

Robert Meersman  
Tharam Dillon  
Pilar Herrero et al. (Eds.)

LNCS 7044

# On the Move to Meaningful Internet Systems: OTM 2011

Confederated International Conferences:  
CoopIS, DOA-SVI, and ODBASE 2011  
Hersonissos, Crete, Greece, October 2011, Proceedings, Part I

1  
Part I

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

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

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbruecken, Germany*

Robert Meersman Tharam Dillon Pilar Herrero  
Akhil Kumar Manfred Reichert Li Qing  
Beng-Chin Ooi Ernesto Damiani Douglas C. Schmidt  
Jules White Manfred Hauswirth Pascal Hitzler  
Mukesh Mohania (Eds.)

# On the Move to Meaningful Internet Systems: OTM 2011

Confederated International Conferences:  
CoopIS, DOA-SVI, and ODBASE 2011  
Hersonissos, Crete, Greece, October 17-21, 2011  
Proceedings, Part I

Volume Editors

Robert Meersman, Vrije Universiteit Brussel, Belgium, meersman@vub.ac.be

Tharam Dillon, Curtin University of Technology, Australia, t.dillon@curtin.edu.au

Pilar Herrero, Universidad Politécnica de Madrid, Spain, pherrero@fi.upm.es

Akhil Kumar, Pennsylvania State University, USA, akhilkumar@psu.edu

Manfred Reichert, University of Ulm, Germany, manfred.reichert@uni-ulm.de

Li Qing, City University of Hong Kong, liqing.thu@gmail.com

Beng-Chin Ooi, National University of Singapore, ooibc@comp.nus.edu.sg

Ernesto Damiani, University of Milan, Italy, ernesto.damiani@unimi.it

Douglas C. Schmidt, Vanderbilt University, USA, schmidt@dre.vanderbilt.edu

Jules White, Virginia Tech, Blacksburg, USA, julesw@vt.edu

Manfred Hauswirth, DERI, Galway, Ireland, manfred.hauswirth@deri.org

Pascal Hitzler, Kno.e.sis, Wright State University, USA, pascal.hitzler@wright.edu

Mukesh Mohania, IBM India, New Delhi, mkmukesh@in.ibm.com

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-25108-5

e-ISBN 978-3-642-25109-2

DOI 10.1007/978-3-642-25109-2

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011940438

CR Subject Classification (1998): C.2, D.2, H.4, I.2, H.2-3, J.1, K.6.5

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

© Springer-Verlag Berlin Heidelberg 2011

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

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

*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](http://www.springer.com))

# General Co-chairs' Message for OnTheMove 2011

The OnTheMove 2011 event in Heraklion, Crete, held during October 17–21, further consolidated the growth of the conference series that was started in Irvine, California, in 2002, and held in Catania, Sicily, in 2003, in Cyprus in 2004 and 2005, in Montpellier in 2006, in Vilamoura in 2007 and 2009, in Monterrey, Mexico, in 2008, and in Heraklion 2010. The event continues to attract a diversified and representative selection of today's worldwide research on the scientific concepts underlying new computing paradigms, which, of necessity, must be distributed, heterogeneous, and autonomous yet meaningfully collaborative. Indeed, as such large, complex, and networked intelligent information systems become the focus and norm for computing, there continues to be an acute and even increasing need to address and discuss face to face in an integrated forum the implied software, system, and enterprise issues as well as methodological, semantic, theoretical, and application issues. As we all realize, email, the Internet, and even video conferences are not by themselves sufficient for effective and efficient scientific exchange.

The OnTheMove (OTM) Federated Conference series has been created to cover the scientific exchange needs of the community/ies that work in the broad yet closely connected fundamental technological spectrum of Web-based distributed computing. The OTM program every year covers data and Web semantics, distributed objects, Web services, databases, information systems, enterprise workflow and collaboration, ubiquity, interoperability, mobility, grid and high-performance computing.

OnTheMove does not consider itself a so-called multi-conference event but instead is proud to give meaning to the “federated” aspect in its full title : it aspires to be a primary scientific meeting place where all aspects of research and development of Internet- and intranet-based systems in organizations and for e-business are discussed in a scientifically motivated way, in a forum of (loosely) interconnected workshops and conferences. This tenth edition of the OTM Federated Conferences event therefore once more provided an opportunity for researchers and practitioners to understand and publish these developments within their individual as well as within their broader contexts. To further promote synergy and coherence, the main conferences of OTM 2011 were conceived against a background of three interlocking global themes:

- Virtual (“Cloud”) Computing Infrastructures and Security
- The Internet of Things, and Semantic Web 2.0
- Collaborative (“Social”) Computing for the Enterprise

Originally the federative structure of OTM was formed by the co-location of three related, complementary, and successful main conference series: DOA

(Distributed Objects and Applications, since 1999), covering the relevant infrastructure-enabling technologies, ODBASE (Ontologies, DataBases and Applications of Semantics, since 2002) covering Web semantics, XML databases and ontologies, and CoopIS (Cooperative Information Systems, since 1993) cover the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. In 2007 the IS workshop (Information Security) was added to try cover also the specific issues of security in complex Internet-based information systems, and in this 2011 edition these security aspects became merged with DOA under the umbrella description of “secure virtual infrastructures,” or DOA-SVI. Each of the main conferences specifically seeks high-quality contributions and encourages researchers to treat their respective topics within a framework that incorporates jointly (a) theory, (b) conceptual design and development, and (c) applications, in particular case studies and industrial solutions.

Following and expanding the model created in 2003, we again solicited and selected quality workshop proposals to complement the more “archival” nature of the main conferences with research results in a number of selected and more “avant-garde” areas related to the general topic of Web-based distributed computing. For instance, the so-called Semantic Web has given rise to several novel research areas combining linguistics, information systems technology, and artificial intelligence, such as the modeling of (legal) regulatory systems and the ubiquitous nature of their usage. We were glad to see that six of our earlier successful workshops (EI2N, SWWS, ORM, MONET, ISDE, SeDeS) re-appeared in 2011 with in some cases a fourth or even fifth edition, often in alliance with other older or newly emerging workshops, and that three brand-new independent workshops could be selected from proposals and hosted: INBAST, RASEP, and VADER. (INBAST was merged with the new Industry Track, under the auspicious leadership of Hervé Panetto and OMG’s Richard Mark Soley.)

We are also proud in particular to note the co-sponsorship of the US National Science Foundation (NSF) for the EI2N workshop (also initiated by Hervé), and which increasingly profiles itself as a successful incubator for new “CoopIS-related” research aspects and topics. Our OTM registration format (“one workshop buys all”) actively intends to stimulate workshop audiences to productively mingle with each other and, optionally, with those of the main conferences.

We were again most happy to see once more in 2011 the number of quality submissions for the OnTheMove Academy (OTMA, formerly called Doctoral Consortium Workshop), our “vision for the future” in research in the areas covered by OTM, managed by a dedicated team of collaborators led by Peter Spyns and Anja Schanzenberger, and of course by the OTMA Dean, Erich Neuhold, responsible for our unique interactive formula to bring PhD students together. In the OTM Academy, PhD research proposals are submitted for peer review; selected submissions and their approaches are (eventually) presented by the students in front of a wider audience at the conference, and independently and extensively analyzed and discussed in front of the audience by a panel of senior professors.

As said, all three main conferences and the associated workshops shared the distributed aspects of modern computing systems, and the resulting application pull created by the Internet and the so-called Semantic Web. For DOA-SVI 2011, the primary emphasis stayed on the distributed object infrastructure and its virtual and security aspects; for ODBASE 2011, the focus became the knowledge bases and methods required for enabling the use of formal semantics in Web-based databases and information systems; for CoopIS 2011, the focus as usual was on the interaction of such technologies and methods with management issues, such as occur in networked organizations and enterprises. These subject areas overlap in a scientifically natural fashion and many submissions in fact also treated an envisaged mutual impact among them. As with the earlier editions, the organizers wanted to stimulate this cross-pollination by a “shared” program of famous keynote speakers around the chosen themes. We were quite proud to announce:

- Amit Sheth, Wright State University, Ohio, USA
- Schahram Dustdar, Vienna University of Technology, Austria
- Siani Pearson, Hewlett-Packard Laboratories, Bristol, UK
- Niky Riga, Raytheon BBN Technologies, Massachusetts, USA

We received a total of 141 submissions for the three main conferences and 104 submissions in total for the workshops. The numbers are comparable with those for 2010. Not only may we indeed again claim success in attracting an increasingly representative volume of scientific papers, many from the USA and Asia, but these numbers of course allow the Program Committees to compose a high-quality cross-section of current research in the areas covered by OTM. In fact, the Program Chairs of the CoopIS 2011 conferences decided to accept only approximately 1 paper for each 5 submissions, while the ODBASE 2011 PC accepted about the same number of papers for presentation and publication as in 2009 and 2010 (i.e., average 1 paper out of 3-4 submitted, not counting posters). For the workshops and DOA-SVI 2011 the acceptance rate varies but the aim was to stay consistently at about 1 accepted paper for 2-3 submitted, and this of course subordinated to peer assessment of scientific quality.

As usual we have separated the proceedings into three volumes with their own titles, two for the main conferences and one for the workshops, and we are again most grateful to the Springer LNCS team in Heidelberg for their professional suggestions and meticulous collaboration in producing the files for downloading on the USB sticks.

The reviewing process by the respective Program Committees was again performed very professionally, and each paper in the main conferences was reviewed by at least three referees, with arbitrated email discussions in the case of strongly diverging evaluations. It may be worth emphasizing that it is an explicit On-TheMove policy that all conference Program Committees and Chairs make their selections completely autonomously from the OTM organization itself. Like last year, paper proceedings were on separate request and order this year, and incurred an extra charge.

The General Chairs are once more especially grateful to the many people directly or indirectly involved in the set-up of these federated conferences. Not everyone realizes the large number of persons that need to be involved, and the huge amount of work, commitment, and in the uncertain economic and funding climate of 2011 certainly also financial risk, the organization of an event like OTM entails. Apart from the persons in their roles mentioned above, we therefore wish to thank in particular our eight main conference PC Co-chairs:

- CoopIS 2011: Manfred Reichert, Akhil Kumar, Qing Li
- ODBASE 2011: Manfred Hauswirth, Pascal Hitzler, Mukesh Mohania
- DOA-SVI 2011: Ernesto Damiani, Doug Schmidt, Beng Chin Ooi

And similarly the 2011 OTMA and Workshops PC (Co-)chairs (in arbitrary order): Hervé Panetto, Qing Li, J. Cecil, Thomas Moser, Yan Tang (2x), Alok Mishra, Jürgen Münch, Ricardo Colomo Palacios, Deepti Mishra, Patrizia Grifoni, Fernando Ferri, Irina Kondratova, Arianna D'Ulizia, Terry Halpin, Herman Balsters, Almudena Alcaide, Naoki Masuda, Esther Palomar, Arturo Ribagorda, Yan Zhang, Jan Vanthienen, Ernesto Damiani (again), Elizabeth Chang, Paolo Ceravolo, Omar Khadeer Hussain, Miguel Angel Pérez-Toledano, Carlos E. Cuesta, Renaud Pawlak, Javier Cámara, Stefanos Gritzalis, Peter Spyns, Anja Metzner, Erich J. Neuhold, Alfred Holl, and Maria Esther Vidal.

All of them together with their many PC members, performed a superb and professional job in managing the difficult yet existential process of peer review and selection of the best papers from the harvest of submissions. We are all also grateful to our supremely competent and experienced Conference Secretariat and technical support staff in Antwerp and Guadalajara, Jan Demey and Daniel Meersman, and last but certainly not least to our proceedings production team in Perth (DEBII-Curtin University) this year led by Christopher Jones.

The General Co-chairs acknowledge with gratitude the academic freedom, logistic support, and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB), Curtin University, Perth, Australia, and Universidad Politécnica de Madrid (UPM), without which such an enterprise quite simply would not be feasible. We do hope that the results of this federated scientific enterprise contribute to your research and your place in the scientific network... We look forward to seeing you again at next year's event!

August 2011

Robert Meersman  
Tharam Dillon  
Pilar Herrero



# Organization

OTM (On The Move) is a federated event involving a series of major international conferences and workshops. These proceedings contain the papers presented at the OTM 2011 Federated conferences, consisting of three conferences, namely, CoopIS 2011 (Cooperative Information Systems), DOA-SVI 2011 (Secure Virtual Infrastructures), and ODBASE 2011 (Ontologies, Databases and Applications of Semantics).

## Executive Committee

### General Co-chairs

Robert Meersman	VU Brussels, Belgium
Tharam Dillon	Curtin University of Technology, Australia
Pilar Herrero	Universidad Politécnica de Madrid, Spain

### OnTheMove Academy Dean

Erich Neuhold	University of Vienna, Austria
---------------	-------------------------------

### Industry Case Studies Program Chairs

Hervé Panetto	Nancy University, France
Richard Mark Soley	OMG, USA

### CoopIS 2011 PC Co-chairs

Akhil Kumar	Penn State University, USA
Manfred Reichert	University of Ulm, Germany
Qing Li	City University of Hong Kong

### DOA-SVI 2011 PC Co-chairs

Beng Chin Ooi	National University Singapore
Ernesto Damiani	Milan University, Italy
Douglas C. Schmidt	SEI, USA
Jules White	Virginia Polytechnic Institute and State University, USA

### ODBASE 2011 PC Co-chairs

Manfred Hauswirth	DERI, Ireland
Pascal Hitzler	Kno.e.sis, Wright State University, USA
Mukesh Mohania	IBM India

**Publication Chair**

Christopher Jones

Curtin University of Technology, Australia

**Publicity-Sponsorship Chair**

Ana-Cecilia Martinez Barbosa DOA Institute, Belgium

**Logistics Team**

Daniel Meersman

Head of Operations

Ana-Cecilia Martinez Barbosa

Jan Demey

**CoopIS 2011 Program Committee**

Marco Aiello

Hiroyuki Kitagawa

Antonio Ruiz Cortés

Frank Leymann

Joonsoo Bae

Guohui Li

Zohra Bellahsene

Rong Liu

Brian Blake

ZongWei Luo

Nacer Boudjlida

Sanjay K. Madria

James Caverlee

Tiziana Margaria

Jorge Cardoso

Leo Mark

Francisco Curbera

Maristella Matera

Vincenzo D'Andrea

Massimo Mecella

Xiaoyong Du

Jan Mendling

Schahram Dustdar

John Miller

Kaushik Dutta

Arturo Molina

Johann Eder

Nirmal Mukhi

Rik Eshuis

Miyuki Nakano

Ling Feng

Maira C. Norrie

Renato Fileto

Selmin Nurcan

HongGao Harbin

Werner Nutt

Ted Goranson

Gerald Oster

Paul Grefen

Hervé Panetto

Michael Grossniklaus

Cesare Pautasso

Amarnath Gupta

Barbara Pernici

Mohand-Said Hacid

Lakshmish Ramaswamy

Jan Hidders

Stefanie Rinderle-Ma

Birgit Hofreiter

Shazia Sadiq

Zhixing Huang

Ralf Schenkel

Stefan Jablonski

Jialie Shen

Paul Johannesson

Aameek Singh

Epaminondas Kapetanios

Jianwen Su

Dimka Karastoyanova

Xiaoping Sun

Rania Khalaf

Susan Urban

Willem-Jan van den Heuvel  
 Irene Vanderfeesten  
 François B. Vernadat  
 Maria Esther Vidal  
 Barbara Weber  
 Mathias Weske

