organization with localized visibility and authority.

6 Agility, Optimization, and Innovation

Having discussed the doing of work and the coordination of work within the WaaS framework, we discuss some beneficial attributes of this paradigm.

Due to the block-building nature of WaaS, there is agility in setting up large engagements involving work systems that may be globally distributed within or across enterprises. Since service requests contain sufficient information for work, any admissible work system of any kind, whether a crowd or a virtual enterprise, can be plugged in as a provider. Due to encapsulation, how work is done matters little; only what work is done and the collaboration induced between the requestor and provider. As such, business processes need not be integrated nor internal processes changed. Moreover, agility extends to response to change; reconfiguration by re-routing is as easy as initial setup. The recombinant nature of the encapsulated service requests provides exponential flexibility.

A second thing to note about the WaaS paradigm is that visibility is provided into operations without violating providers' privacy. Since the payload information in the encapsulated service request is the *minimal sufficient statistic* needed to do work, by the data processing inequality in information theory [9], it minimizes information leakage. When service requests are delegated, coordination information is not passed to the new provider, again preserving privacy.

Within the WaaS paradigm, optimizing value becomes a partitioning and routing question, but with destination also subject to choice. In particular, determining how to restructure and then re-route service requests is rather similar to the routing problem faced by packet-switched communication networks like the internet. Centralized hub optimizations are essentially equivalent to nonlinear multicommodity flow problems with a further optimization for balancing destinations, cf. [3], for which optimal flows can be found in polynomial time [7]. When performing distributed coordination, as in Fig. 5(c), efficient routing algorithms developed for internet protocols can be adapted.

When BPM approaches are used, optimal coordination becomes a scheduling problem rather than a routing problem; optimal scheduling problems are often NP-hard [22]. Although routing and scheduling are rather similar, the computational complexity of finding an optimal solution can be different.

When design complexity is low, as with small loosely coupled service requests, innovation is easier due to increased opportunities to experiment [1].

7 Concluding Remarks

We presented a new way for describing work as an information flow and then defined the underlying formalisms that enable the decomposition of requests into fine-grained units that can be coordinated and optimized over competing business, customer, provider, and resource objectives. We further demonstrated how this model generalizes over disparate models of work and can be utilized to support different patterns of organization, business, and governance.

Our approach in this paper has been to describe the basic structural elements and decompositions in the WaaS paradigm. Moving forward, detailed study of optimal coordination mechanisms that really make WaaS go is necessary. The role of uncertainty should take greater prominence [24]; after all, "Uncertainty is what typifies projects. It's the nature of the beast" [10].

One of the biggest revolutions in the evolution of multi-cellular organisms occurred when neurons emerged. Before neurons, cells had to be very close to each other to coordinate their functions. After neurons, cells could communicate from a distance and ongoing two-way flow of information became central to complex life. This allowed cells to be rearranged and assigned different functions in a wide variety of life forms. The information flow paradigm for work developed herein may similarly allow an expansion in the variety of economic and organizational forms that are then able to efficiently fill a wide variety of niches.

References

1. Auerswald, P., Kauffman, S., Lobo, J., Shell, K.: The production recipes approach to modeling technological innovation: An application to learning by doing. J. Econ. Dyn. Control 24, 389–450 (2000)
2. Bacon, D.F., Bokelberg, E., Chen, Y., Kash, I.A., Parkes, D.C., Rao, M., Sridharan, M.: Software economies. In: Proc. FSE/SDP Workshop Future Softw. Eng. Research (FoSER), pp. 7–12 (2010)
3. Bagheri, S., Oppenheim, D.V.: Optimizing cross enterprise collaboration using a coordination hub. In: Proc. SRII 2011 Global Conf. (2011)
4. Becker, G.S., Murphy, K.M.: The division of labor, coordination costs, and knowledge. Quart. J. Econ. 107, 1137–1160 (1992)
5. Benkler, Y.: The Wealth of Networks. Yale University Press, New Haven (2006)
6. Bollier, D.: The Future of Work. Aspen Inst., Washington (2011)
7. Cantor, D.G., Gerla, M.: Optimal routing in a packet-switched computer network. Comput. C-23, 1062–1069 (1974)
8. Chaar, J.K., et al.: Work packet delegation in a software factory, Patent Application Publication US 2010/0031226 A1 (2010)
9. Cover, T.M., Thomas, J.A.: Elements of Information Theory. John Wiley & Sons, New York (1991)
10. Goldratt, E.M.: Critical Chain. North River Press (1997)
11. Gumm, D.C.: Distribution dimensions in software development projects: A taxonomy. IEEE Softw. 23, 45–51 (2006)
12. Malone, T.W.: The Future of Work. Harvard Business School Press (2004)
13. Mehandjiev, N., Grefen, P.: Dynamic Business Process Formation for Instant Virtual Enterprises. Springer, London (2010)
14. Mintzberg, H.: Mintzberg on Management. Free Press, New York (1989)
15. Norta, A.H.: Exploring Dynamic Inter-Organizational Business Process Collaboration. Ph.D. thesis, TU-Eindhoven (2007)
16. Oppenheim, D., Bagheri, S., Ratakonda, K., Chee, Y.M.: Coordinating Distributed Operations. In: Maximilien, E.M., Rossi, G., Yuan, S.-T., Ludwig, H., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6568, pp. 213–224. Springer, Heidelberg (2011)
17. Oppenheim, D.V., Bagheri, S., Ratakonda, K., Chee, Y.M.: Agility of enterprise operations across distributed organizations: a model of cross enterprise collaboration. In: Proc. SRII 2011 Global Conf. (2011)
18. Palmisano, S.J.: The globally integrated enterprise. Foreign Aff. 85, 127–136 (2006)
19. Tang, C.S.: Robust strategies for mitigating supply chain disruptions. Int. J. Logist. 9, 33–45 (2006)
20. Tapscott, D., Williams, A.D.: Wikinomics. Portfolio Penguin, New York (2006)
21. Tiwana, A.: Does technological modularity substitute for control? a study of alliance performance in software outsourcing. Strateg. Manage. J. 29, 769–780 (2008)
22. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. 10, 384–393 (1975)
23. Vargo, S.L., Lusch, R.F.: Evolving to a new dominant logic for marketing. J. Mark 68, 1–17 (2004)
24. Varshney, L.R., Oppenheim, D.V.: Coordinating global service delivery in the presence of uncertainty. In: Proc. 12th Int. Research Symp. Service Excellence Manage, QUIS12 (2011)
25. Vergidis, K., Tiwari, A., Majeed, B.: Business process analysis and optimization: Beyond reengineering. IEEE Trans. Syst., Man, Cybern C 38, 69–82 (2008)
26. Wiredu, G.O.: A framework for the analysis of coordination in global software development. In: Proc. Int. Wksp. Global Softw. Dev. Practitioner, pp. 38–44 (2006)

Author Index