Andreas Wombacher  
 Jian Yang  
 Xiaoming Yao  
 Shuigeng Zhou

## DOA-SVI 2011 Program Committee

Rafael Accorsi  
 Moataz Ahmed  
 Ghazi Alkhatib  
 Jaiganesh Balasubramanian  
 Massimo Banzi  
 Elisa Bertino  
 Lionel Brunie  
 Marco Casassa-Mont  
 Fabio Casati  
 Frederic Cuppens  
 Alfredo Cuzzocrea  
 Schahram Dustdar  
 Eduardo Fernandez  
 Elena Ferrari  
 Alban Gabillon  
 Chris Gill  
 Andy Gokhale  
 Nils Gruschka  
 James Hill  
 Patrick Hung  
 David Jiang

Guoliang Li  
 Xumin Liu  
 Joe Loyall  
 Leszek Maciaszek  
 Antonio Maña  
 Priya Narasimhan  
 Jean-Cristophe Pazzaglia  
 Nilabja Roy  
 Joerg Schwenk  
 Ahmed Serhrouchni  
 George Spanoudakis  
 Azzel Taleb-Bendiab  
 Sumant Tambe  
 Bhavani Thuraisingham  
 Setsuo Tsuruta  
 Sai Wu  
 Qi Yu  
 Xiaofang Zhou  
 Aoying Zhou

## ODBASE 2011 Program Committee

Karl Aberer  
 Divyakant Agrawal  
 Harith Alani  
 Sören Auer  
 Payam Barnaghi  
 Ladjel Bellatreche  
 Paul-Alexandru Chirita  
 Sunil Choenni  
 Oscar Corcho  
 Philippe Cudre-Mauroux  
 Bernardo Cuenca Grau

Emanuele Della Valle  
 Prasad Deshpande  
 Jérôme Euzenat  
 Walid Gaaloul  
 Aldo Gangemi  
 Giancarlo Guizzardi  
 Peter Haase  
 Harry Halpin  
 Takahiro Hara  
 Andreas Harth  
 Manfred Hauswirth

Martin Hepp  
 Pascal Hitzler  
 Andreas Hotho  
 Prateek Jain  
 Krzysztof Janowicz  
 Matthias Klusch  
 Shin'Ichi Konomi  
 Manolis Koubarakis  
 Rajasekar Krishnamurthy  
 Shonali Krishnaswamy  
 Reto Krummenacher  
 Werner Kuhn  
 Steffen Lamparter  
 Wookey Lee  
 Sanjay Madria  
 Frederick Maier  
 Mukesh Mohania  
 Anirban Mondal  
 Jeff Z. Pan  
 Kalpdrum Passi  
 Dimitris Plexousakis

Ivana Podnar Zarko  
 Axel Polleres  
 Guilin Qi  
 Prasan Roy  
 Sourav S Bhowmick  
 Satya Sahoo  
 Nandlal Sarda  
 Kai-Uwe Sattler  
 Peter Scheuermann  
 Christoph Schlieder  
 Michael Schrefl  
 Wolf Siberski  
 Srinath Srinivasa  
 Heiner Stuckenschmidt  
 L. Venkata Subramaniam  
 York Sure  
 Kunal Verma  
 Wei Wang  
 Josiane Xavier Parreira  
 Guo-Qiang Zhang

## Supporting and Sponsoring Institutions

OTM 2011 was proudly supported or sponsored by Vrije Universiteit Brussel in Belgium, Curtin University of Technology in Australia, Universidad Politecnica de Madrid in Spain, the Object Management Group, and Collibra.



# Computing for Human Experience: Semantics Empowered Cyber-Physical, Social and Ubiquitous Computing beyond the Web

Amit Sheth

Kno.e.sis, Wright State University, USA

## Short Bio

Amit Sheth is an educator, research and entrepreneur. He is the LexisNexis Ohio Eminent Scholar at the Wright State University, Dayton OH. He directs Kno.e.sis - the Ohio Center of Excellence in Knowledge-enabled Computing which works on topics in Semantic, Social, Sensor and Services computing over Web and in social-cyber-physical systems, with the goal of transitioning from information age to meaning age. Prof. Sheth is an IEEE fellow and is one of the highly cited authors in Computer Science (h-index = 67) and World Wide Web. He is EIC of ISI indexed Intl. Journal of Semantic Web & Information Systems (<http://ijswis.org>), is joint-EIC of Distributed & Parallel Databases, is series co-editor of two Springer book series, and serves on several editorial boards. By licensing his funded university research, he has also founded and managed two successful companies. Several commercial products and many operationally deployed applications have resulted from his R&D.

## Talk

“Computing for Human Experience: Semantics empowered Cyber-Physical, Social and Ubiquitous Computing beyond the Web”

Traditionally, we had to artificially simplify the complexity and richness of the real world to constrained computer models and languages for more efficient computation. Today, devices, sensors, human-in-the-loop participation and social interactions enable something more than a “human instructs machine” paradigm. Web as a system for information sharing is being replaced by pervasive computing with mobile, social, sensor and devices dominated interactions. Correspondingly, computing is moving from targeted tasks focused on improving efficiency and productivity to a vastly richer context that support events and situational awareness, and enrich human experiences encompassing recognition of rich sets of relationships, events and situational awareness with spatio-temporal-thematic elements, and socio-cultural-behavioral facets. Such progress positions us for

what I call an emerging era of “computing for human experience” (CHE). Four of the key enablers of CHE are: (a) bridging the physical/digital (cyber) divide, (b) elevating levels of abstractions and utilizing vast background knowledge to enable integration of machine and human perception, (c) convert raw data and observations, ranging from sensors to social media, into understanding of events and situations that are meaningful to humans, and (d) doing all of the above at massive scale covering the Web and pervasive computing supported humanity. Semantic Web (conceptual models/ontologies and background knowledge, annotations, and reasoning) techniques and technologies play a central role in important tasks such as building context, integrating online and offline interactions, and help enhance human experience in their natural environment.

# Privacy and the Cloud

Siani Pearson

Hewlett-Packard Laboratories

## Short Bio

Siani Pearson is a senior researcher in the Cloud and Security Research Lab (HP Labs Bristol, UK), HP's major European long term applied research centre. She has an MA in Mathematics and Philosophy from Oxford and a PhD in Artificial Intelligence from Edinburgh. She was a Fellow at the Computer Lab in Cambridge University, and for the last 17 years has worked at HP Labs in a variety of research and development programs including collaborations with HP business units and EU PRIME (Privacy and Identity Management for Europe) project.

Siani's current research focus is on privacy enhancing technologies, accountability and the cloud. She is a technical lead on regulatory compliance projects with HP Privacy Office and HP Enterprise Services, and on the collaborative TSB-funded EnCoRe (Ensuring Consent and Revocation) project.

## Talk

"Privacy and the Cloud"

Cloud computing offers a huge potential both for efficiency and new business opportunities (especially in service composition), and is almost certain to deeply transform our IT. However, the convenience and efficiency of this approach comes with a range of potential privacy and security risks. Indeed, a key barrier to the widespread uptake of cloud computing is the lack of trust in clouds by potential customers. This concern is shared by experts: the European Network and Information Security Agency (ENISA)'s cloud computing risk assessment report states "loss of governance" as a top risk of cloud computing, and "data loss or leakages" is one of the top seven threats the Cloud Security Alliance (CSA) lists in its Top Threats to Cloud Computing report.

In this talk I will assess how privacy, security and trust issues occur in the context of cloud computing and explain how complementary regulatory, procedural and technical provisions can be used to help address these issues. In particular, accountability is likely to become a core concept in both the cloud and in new

mechanisms that help increase trust in cloud computing. It is especially helpful for protecting sensitive or confidential information, enhancing consumer trust, clarifying the legal situation in cloud computing, and facilitating cross-border data transfers. I will also talk about some of the innovative technical solutions that we are developing in HP Labs to enhance privacy in the cloud.



# The Social Compute Unit

Schahram Dustdar

Vienna University of Technology (TU Wien)

## Short Bio

Schahram Dustdar (ACM Distinguished Scientist), is full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Vienna University of Technology (TU Wien).

From 1999 - 2007 he worked as the co-founder and chief scientist of Caramba Labs Software AG in Vienna (acquired by Engineering NetWorld AG), a venture capital co-funded software company focused on software for collaborative processes in teams. He is Editor in Chief of Computing (Springer) and on the editorial board of IEEE Internet Computing, as well as author of some 300 publications.

## Talk

“The Social Compute Unit”

Social computing is perceived mainly as a vehicle for establishing and maintaining social (private) relationships as well as utilizing political and social interests. Unsurprisingly, social computing lacks substantial adoption in enterprises. Clearly, collaborative computing is firmly established (as a niche), but no tight integration exists of social and collaborative computing approaches to facilitate mainstream problem solving in and between enterprises or teams of people. In this talk I will present a fresh look at this problem and examine how to integrate people in the form of human-based computing and software services into one composite system, which can be modeled, programmed, and instantiated on a large scale.

# GENI - Global Environment for Network Innovations

Niky Riga

GENI Project Office, Raytheon BBN Technologies

## Short Bio

Niky Riga is a Network Scientist at Raytheon BBN Technologies. Niky joined the GENI Project Office (GPO) in March 2010. As a member of GPO, Niky is responsible for supporting GENI experimenters in integrating and deploying their experiments as well as advocating their requirements to the rest of the GENI community.

Before joining the GPO, Niky worked on multiple innovative projects within the Network Research department of BBN. Her focus is on designing and prototyping pioneering transport services for Mobile Ad-hoc Networks, while her main goal is making innovative, research ideas practical and implementing them on real systems. She has successfully led various integration efforts. Niky earned a Diploma in Electrical and Computer Engineering at the National Technical University of Athens, and an MS degree in Computer Science at Boston University.

## Talk

“GENI - Global Environment for Network Innovations”

The Global Environment for Network Innovations (GENI) is a suite of research infrastructure components rapidly taking shape in prototype form across the US. It is sponsored by the US National Science Foundation, with the goal of becoming the world's first laboratory environment for exploring future Internets at scale, promoting innovations in network science, security, technologies, services, and applications.

GENI allows academic and industrial researchers to perform a new class of experiments that tackle critically important issues in global communications networks such as (a) Science issues: we cannot currently understand or predict the behavior of complex, large-scale networks, (b) Innovation issues: we face substantial barriers to at-scale experimentation with new architectures, services, and technologies (c) Society issues: we increasingly rely on the Internet but are unsure that can we trust its security, privacy or resilience GENI is enabling

researchers to explore these issues by running large-scale, well-instrumented, end-to-end experiments engaging substantial numbers of real users. These experiments may be fully compatible with today's Internet, variations or improvements on today's Internet protocols, or indeed radically novel, clean slate designs. The GENI project is now supporting such experiments across a mesoscale build-out through more than a dozen US campuses, two national backbones, and several regional networks. If this effort proves successful, it will provide a path toward more substantial build-out.

In this keynote presentation, she will introduce GENI through a couple of example use-cases, she will review the growing suite of infrastructure and evolving control framework. She will also present previous and current experiments running in GENI.

# Table of Contents – Part I

## Cooperative Information Systems (CoopIS) 2011

CoopIS 2011 PC Co-chairs' Message . . . . .	1
---	---

## Business Process Repositories

Searching Business Process Repositories Using Operational Similarity . . . <i>Maya Lincoln and Avigdor Gal</i>	2
---	---

Fragment-Based Version Management for Repositories of Business Process Models . . . . .	20
<i>Chathura C. Ekanayake, Marcello La Rosa, Arthur H.M. ter Hofstede, and Marie-Christine Fauvet</i>	

Selecting and Ranking Business Processes with Preferences: An Approach Based on Fuzzy Sets . . . . .	38
<i>Katia Abbaci, Fernando Lemos, Allel Hadjali, Daniela Grigori, Ludovic Liétard, Daniel Rocacher, and Mokrane Bouzeghoub</i>	

Efficient Retrieval of Similar Business Process Models Based on Structure (Short Paper) . . . . .	56
<i>Tao Jin, Jianmin Wang, and Lijie Wen</i>	

## Business Process Compliance and Risk Management

Preservation of Integrity Constraints by Workflow . . . . .	64
<i>Xi Liu, Jianwen Su, and Jian Yang</i>	

Monitoring Business Process Compliance Using Compliance Rule Graphs . . . . .	82
<i>Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam</i>	

History-Aware, Real-Time Risk Detection in Business Processes . . . . .	100
<i>Raffaele Conforti, Giancarlo Fortino, Marcello La Rosa, and Arthur H.M. ter Hofstede</i>	

## Service Orchestration and Workflows

Transactional Process Views . . . . .	119
<i>Rik Eshuis, Jochem Vonk, and Paul Grefen</i>	

Edit Distance-Based Pattern Support Assessment of Orchestration Languages ..... 137  
*Jörg Lenhard, Andreas Schönberger, and Guido Wirtz*

Towards Robust Service Workflows: A Decentralized Approach (Short Paper) ..... 155  
*Mario Henrique Cruz Torres and Tom Holvoet*

**Intelligent Information Systems and Distributed Agent Systems**

Pricing Information Goods in Distributed Agent-Based Information Filtering ..... 163  
*Christos Tryfonopoulos and Laura Maria Andreescu*

Trust Alignment: A Sine Qua Non of Open Multi-agent Systems ..... 182  
*Andrew Koster, Jordi Sabater-Mir, and Marco Schorlemmer*

An Architecture for Defeasible-Reasoning-Based Cooperative Distributed Planning ..... 200  
*Sergio Pajares Ferrando, Eva Onaindia, and Alejandro Torreño*

A Case Retrieval Approach Using Similarity and Association Knowledge ..... 218  
*Yong-Bin Kang, Shonali Krishnaswamy, and Arkady Zaslavsky*

**Emerging Trends in Business Process Support**

FlexCon – Robust Context Handling in Human-Oriented Pervasive Flows ..... 236  
*Hannes Wolf, Klaus Herrmann, and Kurt Rothermel*

An Artifact-Centric Approach to Dynamic Modification of Workflow Execution ..... 256  
*Wei Xu, Jianwen Su, Zhimin Yan, Jian Yang, and Liang Zhang*

Event Cube: Another Perspective on Business Processes ..... 274  
*J.T.S. Ribeiro and A.J.M.M. Weijters*

**Techniques for Building Cooperative Information Systems**

Building eCommerce Systems from Shared Micro-schemas ..... 284  
*Stefania Leone and Moira C. Norrie*

A<sup>2</sup>-VM: A Cooperative Java VM with Support for Resource-Awareness and Cluster-Wide Thread Scheduling ..... 302  
*José Simão, João Lemos, and Luís Veiga*

Peer-Based Relay Scheme of Collaborative Filtering for Research Literature . . . . .	321
<i>Youliang Zhong, Weiliang Zhao, Jian Yang, and Lai Xu</i>	

## Security and Privacy in Collaborative Applications

Detecting and Resolving Conflicts of Mutual-Exclusion and Binding Constraints in a Business Process Context . . . . .	329
<i>Sigrid Schefer, Mark Strembeck, Jan Mendling, and Anne Baumgrass</i>	
Implementation, Optimization and Performance Tests of Privacy Preserving Mechanisms in Homogeneous Collaborative Association Rules Mining . . . . .	347
<i>Marcin Gorawski and Zacheusz Siedlecki</i>	

## Data and Information Management

Segmenting and Labeling Query Sequences in a Multidatabase Environment . . . . .	367
<i>Aybar C. Acar and Amihai Motro</i>	
Combining Resource and Location Awareness in DHTs . . . . .	385
<i>Liz Ribe-Baumann</i>	
SQL Streaming Process in Query Engine Net . . . . .	403
<i>Qiming Chen and Meichun Hsu</i>	
Instance-Based ‘One-to-Some’ Assignment of Similarity Measures to Attributes (Short Paper) . . . . .	412
<i>Tobias Vogel and Felix Naumann</i>	
Matching and Alignment: What Is the Cost of User Post-Match Effort? (Short Paper) . . . . .	421
<i>Fabien Duchateau, Zohra Bellahsene, and Remi Coletta</i>	
<b>Author Index . . . . .</b>	<b>429</b>

## Table of Contents – Part II

### Distributed Objects and Applications and Secure Virtual Infrastructures (DOA-SVI) 2011

DOA-SVI 2011 PC Co-chairs' Message . . . . .	431
--	-----

### Performance Measurement and Optimization

Optimizing Integrated Application Performance with Cache-Aware Metascheduling . . . . .	432
<i>Brian Dougherty, Jules White, Russell Kegley, Jonathan Preston, Douglas C. Schmidt, and Aniruddha Gokhale</i>	

Dynamic Migration of Processing Elements for Optimized Query Execution in Event-Based Systems . . . . .	451
<i>Waldemar Hummer, Philipp Leitner, Benjamin Satzger, and Shahram Dustdar</i>	

A Survey on SLA and Performance Measurement in Cloud Computing . . . . .	469
<i>Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang</i>	

### Instrumentation, Monitoring, and Provisioning

Experiences with Service-Oriented Middleware for Dynamic Instrumentation of Enterprise DRE Systems . . . . .	478
<i>James H. Hill and Douglas C. Schmidt</i>	

Dynamic Event-Based Monitoring in a SOA Environment . . . . .	498
<i>Fabio Souza, Danilo Lopes, Kiev Gama, Nelson Rosa, and Ricardo Lima</i>	

A SIP-Based Network QoS Provisioning Framework for Cloud-Hosted DDS Applications . . . . .	507
<i>Akram Hakiri, Aniruddha Gokhale, Douglas C. Schmidt, Berthou Pascal, Joe Hoffert, and Gayraud Thierry</i>	

### Quality of Service

Continuous Access to Cloud Event Services with Event Pipe Queries . . .	525
<i>Qiming Chen and Meichun Hsu</i>	

QoS-Enabled Distributed Mutual Exclusion in Public Clouds . . . . .	542
<i>James Edmondson, Doug Schmidt, and Aniruddha Gokhale</i>	

## Security and Privacy

Towards Pattern-Based Reliability Certification of Services . . . . .	560
<i>Ingrid Buckley, Eduardo B. Fernandez, Marco Anisetti, Claudio A. Ardagna, Masoud Sadjadi, and Ernesto Damiani</i>	
Direct Anonymous Attestation: Enhancing Cloud Service User Privacy . . . . .	577
<i>Ulrich Greveler, Benjamin Justus, and Dennis Loehr</i>	
Trust Management Languages and Complexity . . . . .	588
<i>Krzysztof Sacha</i>	

## Models and Methods

Ontology-Based Matching of Security Attributes for Personal Data Access in e-Health . . . . .	605
<i>Ioana Ciuciu, Brecht Claerhout, Louis Schilders, and Robert Meersman</i>	
A Unified Ontology for the Virtualization Domain . . . . .	617
<i>Jacopo Silvestro, Daniele Canavese, Emanuele Cesena, and Paolo Smiraglia</i>	
2PSIM: Two Phase Service Identifying Method . . . . .	625
<i>Ali Nikravesh, Fereidoon Shams, Soodeh Farokhi, and Amir Ghaffari</i>	
Automated Statistical Approach for Memory Leak Detection: Case Studies . . . . .	635
<i>Vladimir Šor, Nikita Salnikov-Tarnowski, and Satish Narayana Srirama</i>	

## Ontologies, DataBases, and Applications of Semantics (ODBASE) 2011

ODBASE 2011 PC Co-chairs' Message . . . . .	643
---	-----

## Acquisition of Semantic Information

RDFa Based Annotation of Web Pages through Keyphrases Extraction . . . . .	644
<i>Roberto De Virgilio</i>	



An Ontological and Terminological Resource for n-ary Relation Annotation in Web Data Tables .....	662
<i>Rim Touhami, Patrice Buche, Juliette Dibia-Barthélemy, and Liliana Ibănescu</i>	

Inductive Learning of Disjointness Axioms .....	680
<i>Daniel Fleischhacker and Johanna Völker</i>	

## Use of Semantic Information

Breaking the Deadlock: Simultaneously Discovering Attribute Matching and Cluster Matching with Multi-Objective Simulated Annealing .....	698
<i>Haishan Liu and Dejing Dou</i>	

To Cache or Not To Cache: The Effects of Warming Cache in Complex SPARQL Queries .....	716
<i>Tomas Lampo, María-Esther Vidal, Juan Danilow, and Edna Ruckhaus</i>	

Implementation of Updateable Object Views in the ODRA OODBMS .....	734
<i>Radosław Adamus, Tomasz Marek Kowalski, and Jacek Wiślicki</i>	

## Reuse of Semantic Information

Domain Expert Centered Ontology Reuse for Conceptual Models .....	747
<i>Christian Kop</i>	

Semantic Invalidation of Annotations Due to Ontology Evolution .....	763
<i>Julius Köpke and Johann Eder</i>	

The Role of Constraints in Linked Data .....	781
<i>Marco Antonio Casanova, Karin Koogan Beitman, Antonio Luz Furtado, Vania M.P. Vidal, José A.F. Macedo, Raphael Valle A. Gomes, and Percy E. Rivera Salas</i>	

## ODBASE 2011 Short Papers

A Generic Approach for Combining Linguistic and Context Profile Metrics in Ontology Matching .....	800
<i>DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta</i>	

ADERIS: An Adaptive Query Processor for Joining Federated SPARQL Endpoints .....	808
<i>Steven Lynden, Isao Kojima, Akiyoshi Matono, and Yusuke Tanimura</i>	

Asynchronous Replication for Evolutionary Database Development: A Design for the Experimental Assessment of a Novel Approach . . . . .	818
<i>Helves Humberto Domingues, Fabio Kon, and João Eduardo Ferreira</i>	
Improving the Accuracy of Ontology Alignment through Ensemble Fuzzy Clustering . . . . .	826
<i>Nafisa Afrin Chowdhury and Dejing Dou</i>	
<b>Author Index</b> . . . . .	835

# CoopIS 2011 PC Co-chairs' Message

Welcome to the proceedings of CoopIS 2011 held in beautiful Crete! This was the 19th year of this conference series. CoopIS has established itself as a premier conference in the information science and systems area, and as an important part of the OTM ("OnTheMove") federated conferences, covering different aspects of distributed, intelligent, and cooperative information systems. The focus of CoopIS is on process management technologies, middleware and architectures for cooperative information systems, and cooperative information systems applications.

We had a strong program this year. The selection process was very competitive which resulted in quite a few good papers being rejected. Each paper was reviewed by at least three, and in many cases four, reviewers. From the 95 submissions, we were able to accept only 20 papers as full papers. In addition, eight papers were included in the program as short papers. We are very grateful to the reviewers who worked hard to meet a tight deadline during a vacation period. We were also very ably assisted by the staff at the OTM secretariat, particularly by Jan Demey. Without their invaluable assistance we would not have been able to put the program together. We also thank the OTM General Chairs Robert Meersman and Tharam Dillon for all their support.

August 2011

Manfred Reichert  
Akhil Kumar  
Qing Li

# Searching Business Process Repositories Using Operational Similarity

Maya Lincoln and Avigdor Gal

Technion - Israel Institute of Technology  
mayal@technion.ac.il, avigal@ie.technion.ac.il

**Abstract.** Effective retrieval of relevant know-how segments from business process repositories can save precious employee time and support non-expert users in locating and reusing process data. We present a methodology for searching repositories and retrieving relevant process segments, using business logic that is extracted from real-life process models. The analysis of a process repository enables the construction of three taxonomies with which it is possible to process the search intention in operational terms. We tested the method on the Oracle ERP Business Process Model (OBM), showing the approach to be effective in enabling the search of business process repositories.

**Keywords:** Business process search, Business process repositories, Dynamic segmentation of process models.

## 1 Introduction

Researchers have become increasingly interested in developing methods and tools for automatically retrieving information from business process repositories [3]. Such repositories are considered important and valuable data reservoirs of organizational know-how. In particular, they enable the retrieval of relevant knowledge segments saving precious time and supporting non-expert users in locating and reusing required process data [24].

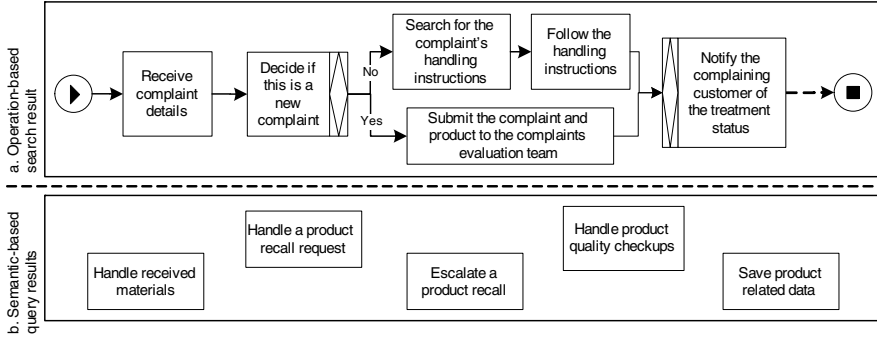
Two common methods for retrieving information from a repository are querying and searching. The former is aimed at retrieving structured information using a structured query language. The significance of querying business processes has been acknowledged by BPMI<sup>1</sup> that launched a Business Process Query Language (BPQL) initiative. The latter allows querying information using keywords or natural language and was shown in other areas (e.g. information retrieval) to be an effective method for non-experts.

Research in the field of business process retrieval has mainly focused on *words semantics* and *structural* similarity analysis techniques [3,17,4]. Using these frameworks one can retrieve process models that either contain semantically related components (e.g. activity names with a specified keyword) or match a requested graph structure (e.g. graph paths in which activity A is followed by

---

<sup>1</sup> Business Process Management Initiative, <http://www.bpmi.org/>

activity B). In this work we tackle the information retrieval challenge from a different angle, with a search framework for business processes that uses operational similarity and enables the retrieval of process *segments* that comply with organizational standards.



**Fig. 1.** An example of search results for “how to handle a product recall”

As a motivating example consider an employee interested in finding out “how to handle a product recall.” An expected outcome of this retrieval request would be a *segment* from the process repository that represents the order of activities that one should follow in order to achieve the required process goal, as illustrated in Fig 1a. The benefit of such a retrieval framework is that the result is ready for execution. Without any preliminary knowledge of the underlying repository structure, the user can receive a full-fledged process model.

The retrieval output is related to the search phrase in *operational* terms. For example, Fig 1a provides a segment that is only marginally similar to the search phrase text. Specifically, two of the search phrase terms (“Handle” and “Recall”) are not represented by any of its activities while the term “Product” is represented by only one activity. Such “how-to” questions are hard to fulfill using common query languages due to the complex logic that is embedded within such questions [4] and especially without specific knowledge on process structure and activity naming. Therefore, using querying techniques, even if they are based on semantic similarity, would likely yield disconnected components, as illustrated in Fig 1b. Such outcome does not tell the user “how-to” fulfill the process goal.

The retrieval framework we propose is based on *operational* similarity. The business logic is extracted from process repositories through the analysis of process activities. Each activity is encoded automatically as a *descriptor* [16]. The collection of all descriptors induces three taxonomies, namely an action scope model, an object grouping model, and an action influence model, with which we can search for the appropriate graph segment, which is the activity flow that provides an answer to the search phrase. The proposed method *dynamically* segments a process repository according to the ad-hoc request as expressed in the user’s search phrase.

We present an empirical evaluation based on real-world data, showing that by utilizing the descriptor taxonomies and the process model retrieval method it is possible to effectively support the search of process repositories in operational terms. On average, the best result (that was closest to the expected result) retrieved in each experiment had a 93% precision and was ranked almost always as the first option in the result list.

This work proposes an innovative method for searching business process models while making use of the *how-to* knowledge that is encoded in business process repositories. The following contributions are presented: (a) generic support to an operation-based search of business process models; (b) automatic extraction of business logic from business process repositories; (c) capability to generate ad-hoc process model segments; and (d) an empirical analysis that evaluates the method's usefulness.

The rest of the paper is organized as follows: we present related work in Section 2 positioning our work with respect to previous research. In Section 3 we present the notion of dynamic segmentation in process model repositories. In Section 4 we present an activity decomposition model that is used as the foundation for creating action and object taxonomies. We formulate the search problem and describe our method for searching business process repositories in Section 5. Section 6 introduces our empirical analysis. We conclude in Section 7.

## 2 Related Work

Related works include query and search techniques in BPM. Works such as [19,20,4,2,9] query business process repositories to extract process model (graph) segments. Such methods require prior knowledge of the structure of the process repository and the exact notation that is used to express it while our work offers techniques that work well even without prior knowledge regarding the process repository.

Keyword search on general tree or graph data structures can also be applied to process repositories [12,10,11]. These methods allow users to find information without having to learn a complex query language or getting prior knowledge of the process structure. Some works extend the tree and graph keyword search methods to support a more intuitive interface for the user by enabling searches based on natural language [14,13]. According to [1], the straightforwardness of a natural language makes it the most desirable database query interface. The retrieved information in both keyword and natural language search methods is in the form of single process model components such as activities and roles. Our work extends such works to support the retrieval of complete segments by applying dynamic segmentation of the process repository. The search result is a compendium of data (a segment of a business process model) related to the operational meaning of the searched text.

Another line of work focuses on automatic construction of process data ontologies. The work in [5] proposes a query-by-example approach that relies on ontological description of business processes, activities, and their relationships,

which can be automatically built from the workflow models themselves. The work in [6] automatically extracts the semantics from searched conceptual models, without requiring manual meta-data annotation, while basing its method on a model-independent framework. Our framework differs from these works in that we target the automatic extraction and usage of the operational layer (the “how-to”) and the business rules encapsulated in the process repository.

### 3 Dynamic Segmentation

This section describes dynamic segmentation in process model repositories. Section 3.1 presents a model for business process repositories followed by a description and a formal model of a dynamic segmentation.

#### 3.1 A Business Process Repository and Static Segmentation

The Workflow Management Coalition (WFMC) [7] defines a *business process model* as a “set of one or more linked procedures or activities which collectively realize a business objective or policy goal.” An example of such a business process model is the “Fulfill a Purchase Order” process model, presented in Fig. 2a using YAWL [22]. The dashed lines represent parts of the process that are not shown due to space limitations.

A *process repository* is a collection of interconnected process models, where the execution of an activity in one process model may invoke the execution of another activity in another process model. Formally, we define a process repository to be a graph  $G = (V, E)$ , where  $V$  represents activities and an edge  $(a_i, a_j)$  in  $E$  exists if  $a_j$ ’s execution follows that of  $a_i$ .

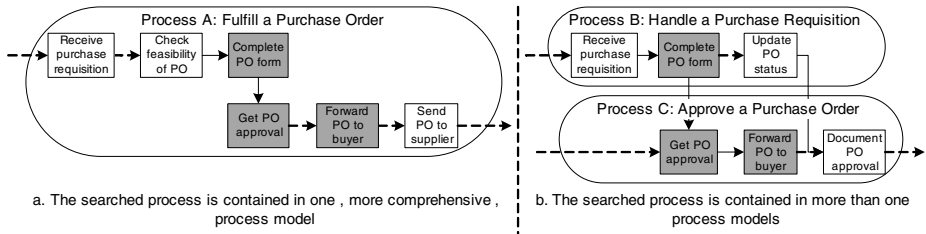


Fig. 2. Examples of process models

Each of the process models in the process repository has a name, and we refer to it as a *static* segment of the process repository. For example, the processes “Handle a Purchase Requisition” in Fig. 2b represents a segment of process activities (steps). This static, pre-defined, segmentation of process repositories is determined according to the logic of the repository developer and changes from one repository to the other. To illustrate, the SAP business process repository consists of more than 16,500 activities segmented into more than 2,400 processes

while the Oracle business process repository (OBM) consists of about 9,700 activities segmented into 1,553 processes. 92% of the two repositories refer to the same business areas (e.g. processes from the domain of accounting, human resource management, production, logistics, etc.), yet only 23% of the process names are semantically similar. The rest of the processes have segments that do not share starting and/or ending activities.

### 3.2 Dynamic Segments in Business Process Repositories

It is possible to partition the process repository graph in different ways, creating new, *dynamic*, process model segments. To illustrate, consider a process for manually creating a purchase order. In one repository, this process can be part of a larger process model called: “Fulfill a Purchase Order,” which includes the creation, approval, and submission of a purchase order (see illustration in Fig. 2a, where the dynamic segment activities are shaded). In a different process repository, with different process models, the sequence of activities that fulfill the example process goal cut across two static process models: “Handle a Purchase Requisition” and “Approve a Purchase Order” (see illustration in Fig. 2b). Note that although in both repositories there is an activity sequence that fulfills the searched process goal (Manually Create a Purchase Order) - this process model is not represented separately (as a standalone segment) in those repositories and therefore does not have a pre-defined process name.

Formally, we can define a dynamic segment to be a sub-graph induced by  $G$ . Let  $\mathcal{G}$  be the set of all subgraphs that are induced by  $G$ , and hence, the set of all possible dynamic segmentations of a repository.

$$\mathcal{G} = \{G' = (V', E') \mid V' = \{v_1, v_2, \dots, v_n\} \subseteq 2^{|V|} \wedge \forall \{v_i, v_j\} \in V', (v_i, v_j) \in E' \text{ if } (v_i, v_j) \in E\}$$

## 4 Descriptor Analysis

This section enhances the descriptor model of [16,15] to support process model search. We provide a brief overview of the descriptor model in Section 4.1 followed by an introduction of three new taxonomies in sections 4.2-4.4. To illustrate and assess the taxonomies we use the Oracle Applications ERP process repository.

### 4.1 The Descriptor Model

In the Process Descriptor Catalog model (“PDC”) [16] each activity is composed of one action, one object that the action acts upon, and possibly one or more action and object qualifiers. Qualifiers provide an additional description to actions and objects. In particular, a qualifier of an object is roughly related to an object state. State-of-the-art Natural Language Processing (NLP) systems, e.g., the *Stanford Parser*<sup>2</sup> can be used to automatically decompose process and activity names into *process/activity descriptors*. Each descriptor,  $d$ , can therefore

<sup>2</sup> <http://nlp.stanford.edu:8080/parser/index.jsp>



be represented as a tuple  $d = (o, oq, a, aq)$ , where  $o$  is an object,  $oq$  is a set of object qualifiers,  $a$  is an action, and  $aq$  is a set of action qualifiers.

For example, in Fig. 4 the activity “Notify the complaining customer of the treatment status” generates the following activity descriptor: (“customer”, “complaining”, “notify”, “of the treatment status”).

We denote by  $A$  the set of all actions (including qualifiers) in a process repository,  $G$ . Similarly,  $O$  denotes the set of all objects. We also denote by  $a(d)$  the action part of the descriptor, including the action qualifiers, e.g., “notify of the evaluation status” in the example. Similarly,  $o(d)$  denotes the object part.

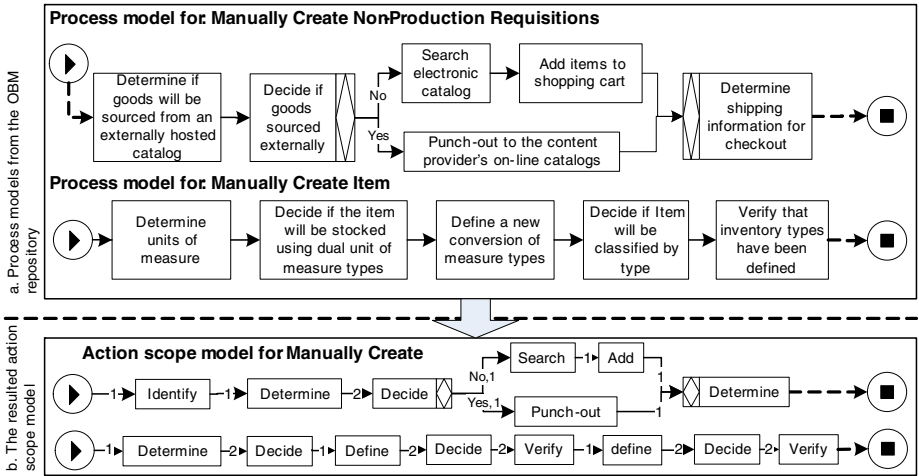
## 4.2 The Action Scope Model (ASM)

The action scope model is a graph  $ASM = (V_{ASM}, E_{ASM})$ , which represents the relationship between an action in a process name (a *primary action*) and the actions in its corresponding static model segment. As such, it represents an operational meaning of primary actions in the repository. Recall that a process repository consists of pre-defined process segments. We use this segmentation for learning about the scope of actions in the following way. Each action in the repository is related with a set of directional graphs of actions that represent the order of actions within this primary action’s segments. Therefore,  $V_{ASM}$  is a set of descriptor actions that are found in the segments of the primary action. An edge in  $E_{ASM}$  connects two actions that appear sequentially in the process model. Since such a primary action can be part of more than one process name, and since the same action may be represented more than once in the same process model segment - each edge in the action scope model is labeled with its weight, denoted  $w(e)$ , calculated by the number of its repetitions in the related process model segments. Graph splits are also represented in the action scope model. We denote by  $create_{ASM}(PM)$  the action of creating an ASM from a given static model segment,  $PM$ .

As a motivation for using this model in the context of searching process models, we analyzed 17 real-life processes from the Oracle Business Model (OBM) (we elaborate on the experiment setup in Section 6). Based on this sample we calculated a *tf-idf* measure [18] for each action as follows. For a group of processes that share the same primary action, we count the number of times each action appears in the group (serves to compute *tf*) and the number of different processes in the repository (processes with a different primary action) where this action appears overall (serves to compute *idf*). For example, out of the 17 processes, five contain the primary action “Create,” including 49 activities altogether. The action “Define” is presented five times in the set of all actions: four times in the “Create” processes, and one time in a different process. Therefore, its *tf-idf* is  $(4/49) * (\log(13/(1 + 2))) = 0.052$ . Note that the five “Create” processes are taken into account as one large process for this calculation.

In order to examine the quality of this value, we compare it to a random *tf-idf* values as follows. When assuming a random distribution of activities over processes, an activity that repeats  $n$  times in the repository has an expected  $n/k$  appearances in each process, assuming  $k$  processes. Therefore, given  $m$  processes

that share the same primary action, we expect an activity to appear  $n*m/k$  times in these processes, and the  $tf-idf$  computation is done accordingly. Continuing with the previous example, with five appearances of “Define” in the repository and 17 processes, we expect “Define” to appear  $5/17$  times in a process and  $5 * 5/17 = 1.47$  times in “Create” processes and 3.53 times elsewhere. This results in a  $tf-idf$  of 0.01. When applying such a calculation to all actions in all the 17 processes, it was found that on average, their  $tf-idf$  is 4.6 times better than the random value. We are therefore encouraged to learn about participating actions in a process segment given a primary action.



**Fig. 3.** A segment of the action scope model for the action “Manually Create” in the OBM repository for the Procurement category

Consider the following two processes from the OBM repository: “Manually Create Non-Production Requisitions” and “Manually Create Item.” These processes are represented in the OBM by corresponding graph segments as illustrated in Fig 3a. Using these two process models, it is possible to generate an action scope model for the action “Manually Create” (Fig 3b). The dashed lines in this illustration represent graph parts that are not shown due to space limitations. According to this example, there are two optional action paths compatible to the “Manually Create” action starting by either “Identify” or “Determine.” Since “Decide” follows “Determine” twice in this model, the respective edge weight is set to 2.

Given an action scope model graph,  $ASM$ , the intensity of any path  $i$  of length  $n - 1$ ,  $(a_{i,1}, a_{i,2}, \dots, a_{i,n})$  in  $ASM$ ,  $l_i^{ASM}$ , is computed to be:

$$l_i^{ASM} = 1 - \frac{n-1}{\sum_{j=1}^{n-1} w(a_{i,j}, a_{i,j+1})}$$

The values of  $\iota_i^{ASM}$  are in  $[0,1]$ , with higher values representing a stronger, more common  $ASM_i$ . For example, the intensity of the second path in the action scope model for “Manually Create” (Fig. 3b) is calculated as follows:

$$\iota_2^{ASM} = 1 - \frac{8}{1+2+1+2+2+1+2+2} = 0.385.$$

We define *fitness*,  $f^{ASM}$ , between a segment,  $PM$ , and an  $ASM$  to be zero if there is no path in the  $ASM$  that represents the  $PM$ . If such path exists, the *fitness* value is determined by its intensity. Formally, we define *fitness* as follows:

$$f^{ASM}(PM, ASM) = \begin{cases} \iota_i^{ASM} & \exists ASM_i \in ASM \mid create_{ASM}(PM) = ASM_i \\ 0 & otherwise \end{cases}$$

### 4.3 The Object Grouping Model (OGM)

The object grouping model represents the relationship between an object (*primary object*) and the objects in its corresponding static model segment. As such, it represents an *operational* meaning of primary objects in the repository. Since such a primary object can be part of more than one process segment, and since the same object may be represented more than once in the same process model segment - each edge in the object grouping model is labeled with its weight calculated by the number of its repetitions in the related process model segments. Therefore, an object grouping model of an object,  $o$ , denoted as  $OGM(o)$  is a bag (repetitions allowed)  $\{o_1, o_2, \dots, o_k\}$ .

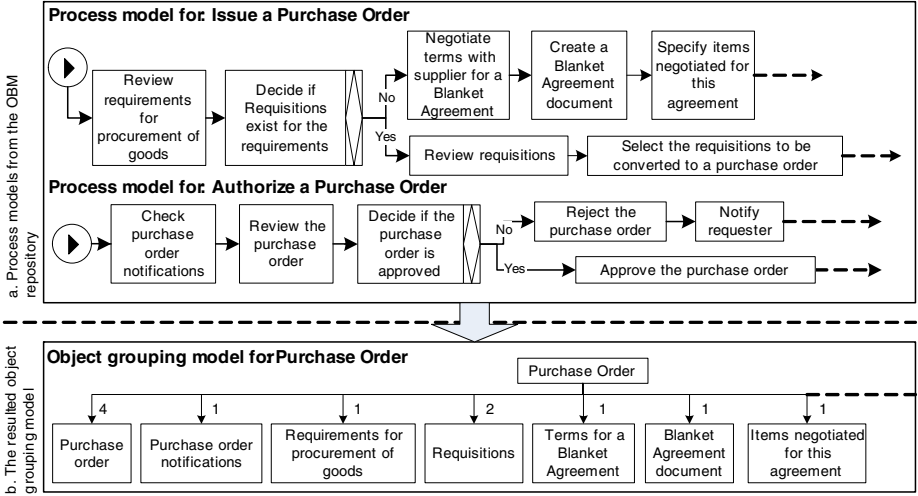
As a motivation for this model usage in the context of this work, and similarly to the *tf-idf* calculations made for assessing the relevance of using the  $ASM$ , we calculated the *tf-idf* measure for each object in each group of processes that share the same primary object. As a result, we found that on average, their *tf-idf* is 5.3 times better than the random value. We are therefore encouraged to learn about participating objects in a process segment given a primary object.

To illustrate, consider two processes from the OBM repository: “Issue a Purchase Order” and “Authorize a Purchase Order.” These processes are represented in the OBM by corresponding graph segments as illustrated in Fig 4a. Using these two process models, it is possible to generate an object grouping model for the object “Purchase Order,” as illustrated in Fig 4b. According to this example, process models that are related to the object “Purchase Order” deal also with objects such as “Purchase Order Notifications,” “Blanket Agreement Document,” and “Requisitions,” while the last option’s weight equals 2 since it is represented twice in the two input process models.

Given a segment,  $PM$ , (possibly a result of a user’s search phrase) we calculate its proximity,  $\lambda^{OGM}$ , to a given object grouping model,  $OGM$ , as follows. We denote by  $OPM$  the set of all objects in  $PM$ . We compute precision and recall on  $OPM$  and  $OGM(o)$  as follows:

$$P(PM, OGM(o)) = \frac{|OPM \cap OGM(o)|}{|OPM|}; R(PM, OGM(o)) = \frac{|OPM \cap OGM(o)|}{|OGM(o)|}.$$

$PM$  is considered more similar to  $OGM(o)$  as both  $P(PM, OGM(o))$  and

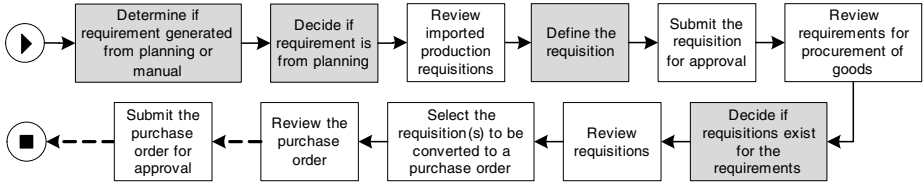


**Fig. 4.** A segment of the object grouping model for “Purchase Order” in the OBM repository for the Procurement category

$R(PM, OGM(o))$  are higher. Therefore we define  $\lambda^{OGM}(PM, OGM(o))$  as the harmonic mean of these figures:

$$\lambda^{OGM}(PM, OGM(o)) = \frac{P(PM, OGM(o)) * R(PM, OGM(o))}{P(PM, OGM(o)) + R(PM, OGM(o))} * 2$$

$\lambda^{OGM}$  ranges between  $[0,1]$ , and higher values represent a stronger match between the given  $PM$  and  $OGM(o)$ . It is also possible to define a threshold,  $th^{OGM}$ , below which  $\lambda^{OGM}$  is set to zero.



**Fig. 5.** An example of a process model for “Manually create purchase order”

For example, let us examine a given process model  $PM$  that represents the flow of activities for “Manually create purchase order,” as illustrated in Fig. 5. The object grouping model for “Purchase Order,” (Fig. 4) and  $PM$  share three objects in common: “Purchase order,” “Requisition,” and “Requirements for procurement of goods,” that repeat in  $PM$  activities once, twice, and once again, respectively (see Fig. 6). Both models consist of 11 objects, each. Therefore,

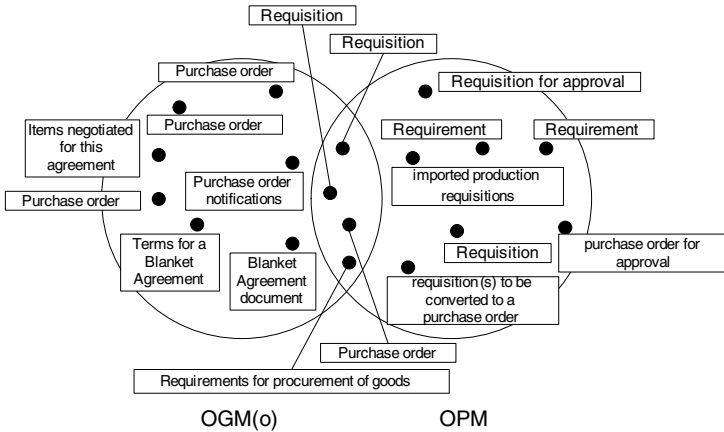


Fig. 6. An example of proximity calculation

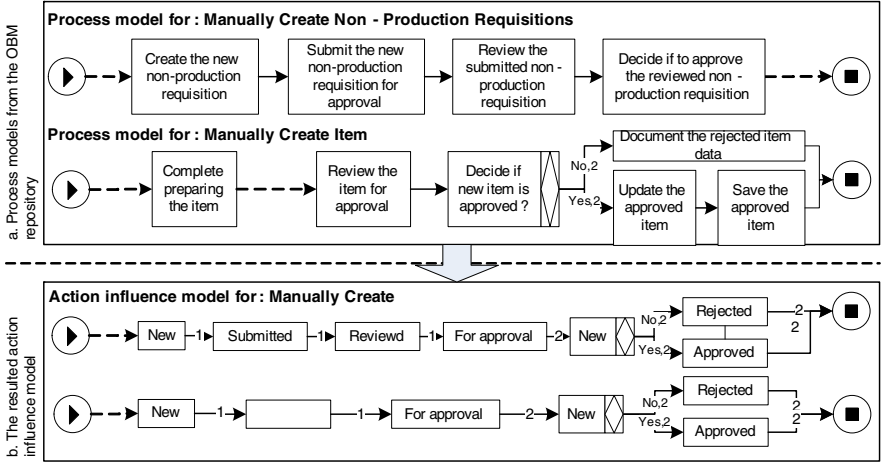
precision and recall are both  $4/11$  (single and plural forms of the same object are considered to be the same) and the proximity in our example is calculated to be 0.18.

#### 4.4 The Action Influence Model (AIM)

The action influence model represents the relationship between a primary action and the flow of states (object qualifiers) of the primary object in static model segments that correspond to the primary action. As such, it reflects the influence of a primary action on the way a primary object changes. Each edge in the action influence model is labeled with its weight representing the number of its repetitions in the related process model segments.

As a motivation for this model usage in the context of this work, and similarly to the calculations made for the *ASM*, we calculated the *tf-idf* measure for each two adjacent object-qualifiers (object-qualifier pairs) in any of the static model segments that share the same primary action. We found that on average, their *tf-idf* is 3.6 times better than the random value. We can therefore deduce that it is possible to learn about participating object-qualifier pairs in a process segment given a primary action.

To illustrate, consider the two process models named: “Manually Create Non-Production Requisitions” and “Manually Create Item.” They both deal with *manual creation*, but focus on different objects, “Non-Production Requisitions” and “Item.” Their initial part is illustrated in Fig. 3a and continued in Fig. 7a. By following changes to the qualifiers of the primary object in these process models we end up with the action influence model for “Manually Create” as illustrated in Fig. 7b. In this example, both primary object states change from “For approval” to “New” and then to “Rejected” or “Approved” in their corresponding process models and therefore the corresponding edges in the action influence model are labeled with weight of 2. In addition, we note that one of the



**Fig. 7.** A segment of the action influence model for the action “Create” in the OBM repository for the Procurement category

qualifiers in the action influence model is represented by an empty rectangle since its corresponding activity included the object “Item” as is, without any qualifiers. Also, it is worth noting that the first two activities in the process model for “Manually Create Non-Production Requisitions” represent the object “New Item.” Nevertheless, the corresponding qualifier “New” is represented only once at the beginning of the action influence model, since no change has been made to the object “Item” when advancing the process between these two activities.

Given a process model,  $PM$ , it is possible to calculate its similarity,  $\sigma_i^{AIM}$ , to a given path in an action influence model,  $AIM_i$ , using one of the state-of-the-art methods for assessing similarity between process models (e.g. [23],[8],[21]). For that purpose, an action influence model is created for the primary object of  $PM$  (referred to as a temporary action influence model,  $TAIM$ ) and then compared to  $AIM_i$ .  $\sigma_i^{AIM}$  can be then normalized to the range of  $[0,1]$ . On top of this score for each path in  $AIM$ , we also add an additional score that reflects the weights of the matched edges, so that the proximity between  $PM$  and  $AIM$ ,  $\lambda^{AIM}$ , is calculated as follows:

$$\lambda^{AIM}(PM, AIM) = \frac{\sum_{i=1}^k (\sigma_i^{AIM} + \frac{\sum_{e \in AIM_i \cap TAIM} w(e)}{\sum_{e \in AIM_i} w(e)})}{k} * \frac{1}{2},$$

where  $w(e)$  is the weight assigned to  $e$  in  $AIM_i$ , and  $k$  is the number of paths in  $AIM$ . Note that since the additional score refers only to edges, it can be zero even when  $\sigma_i^{AIM} > 0$  (e.g. when some activity names are matched) and therefore it is not multiplied, but added to  $\sigma_i^{AIM}$ .

$\lambda^{AIM}$  ranges between  $[0,1]$ , where higher values reflect a higher proximity of the given  $PM$  and  $AIM$ . It is also possible to define a threshold,  $th^{AIM}$ , in a similar manner as  $th^{OGM}$ .

Following our example, we first generate a temporary *AIM* for “Manually Create” based on the example process model graph (Fig. 5). Since the primary object “Purchase Order” is presented twice in this process model, at first without qualifiers, and then with the qualifier “For approval,” the temporary *AIM* contains these two qualifiers, as illustrated in Fig. 8.

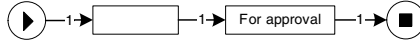


Fig. 8. An example of a temporary *AIM*

At the next phase of our example, we calculate the similarity between the temporary *AIM* and each of the two object qualifier paths presented in the action influence model for “Manually Create” (Fig. 7b). To do that we use the similarity method presented in [23]. There is a full match (similar text) of one node (“For approval”) between the temporary *AIM* and the first path in the *AIM* for “Manually Create,” and a full match of two nodes between the temporary *AIM* and the second path in the *AIM* for “Manually Create.” Therefore, the similarity score that these matches contribute is 1 and 2 for the first and second path, respectively. Since the optimal match between the temporary *AIM* and each of the paths in the *AIM* for “Manually Create” is a situation in which the entire temporary *AIM* is fully matched, the optimal similarity score is 2. Therefore, the similarity score between the temporal *AIM* and the first and second paths is  $\sigma_1 = \frac{1}{2}$  and  $\sigma_2 = \frac{2}{2} = 1$ , respectively. In addition, there are no matched edges between the temporary *AIM* and the first path (since the node labeled “For approval” is not directly connected to the start or end nodes). Nevertheless, there is one matched edge between the temporary *AIM* and the second path - that directly connects the node with no qualifiers with the “For approval” node. This edge’s occurrence is 1 and its weight is 1, resulting in a score addition of  $\frac{1}{12}$  to the score of the second path. This addition is divided by the maximal number of matched edges (3 in this example). As a result, the final similarity score is:  $\lambda^{AIM}(PM, AIM) = \frac{(\frac{1}{2}+0)+(\frac{1}{12})}{2} * \frac{1}{2} = 0.4$ .

## 5 The Process Model Search Problem and Method

This section describes the process model search method. We first provide a formal description of the search problem (Section 5.1). Then, we describe in Section 5.2 the use of the three taxonomies of Section 4 in a search procedure.

### 5.1 The Process Model Search Problem

Let  $G$  be a graph that represents a process repository. Let  $\lambda$  be a benefit model  $\lambda : \mathcal{G}, A, O \rightarrow [0, 1]$  and let  $S$  be a search phrase, given either as a descriptor or in natural language that is converted into a descriptor (see Section 4.1).

Given  $S$  and a segment  $G'$  we define  $\lambda$  as follows:

$$\lambda(G', a(S), o(S)) = f(G', ASM(a(S)) * \lambda^{OGM}(G', OGM(o(S))) * \lambda^{AIM}(G', AIM(a(S))) \quad (1)$$

Given  $S$ ,  $G$ , and  $\lambda$  find a segment,  $G'$ , that best fits the search phrase. The process model search problem can be formally defined as follows:

*Problem 1.* Given  $G$ ,  $S$ , and  $\lambda$ , find  $G' \in \mathcal{G}$  s.t.  $G' = \operatorname{argmax}_{G'' \in \mathcal{G}} \lambda(S, G'')$

In what follows we actually find the top  $k$  segments, ranked according to their operational relevance to the searched phrase. This way we expand the retrieved result range, allowing users to examine more than one result that may also contain useful information with regards to the search problem.

## 5.2 The Process Model Search Method

The process model search method (PMSM) relies on an underlying process descriptor analysis model and dynamically segments a business process repository to fit a given search phrase. We use the search request “*Manually Create a Purchase Order*” to illustrate the suggested method.

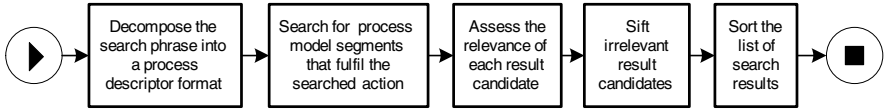


Fig. 9. The process model search method

**Method Overview.** The search procedure is composed of five main phases as follows (see illustration in Fig. 9). At first, it receives as input the name of a required process model in natural language or as a process descriptor. For the former, the input is automatically decomposed into a process descriptor format using NLP systems such as the *Stanford Parser*. According to this phase, the search phrase in our example will be transformed into the following process descriptor: object=“order,” action=“create,” object qualifier=“purchase,” action qualifier=“manually.” If more than one action and one object appear in the search phrase, it is automatically interpreted as separate descriptors (expressing all possible combinations of the given descriptor components) that are separately searched. Handling search phrases that include multiple process names as a unified process group is a topic for future work.

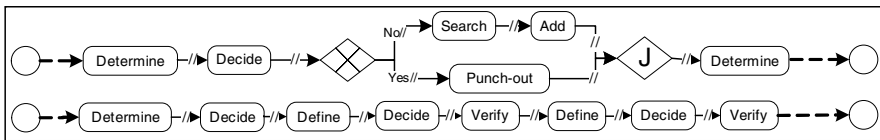
Based on the process descriptor input (the “target descriptor”), the PMSM searches first for all process model segments within the process repository that can be relevant candidates for the search result. This dynamic segmentation phase is the most important phase in this method since it enables the discovery of process models that do not have a pre-defined static segment in the repository, but are rather a part of or a sequential collection of pre-segmented process models (see Section 3.2).



At the next phase each process model option is assessed according to three orthogonal measures that reflect its relevance to the search request using proximity to the action scope model, the object grouping model, and the action influence model. According to these three relevance measures, non-relevant process models are removed from the option list, and finally the remaining process model options are being sorted according to a weighted grade of the three relevance measures. The sorted process model segment list is presented to the requester - as the search result for her request.

**Phase 1: Dynamic Segmentation.** The goal of this phase is to retrieve all process model segments that fulfill the target **action**. For example, given the target action “Manually Create,” the PMSM will search at this phase all process model segments that are aimed at *manually creating* an object of any type, without restricting the segment to the target object. Note that a naïve solution in this phase would be to examine all possible process model segments within the repository. Nevertheless, such algorithm can be highly inefficient (with  $n$  static segments and  $m$  the size of the biggest segment, there are  $2^{nm}$  induced dynamic segments over  $G$ ). Therefore, we reduce the collection of segments by selecting only relevant candidates at the first phase. Also note that this phase focuses on actions rather than objects as a basis for retrieving optional search results, since the search method is based on the operational meaning of the search and therefore all process model segments that relate to the search action represent a full set of optional results from which we select relevant results that are also related to the searched object.

To do that, the PMSM searches in the process repository for graph segments that are similar - both *structurally* and *textually* - to the action scope model of the target action. Since the action scope model is a graph of activities with partial names (only the action part is represented) - it is possible to convert it into a query statement using any state-of-the-art business process query mechanisms (e.g. [24]) and search for graph segments that are similar to it.



**Fig. 10.** Segments of *BPMN-Q* queries generated from the ASM for “Manually Create” in the OBM repository

In our example, the action scope model for “Manually Create” consists of two graphs (see Fig. 3). Each of these graphs will be converted into a query statement and be searched separately in the repository. A representation of those two graphs as a query statement using the *BPMN-Q* method suggested in [2] is illustrated in Fig. 10. The dashed lines in this illustration represent query parts that are not presented here due to space limitations. To relax the generated

query statements we mark all their edges with “//” - stating that there may be zero or more activities between each connected activities in the query result.

By running the two queries of our example on the OBM repository using the *BPMN-Q* method, 14 results were retrieved. One of those results is presented in Fig. 5. This example highlights the need for dynamic segmentation. This segment is combined sequentially from three process models in the OBM repository - starting with the last five activities of “Create Production Requisitions,” continuing with “Issue a Purchase Order” (Fig. 4a) and terminating with the first activities of “Authorize a Purchase Order” (also presented in Fig. 4a).

**Phase 2: Assessing the Relevance of Result Candidates.** This phase is aimed at determining the relevance of each result option retrieved at the previous phase to the search request. To do that, we use the three measures defined in Section 4 as follows.

1. Each result retrieved at phase 1 is compatible with one of the action paths in the action scope model, and therefore related to the intensity,  $i_j$  of this path (as detailed in Section 3).
2. We calculate each result’s  $\lambda^{OGM}$  (Section 4.3) as a measure for its proximity to the target object. To understand the necessity of this measure consider, for example, a process segment resulted at phase 1 that highly represents the “Manually Create” action (the order of its actions represents one of the action sequences in the *ASM* of the “Manually Create” action). Nevertheless, none of the objects involved in this process model participates in process models related to the object “Purchase Order.” In this case we can deduce that this segment is only loosely related to the primary object in our example.
3. The proximity to the action influence model is aimed at preferring results that express a *typical* modification of the primary object states as a result of applying the primary action. To measure the proximity between each result to the primary action’s influence model we use  $\lambda^{AIM}$  (Section 4.4).

**Phase 3: Sifting Irrelevant Result Candidates.** In this phase two thresholds,  $th^{OGM}$  and  $th^{AIM}$  (see sections 4.3 and 4.4), are used to determine the inclusion of each result candidate in the final result list. Note that it is not sufficient to apply a threshold on the final grade (as calculated in phase 4) to determine inclusion. A candidate may receive a relatively high final grade and still be irrelevant to the searched phrase. For example, a candidate may be highly similar to the search phrase’s action influence model but may not consist any relevant objects from its object grouping model. Therefore, we include in the final result list only results for which  $\lambda^{OGM} > th^{OGM}$  and  $\lambda^{AIM} > th^{AIM}$ .

**Phase 4: Sorting the List of Search Results.** At this phase a final grade,  $\lambda_i$ , for each result candidate,  $R_i$ , is calculated according to Eq. 11 using the grades of the three measures calculated in phase 2.

Following our example, the final grade for the optional result presented in Fig. 5 is:  $\lambda_i = 0.385 * 0.18 * 0.4 = 0.03$ . Finally, the optional process models are

sorted according to their final grade in an ascending order - from the closest to the most distant option as regards to the user's search phrase.

## 6 Experiments

We now present an empirical evaluation of the proposed method effectiveness. We first present our experimental setup and describe the data that was used. Then, we present the experiment results and provide an empirical analysis of these results.

**Data Set and Experiment Setup.** We chose a set of 17 real-life processes from the Oracle Business Model (OBM)<sup>3</sup> comprising 152 activities altogether. Using these processes we created a "process repository database," that includes the 17 process models and their derived taxonomies.

To evaluate the suggested method we conducted 17 experiments. At each experiment, a single process was removed from the database and then was searched according to its name. More specifically, each experiment was conducted according to the following steps: (a) preparation: removal of one of the process names from the database and reconstruction of the taxonomies so that the database will contain the process model (its graph segment) but the three taxonomies will not contain any of its descriptor components; (b) search for the removed process in the database, using its name as the search phrase.

The similarity between the result's temporal *AIM* and each of the object qualifier paths in the primary action's *AIM* (see Section 4.4) was calculated using the method in [23]. In addition, the threshold parameters for sifting irrelevant result candidates (defined in Section 5.2) were set to:  $th^{OGM} = 0.1$ ,  $th^{AIM} = 0.2$ .

The search results have then been objectively evaluated with respect to the original, removed, process. For each of the 17 experiments we also chose the "best result," the one most similar to the goal process model, calculated using the similarity method presented in [23]. Our metrics for measuring the effectiveness of the method, as detailed below, assess both the quality of an average result, as well as the average quality of the best result in each experiment - showing the best performance of the search method.

We use five metrics to measure the effectiveness of the method. For all results and for the best result we compute: (1) the average percentage of correct activities, showing how similar the results are with respect to the goal, correct result; (2) the average percentage of redundant activities, showing the amount of redundant, unnecessary activities in the retrieved results. Finally, we identify the average location of the best result in the list of search results - showing the effectiveness of the grading mechanism, that aims at locating the best result at the beginning of the result list.

**Results and Analysis.** On average, each result contained 81.7% of the goal process model activities. On average, the best result has a higher overlap of

<sup>3</sup> <http://www.oracle.com/applications/tutor/index.html>

92.9% with the goal process model and was ranked in a high location in the list of results (1.2), usually in the first place. In addition to the correct activities, the results also contained, on average, 14.2% of redundant activities out of the goal process model, while this percentage was lower (8.0%) in the best result.

These experiments have shown the usefulness of using a descriptor repository in searching for business process models based on their operational meaning. We also showed the method to be effective in the given experimental setup, both in terms of the similarity between the results and the goal result and with respect to the ranking of the best result.

## 7 Conclusions

We proposed a mechanism to automate the search of process models that saves search time and supports non-expert users in searching for business process models in a process repository using their own terminology of the process goal. By analyzing a process repository we can automatically create three taxonomies with which it is possible to process the search intention in operational terms. We provide a formal model of the problem, a method for providing a ranked response to a user's request, and we show the method to be empirically effective, using the Oracle ERP Business Process Model, as a testbed.

The proposed method and experiments provide a starting point that can already be applied in real-life scenarios, yet several research issues remain open. We mention four such extensions here. First, extending the empirical study to further examine the quality of retrieved search results aiming at improving and fine-tuning the method. Second, supporting search phrase relaxations to retrieve more result options in cases where the search phrase retrieves only few or no results. Third, extending the method for supporting synonyms to extend the repository vocabulary. Forth, supporting multiple descriptors in a search phrase.

## References

1. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases—an introduction. *Natural Language Engineering* 1(01), 29–81 (1995)
2. Awad, A.: BPMN-Q: A Language to Query Business Processes. In: EMISA 2007, vol. 119, pp. 115–128 (2007)
3. Awad, A., Polyvyanyy, A., Weske, M.: Semantic querying of business process models. In: 12th International IEEE Enterprise Distributed Object Computing Conference, pp. 85–94. IEEE (2008)
4. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes with BP-QL. *Information Systems* 33(6), 477–507 (2008)
5. Belhajjame, K., Brambilla, M.: Ontology-based description and discovery of business processes. *Enterprise. Business-Process and Information Systems Modeling*, 85–98 (2009)
6. Bozzon, A., Brambilla, M., Fraternali, P.: Searching Repositories of Web Application Models. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 1–15. Springer, Heidelberg (2010)

7. Coalition, W.M.: The workflow management coalition specification - terminology & glossary. Technical report, Technical Report WFMC-TC-1011, Workflow Management Coalition (1999)
8. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proceedings of the fourth Asia-Pacific Conference on Conceptual Modelling, APCCM 2007, pp. 71–80. Australian Computer Society, Inc., Darlinghurst (2007)
9. Goderis, A., Li, P., Goble, C.: Workflow discovery: the problem, a case study from e-Science and a graph-based solution (2006)
10. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked keyword search over XML documents. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 16–27. ACM (2003)
11. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 305–316. ACM (2007)
12. Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword proximity search on XML graphs (2003)
13. Katz, B., Lin, J., Quan, D.: Natural language annotations for the Semantic Web. In: Meersman, R., Tari, Z. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 1317–1331. Springer, Heidelberg (2002)
14. Li, Y., Yang, H., Jagadish, H.V.: NaLIX: A generic natural language search environment for XML data. ACM Transactions on Database Systems (TODS) 32(4), 30 (2007)
15. Lincoln, M., Golani, M., Gal, A.: Machine-Assisted Design of Business Process Models Using Descriptor Space Analysis. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 128–144. Springer, Heidelberg (2010)
16. Lincoln, M., Karni, R., Wasser, A.: A Framework for Ontological Standardization of Business Process Content. In: International Conference on Enterprise Information Systems, pp. 257–263 (2007)
17. Markovic, I., Pereira, A.C., Stojanovic, N.: A framework for querying in business process modelling. In: Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI), Munchen, Germany (2008)
18. McGill, M.J., Salton, G.: Introduction to modern information retrieval. McGraw-Hill (1983)
19. Momotko, M., Subieta, K.: Process query language: A Way to Make Workflow Processes More Flexible. In: Benczúr, A.A., Demetrovics, J., Gottlob, G. (eds.) ADBIS 2004. LNCS, vol. 3255, pp. 306–321. Springer, Heidelberg (2004)
20. Shao, Q., Sun, P., Chen, Y.: WISE: a workflow information search engine. In: IEEE 25th International Conference on ICDE 2009, pp. 1491–1494. IEEE (2009)
21. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.T.: Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 129–144. 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. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring Similarity Between Business Process Models. In: Bellahsene, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008)
24. Yan, Z., Dijkman, R., Grefen, P.: Business Process Model Repositories-Framework and Survey. Technical report, Beta Working Papers

# Fragment-Based Version Management for Repositories of Business Process Models

Chathura C. Ekanayake<sup>1</sup>, Marcello La Rosa<sup>1</sup>,  
Arthur H.M. ter Hofstede<sup>1,2</sup>, and Marie-Christine Fauvet<sup>3</sup>

<sup>1</sup> Queensland University of Technology, Australia  
{c.ekanayake,m.larosa,a.terhofstede}@qut.edu.au

<sup>2</sup> Eindhoven University of Technology, The Netherlands

<sup>3</sup> University of Grenoble, France  
marie-christine.fauvet@imag.fr

**Abstract.** As organizations reach higher levels of Business Process Management maturity, they tend to accumulate large collections of process models. These repositories may contain thousands of activities and be managed by different stakeholders with varying skills and responsibilities. However, while being of great value, these repositories induce high management costs. Thus, it becomes essential to keep track of the various model versions as they may mutually overlap, supersede one another and evolve over time. We propose an innovative versioning model, and associated storage structure, specifically designed to maximize sharing across process models and process model versions, reduce conflicts in concurrent edits and automatically handle controlled change propagation. The focal point of this technique is to version single process model fragments, rather than entire process models. Indeed empirical evidence shows that real-life process model repositories have numerous duplicate fragments. Experiments on two industrial datasets confirm the usefulness of our technique.

## 1 Introduction

Organizations need to develop process models to document different aspects of their business operations. For example, process models are used to communicate changes in existing operations to relevant stakeholders, document procedures for compliance inspection by auditors or guide the development of IT systems [31]. Such process models are constantly updated to suit new or changed requirements, and this typically leads to different versions of the same process model. Thus, organizations tend to accumulate large numbers of process models over time [25]. For example, Suncorp, one of the largest Australian insurers, maintain a repository of 6,000+ process models [24], whereas the Chinese railway company CNR has 200,000+ models.

The requirement to deal with an increasing number of process models within organizations poses a maintenance challenge. Especially, it becomes essential to keep track of the various models as they may mutually overlap, supersede one another and evolve over time. Moreover, process models in large organizations are typically edited by stakeholders with varying skills, responsibilities and goals, sometimes distributed across independent organizational units [7]. This calls for techniques to efficiently store process models and manage their evolution over time.

In this paper, we propose a novel *versioning model* and associated *storage structure* which are specifically designed for process model repositories. The main innovation lies in storing and versioning single process fragments (i.e. subgraphs), rather than entire process models. In this way duplicate fragments across different process models, or across different versions of the same process model, are stored only once. In fact, empirical evidence [40] shows that industrial process model collections feature a high number of duplicate fragments. This occurs as new process models are created by copying fragments from existing models within the same collection. For example, we identified nearly 14% of redundant content in the SAP R/3 reference model [20]. Further, when a new process model version is created, only a subset of all its fragments typically changes, leaving all other fragments unchanged across all versions of the same model.

Besides effectively reducing the storage requirements of (large) process model repositories, our technique provides three benefits. First, it keeps track of shared fragments both *horizontally*, i.e. across different models, and *vertically*, i.e. across different versions of the same model. As a result, this information is readily available to the repository users, who can monitor the various relations among process model versions. Second, it increases *concurrent editing*, since locks can be obtained at the granularity of single fragments. Based on the assumption that different users typically work on different fragments at the same time, it is no longer necessary to lock an entire process model, but only those fragments that will actually be affected by a change. As a result, the use of traditional conflict resolution techniques is limited to situations in which the same fragment is edited by multiple users concurrently. Finally, our technique provides sophisticated *change propagation*. For example, if an error is detected in a shared fragment, the fix can be automatically propagated to all process models containing that fragment, without having to edit each process model individually. This in turn can facilitate reuse and standardization of best business practices throughout the process model repository. To the best of our knowledge, the use of process fragments for version control, concurrency control (i.e. locking) and change propagation of process model collections has not been studied in existing research. Commercial BPM suites only offer propagation of attribute changes at the node level, e.g. a label change.

The proposed technique is independent of the process modeling language being adopted as all the developed methods operate on an abstract modeling notation. Thus, we can manage processes modeled in a variety of languages, e.g. BPMN, EPCs, BPEL, YAWL. We implemented this technique on top of the MySQL relational DBMS and used the prototype to conduct experiments on two industrial process model collections. The results show that the technique yields a significant gain in storage space and demonstrate the usefulness of its locking and change propagation mechanisms.

We present our technique in three steps. First, we introduce the versioning model in Sec. 2. Next, we describe our locking mechanism in Sec. 3 and finally our controlled changed propagation in Sec. 4. In Sec. 5 we discuss the storage structure used to implement our technique on top of relational DBMSs. We present the experimental setup and results in Sec. 6 and discuss related work in Sec. 7. We draw conclusions in Sec. 8.

## 2 Versioning Model

We define process model versions according to a branching model which is inspired by popular version-control systems such as Concurrent Version Systems (CVS) [5] and

Apache Subversion (SVN)<sup>1</sup> Accordingly, each process model can have one or more *branches* to account for co-existing developments. Each branch contains a sequence of process versions and has a unique name within a process model.

A new branch can be created by “branching out” from a version in another existing branch, where the existing branch may belong to the same process model (*internal branching*) or to another process model (*external branching*). The *primary* branch is the first branch being created for a process model, and as such it can be new or be derived via external branching. Non-primary branches of a process model can only be derived via internal branching. Only the last version of a branch, namely the *current version* can be modified.

A modification to a current version produces a new version in the same branch which becomes the current version. According to this versioning model, a specific version of a process model is referred to by the tuple (process model name, branch name, version number). Optionally, a version may have a name which needs not be unique. This model is shown in Fig. 1 by using an example from the insurance domain. Here the primary branch is new and named “Home”, whereas “Motor”, “Private” and “Commercial” are all secondary branches. For example, version 1.0 of the Motor branch, named “alpha”, is derived from version 1.1 of the Home branch, named “signed”.

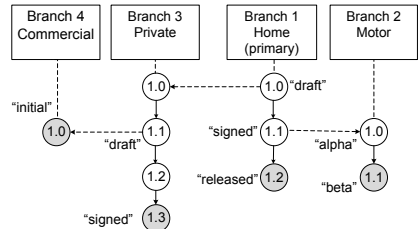


Fig. 1. Process model versioning (current version of each branch is shaded)

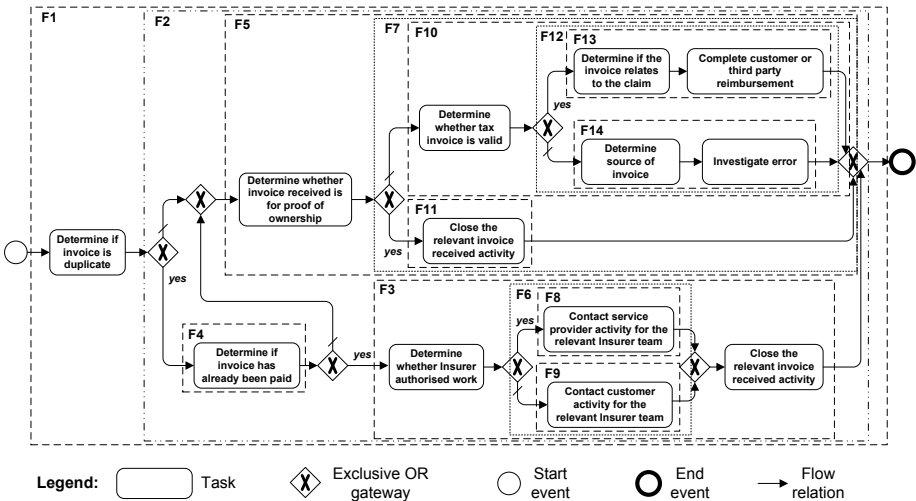


Fig. 2. Version 1.0 of the Home insurance claims process model, and its RPST fragments

<sup>1</sup> <http://subversion.apache.org>



The focal idea of our versioning model is to use process model fragments as storage units. To obtain all fragments from a process model, we use the Refined Process Structure Tree (RPST) [41]. The RPST is a linear-time method to decompose a process model into a tree of hierarchical *SESE fragments*. A SESE fragment is a subgraph of a process model with a single entry and a single exit node. Each fragment in the hierarchy contains all fragments at the lower level, but fragments at the same level are disjoint. Thus, a given process model has only one RPST decomposition. The advantage of using SESE fragments is that they are modular: any change inside a fragment does not affect other fragments outside the modified fragment. Fig. 2 shows version 1.0 of the Home insurance claims process model, and its RPST decomposition. The notation is BPMN.

For each model, we store its SESE fragments with their composition relationships. A fragment may contain one or more child fragments, each of which may also contain child fragments, forming a tree structure. Fig. 3 shows the fragment version tree of the process model in Fig. 2

We maintain a version history for each fragment. Each fragment has a sequence of versions and the latest version is named as the *current* version. When a new fragment is added, its version sequence starts with 1 and is incremented by one for each subsequent version. Fig. 3 depicts fragments as rectangles and fragment versions as circles; version numbers are shown inside circles. As all fragments in this example are new, each fragment has version 1. Each process model version points to the root fragment version of its fragment version tree.

By using fragments as units of storage, we can efficiently support version control, change management and concurrency control for process models. Before describing how we realize such operations, we explain how a fragment is stored in the repository. Each fragment version needs to store its *composition relationships* and its *structure*. The composition relationships contain the identifiers of all the immediate child fragment versions. The structure of a fragment version is the subgraph of that fragment version where the subgraphs of all its child fragment versions are replaced by placeholders called *pockets*. Each pocket is associated with an identifier and within the structure of a particular fragment version, it points to one child fragment version. In this way we can maximize reuse across fragments, since two fragments can share the same structure but point to different child fragment versions from within their pockets. Fig. 4 shows the structure of fragment F2 from Fig. 2. This structure contains three child fragments, each represented by a pocket. In version 1 of F2, pocket 1 points to version 1 of F3, pocket 2 to version 1 of F4 and pocket 5 to version 1 of F5. Next, we describe how to reuse structures by mapping different child fragment versions to pockets.

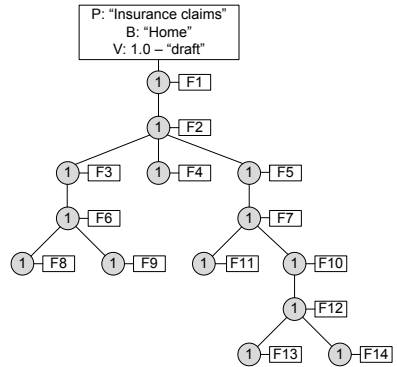


Fig. 3. RPST of model in Fig. 2

### 2.1 Vertical Sharing

Process models are not static artifacts but evolve with an organization. As we store individual fragments, all unmodified fragments can be shared across different versions

of the same process model. We call this *vertical sharing*. When a new version of a process model is created, only those fragments that have changed or that have been added are stored. Fig. 5 shows the derivation of version 1.1 from version 1.0 of the Home insurance claims process by modifying fragment F3.

Fragment F3 is modified by removing F6 and adding F25 and F32. This leads to a new version of F3 with the modified content (version 2). In addition, new versions of F2 and F1 need to be created with the modified composition relationships. All other fragments (i.e. F4 to F14) remain the same and are shared between version 1.0 and 1.1 of the Home insurance process.

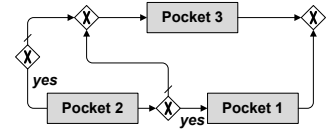


Fig. 4. Structure of fragment F2 from the model in Fig. 2

As we mentioned earlier, we reuse structures of fragments across subsequent fragment versions in order to avoid redundancy. For example, changing fragment F3 does not affect the structure of fragment F2. However, a new version of F2 has to be created to represent the modified composition relationships (i.e. replacement of version 1 of F3 with version 2). Thus, the structure can be shared across versions 1 and 2 of F2. Let us consider the

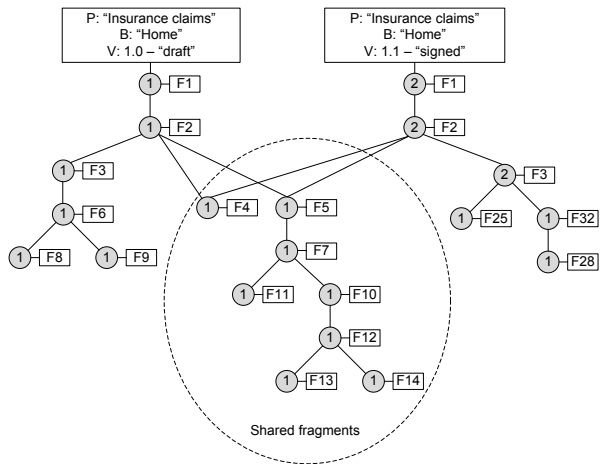


Fig. 5. Sharing fragments across multiple versions of the same process model

structure of version 1 of F2 as shown in Fig. 4. According to the example, version 1 of F2 maps version 1 of fragments F3, F4 and F5 to pockets 1, 2 and 3 respectively. In version 2 of F2, the structure does not change except for the mapping of pocket 1 which now points to version 2 of F3. Thus, we reuse the structure of version 1 of F2 in its version 2 simply by changing the mapping of its pocket 1.

## 2.2 Horizontal Sharing

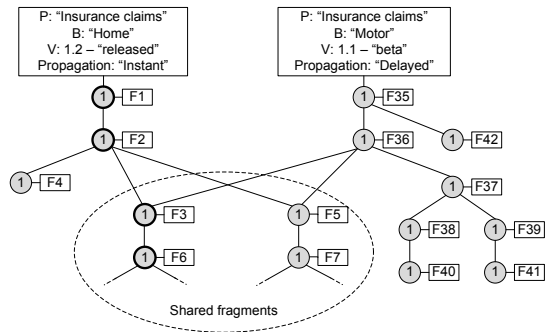
Real-life process model repositories hardly have unique process models. It is common in fact that multiple process models share common fragments. For example, we identified 840 duplicate fragments in the SAP reference model. In order to avoid such redundancy, we also allow fragment versions to be shared among multiple branches within or across process models. We call this *horizontal sharing*. By keeping track of such derivation relationships, we can efficiently propagate changes and keep the repository in a consistent state. As an example, Fig. 6 shows the relationship between version

1.2 of the Home insurance branch and version 1.1 of the Motor insurance branch, which share fragments F3 and F5, and their child fragments. Similar sharing relations can exist between branches of different process models.

### 3 Locking

If two or more users try to modify two overlapping sections within the same process model or across different process models, the resulting process model(s) may become inconsistent. The solution used by current process model repositories to avoid such conflicts is to lock an entire process model before editing it. However, such a solution limits the ability for collaboration, especially in light of the current trend for collaborative process modeling, as only one user can edit a process model at a time. We propose a fragment-based locking mechanism for process models which supports increased collaboration while reducing the number of conflicts.

Users can lock individual fragments, upon which, any subsequent locking requests to those fragments will be denied. When a lock is requested for a fragment, we need to consider the lock granted for that fragment, as well as the locks of its ancestor and descendant fragments. To illustrate this, let us assume that a user requests a lock for F3 in Fig. 6 and



**Fig. 6.** Sharing fragments across different process model branches

that a lock has already been granted for its child fragment F6. If the requested lock is granted for F3, both F3 and F6 can be edited concurrently. As F3 contains F6, the user editing F3 can also edit the content of F6, which may result in a conflict with the edits done by the other user on F6. Thus, in this situation a lock for F3 cannot be granted. The same situation holds for the ancestor fragments of F3. If any ancestor fragment of F3 (e.g. F2) is locked, a lock for F3 cannot be granted. Thus, a fragment can only be locked if a lock has not yet been granted for that fragment and for any of its ancestor or descendant fragments. For example, two users can lock F3 and F7 at the same time. Concurrent updates to these two fragments do not cause conflicts, as neither of these fragments contain the other fragment. In this case, any subsequent lock request for fragments F3 and F7, and for their descendant and ancestor fragments will be denied.

This fragment-based locking mechanism is realized by associating two locking attributes with each fragment: a boolean *direct lock* and an integer *indirect lock counter*. A direct lock is assigned to a fragment that is directly locked by a user and gives the user the actual right to edit that fragment. The indirect lock counter is used to prevent conflicting lockings to descendant fragments. It is set to zero and incremented by one every time a descendant of the fragment in question is directly locked. A direct lock can only be placed if a fragment is not directly locked, its indirect lock counter is zero

and none of its ancestor fragments is directly locked either. If so, the fragment is locked and the indirect lock counters of all its ancestors are incremented. Once a request for removing a lock is issued, the direct lock for that fragment is removed and the indirect lock counters of all its ancestor fragments are decremented. The indirect lock counter is required as multiple descendant fragments of a given fragment may be directly locked at the same time. In such situations, the counter of that fragment should not be reset until all direct locks of its descendant fragments have been released.

## 4 Controlled Change Propagation

In current process model repositories, similarity relations between different process models are not kept, so an update to a section of a process model remains confined to that process model, without affecting all process models of the repository that share (parts of) that section. This problem where two or more process models become “out-of-synch” is currently rectified manually, through maintenance cycles which are laborious and error-prone. For example, a team of business analysts at Suncorp was recently involved in a process consolidation effort between two of their insurance products, due to an update to one of the two products. However, it took them 130 man-hours to identify 25% of the shared fragments between the process models for these two products [24]. In fact, our experience tells us that real-life collections suffer from frequent mismatches among similar process models.

Since we reuse fragments across multiple process models, this provides a great opportunity to simplify the maintenance of the repository. For example, if a possible improvement is identified for fragment F3 of Fig. 6 that improvement can be made available immediately to both the Home and Motor insurance process models, since this fragment is shared by both these models. However, propagating fragment changes immediately to all affected process models may not be always desirable. Let us assume that the current version of the Motor insurance process model has been deployed in an active business environment. If an update to F3 has introduced an error, that error will immediately affect the Motor insurance process model, which could potentially impact important business operations. In order to prevent such situations, we support a flexible change propagation mechanism, where change propagations are controlled by a propagation policy associated with process model branches. The propagation policy of a process model branch can be set as either *instant propagation* or *delayed propagation*. If instant propagation is used in a branch, any change to any fragment in the current version of that branch is recursively propagated to all ascending fragments of that fragment in the current version, until the root fragment. Since the root fragment changes, a new version for that branch will be created, which will become the current version. If delayed propagation is used in a branch, changes to a fragment will not be immediately propagated throughout the current version. Instead, such changes will create pending updates for the current version. Then owners of the affected process model are notified of all pending updates for that model. They can then review the pending updates and only trigger the necessary ones. Once a pending update is triggered, it will be propagated and a new version of the interested process model will be created.

Coming back to the example in Fig. 6 let us assume that the change propagation policy of the Home insurance branch is set to *instant* while that of the Motor insurance

branch is set to *delayed*. If fragment F6 is updated (i.e. version 2 of F6 is created), new versions will instantly be created for all the ancestor fragments of F6 in the current version of Home (i.e. F3, F2 and F1, shown with a thicker border Fig. 6). As a new version is created for F1, which is the root fragment of Home, a new version of this process model will also be created, say version 1.3. On the other hand, since the Motor branch has a delayed propagation policy, new versions will not be created for the ancestor fragments of F6 in the current version of this branch. This means that F3 in Motor will still point to version 1 of F6, F36 to version 1 of F3 and F35 to version 1 of F36. Thus, the current version of Motor will still use version 1 of F6 and remain the same. However, the pending updates will be notified to the owner of the current version of Motor, who can decide whether or not to implement them.

Sometimes one may not need to create a new fragment version/process model version when a fragment is modified, e.g. after fixing a minor error. Our technique supports such in-place editing of fragments, where the edited fragment version and all its ancestor fragments are updated without creating new versions. Changes performed in this mode will be available to all ancestor fragments instantly, irrespective of the change propagation policies.

## 5 Conceptualization of the Storage Structure

We now describe the conceptual model used to store our versioning system on top of a relational DBMS. The algorithms to populate and use this data structure, e.g. inserting or updating a fragment, can be found in the technical report [15].

An Object Role Modeling diagram of the storage structure is shown in Fig. 7. For illustration purposes, we populated this model with information from two process models: “Insurance claims” (the example used so far) and “Order processing”. Each process has two branches (e.g. Insurance claims has branches “Home” and “Motor”). Further, each branch has a root process model (i.e. the root Node), representing the first version of that branch. For example, the root process model of the Motor branch of the insurance claims process has node identifier N4 and refers to version number 1.0 having version name “alpha”. Each branch has a sequence of nodes where each node represents one version of a process model. Each node can have at most one immediate predecessor. For example, node N5 refers to version number 1.1 of its branch, and is the successor of node N4. The root node of a primary branch may optionally be derived from a node of an external process model branch (none in the sample population). The root node of a non-primary branch is always derived from a node of an internal process model branch. For example, the root node of the Motor branch (node identifier N4) is derived from node N2 of the Home branch.

Each node in a branch (i.e. each process model version) has an associated fragment version tree. In our example, the root fragment versions of process model versions 1.0 and 1.1 of the Home branch (i.e. nodes N1 and N2) are FV1 and FV6. FV1 and FV6 are both contained in fragment F1 according to the sample population. Thus, FV1 and FV6 are two versions of the same fragment. In fact, FV1 is mapped to fragment version number 1 whilst FV6 is mapped to fragment version number 2 of F1. A fragment version can have multiple parents and children. For example, FV2 is the parent fragment of FV3, FV4 and FV5, while FV3 is the child of both FV2 and FV7. Hence, FV3 is

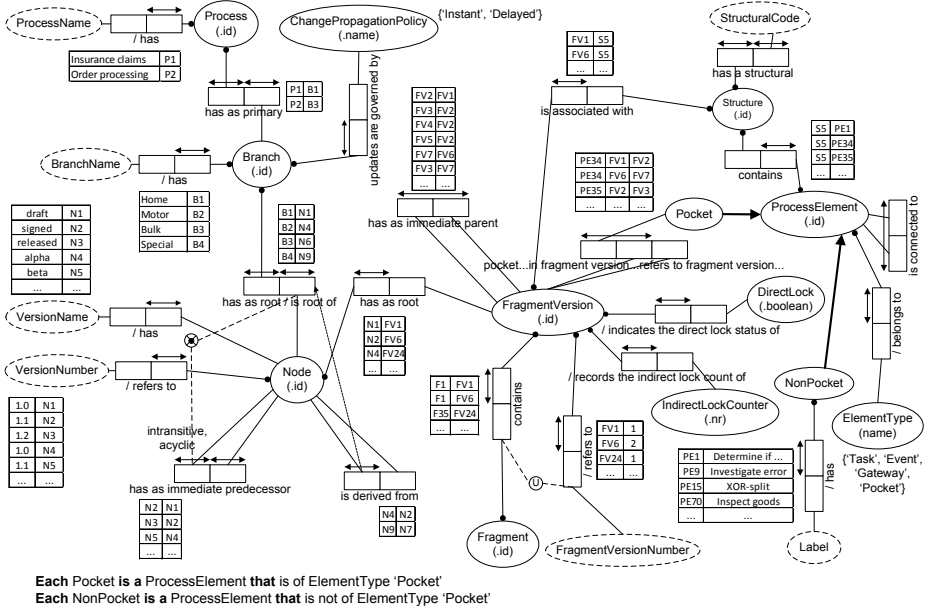


Fig. 7. Object-Role Modeling diagram of the storage structure

shared between FV2 and FV7. A fragment version is associated with a structure which stores all process elements contained only in that fragment version. A structure is associated with a structural code, which is computed by considering its elements and their interconnections. The structural code is used to efficiently compare structures of fragments. Furthermore, two fragments can be efficiently compared by considering both structural codes and composition relationships. Process elements within structures can be of type non-pocket (i.e. tasks, events, gateways) and pocket. A pocket is a placeholder for a child fragment. Continuing our running example, in fragment version FV1, pocket PE34 is mapped to fragment version FV2 while in FV6, PE34 is mapped to FV7. Thus, FV1 and FV6 share the structure S5 with different mapping for pocket PE34. Finally, the diagram models the association of change propagation policies with process branches and locking attributes with fragment versions.

As shown in the diagram of Fig. 7 we use a directed attributed graph of vertices (i.e. process elements) and edges (i.e. flow relations) to represent process models and fragments. Process elements can be tasks, events (e.g. timer or message events), gateways (e.g. AND-split, XOR-split, OR-join) and pockets. This meta-model is an extension of the canonical format used in the AProMoRe repository [25], where we introduced a new process element, namely the Pocket, to act as a placeholder for dynamically-computed child fragments. This abstract representation allows us to apply version control to process models developed in multiple business process modeling languages (e.g. BPMN, YAWL, EPCs, BPEL), as well as to facilitate change propagation and concurrency control on those process models, regardless of their modeling language. For example, in order to version EPC models, we only have to convert EPCs to our representation format and vice versa. Once EPC models are converted to our representation format, those

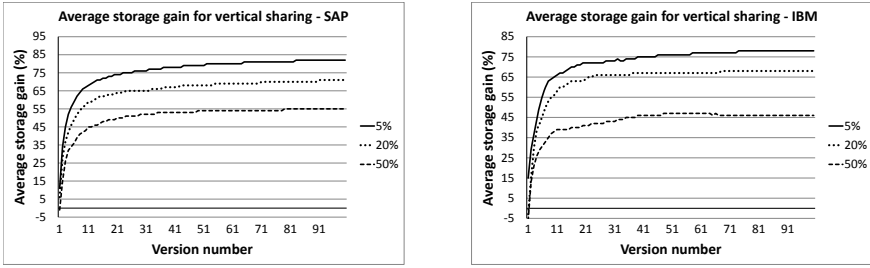
process models can be stored as tuples in the relational schema derived from Fig. 7. A full mapping between AProMoRe’s canonical format and various process modeling languages is provided in [25]. We observe that in order to achieve language-independence, AProMoRe’s canonical format covers only a set of concepts which are common to most process modeling languages.

## 6 Evaluation

We implemented the proposed versioning model and associated storage structure in Java on top of the MySQL DBMS, and used this prototype to evaluate our technique. We conducted the experiments on two industrial process model collections: 595 EPC models from the SAP R/3 reference model and 248 EPC models from IBM’s BIT library<sup>2</sup>. First, we measured the gain induced by vertical sharing. We took a set of models with varying size (ranging from 25 to 107 nodes for the SAP dataset and from 10 to 40 nodes for the IBM dataset), and for each of them we created 100 subsequent versions by randomly updating a set of adjacent nodes (i.e. localized changes). We allowed four types of basic change operations with corresponding probabilities: change task label (33%), delete task (33%), insert a task between two adjacent nodes (17%) and insert a task in parallel to another task (17%). These probabilities were chosen to balance insertions and deletions so as to prevent excessive growth or shrinkage of a process model, thus simulating localized changes. For each model, we repeated the experiment by changing 5%, 20% and 50% of the models’ size. After creating a new version, we calculated the vertical storage gain  $G_v$  compared to storing full process model versions. Let  $N$  be the number of nodes for storing full versions and  $N_v$  the number of nodes stored if sharing fragments vertically. Then  $G_v = (N - N_v) \cdot 100/N$ . Fig. 8 reports the average  $G_v$  for each dataset, by aggregating the values of all changed process models. Our technique incurs a slight initial overhead due to storing pockets and edges connecting pockets. However, the vertical storage gain rapidly increases as we add new versions. For the SAP dataset it levels off at 82% for small updates (5% of model size), and 55% for larger updates (50% of size) whilst for the IBM dataset it levels off at 78% for small updates and 46% for larger updates. This confirms our intuition that storing duplicate fragments only once across different process model versions can dramatically reduce the overall repository size.

Second, we measured the gain  $G_h$  induced by horizontal sharing. For each dataset, we randomly inserted all process models in the repository, and as we increased the size of the repository, we compared the size of storing duplicate fragments only once with the size of storing full process models. We only counted the size of maximal fragments across different process models, i.e. we excluded child fragments within shared fragments. Let  $N$  be the number of nodes for storing full process models,  $F$  the set of fragments,  $N_f$  the number of nodes of fragment  $f$  and  $O_f$  the number of its occurrences. Then  $G_h = \sum_{f \in F} N_f \cdot (O_f - 1)/N \cdot 100$ . Fig. 9a shows the results of this experiment. As expected, the horizontal gain increases with the number of process models reaching a final value of 35.6% for the SAP dataset and 21% for the IBM dataset. This trend is determined by the increasing number of shared fragments as the total size of the repository increases. For example, for the SAP dataset there are 98 shared fragments

<sup>2</sup> <http://www.zurich.ibm.com/csc/bit/downloads.html>



**Fig. 8.** Average storage gain when sharing fragments across versions of the same process model

when the repository is populated with 100 process models and this number increases to 840 fragments with the full dataset. This gives an indication of the reduction in maintenance effort, as any update to any of those fragments or their child fragments, will be automatically reflected onto all process models containing those fragments.

Following from the results of the previous experiment, we tested the effects of change propagation onto the repository. We populated the repository with the SAP dataset and performed 100 updates on randomly selected fragments. An update to a fragment consists of a combination of the following operations with associated probabilities: label change (33%), serial node insertion (17%), parallel node insertion (33%) and node deletion (33%). The total number of operations performed in an update is proportional to the number of nodes in the fragment being updated. In these tests we set the operations-to-nodes ratio to one. For example, when updating a fragment with 10 nodes, 10 operations were performed consisting of approximately 3 label changes, 3 node deletions, 2 serial node insertions and 2 parallel node deletions.

The change propagation policy of all process models was set to instant propagation during these tests as we wanted all changes to be immediately propagated to all affected models. After each update, we measured the total number of automatically propagated changes in the repository. We repeated the same experiment for the IBM dataset. The average results for 10 test runs with both datasets are shown in Fig. 9b. Accordingly, the number of propagated changes increases with the number of updates performed on a process model collection. For example, on average 20 automatic changes were applied by the repository across different process models when 100 updates were performed on the SAP dataset. If our change propagation method is not used, process modelers have to analyze the entire process model collection and apply all these changes to relevant process models manually, which could be a time consuming and error-prone activity. Thus, automatic change propagation provides indeed a significant benefit in maintaining the consistency of the repository.

Finally, we measured the effectiveness of our fragment-based locking by comparing it with the model-based locking available in current process model repositories. In this experiment, we used software agents to randomly lock fragments of a given process model collection in order to simulate random updates. We first generated a sequence of locking actions for each agent and saved it in a file. An action is a tuple (*process model identifier, fragment identifier, locking duration*). For example action (12, 25, 560) forces an agent to lock fragment 25 of process model 12 for 560 milliseconds. For each



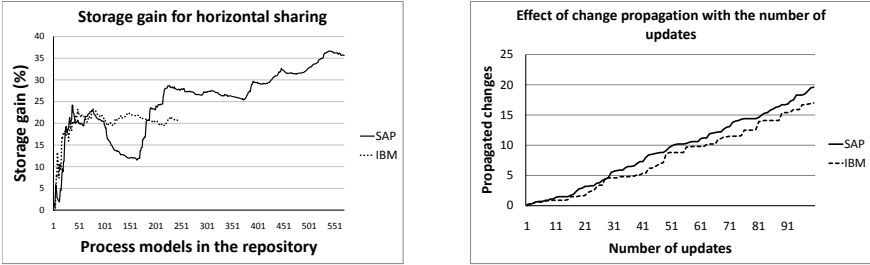


Fig. 9. Vertical storage gain (a) and change propagation (b) with the growth of the repository

action, the process model was selected using a uniform probabilistic distribution over all process models in a given collection. The fragment was selected based on a Gaussian distribution over the sizes of the fragments of the selected process model, where the mean size of the distribution was set to 10% of the size of the selected process model. The locking duration was determined based on an inverse exponential distribution with mean of 5 seconds, in order to speed up the tests.

Once all action files were generated, we executed two tests for each file: i) each agent attempted to lock only the specified fragment; ii) each agent attempted to lock the entire process model for each action, to simulate the traditional model-based locking. We executed these tests for two process model collections, with 10 and 30 process models, chosen with uniform size distribution from the SAP dataset. We used these small numbers of process models as in an average BPM project multiple users typically work collaboratively on a small set of process models. For each collection, we performed three tests by varying the number of concurrent agents from 10, to 20 and 30, and we computed the success rate for each test as the ratio of the number of successful operations over the number of total operations. The results are shown in Fig. 10.

As expected, the fragment-based locking mechanism scored the highest success rate in all tests. We also observed that the gain of this locking compared to that of model-based locking increases with the increase of concurrent agents (for example, when using 10 agents on 30 process models, fragment level locking facilitated 15% more operations than process level locking, while fragment level locking facilitated 110% more operations for 30 agents). Further, this gain is higher when agents are competing for a

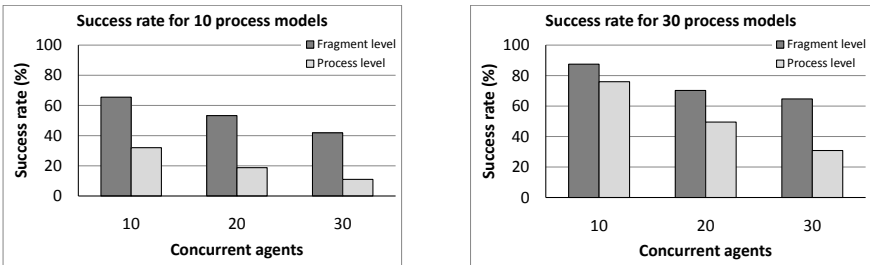


Fig. 10. Success rate of locking operations in 10 process models (a) and 30 process models (b)

smaller number of process models. Thus, we can conclude that our fragment-based locking mechanism is more effective than the traditional model-based locking.

## 7 Related Work

In this section we discuss related work in the field of BPM as well as in other fields, such as software engineering and computer aided design. Our discussion is categorized under version control, repositories, process model changes and concurrency control.

### 7.1 Version Control

Version control has been extensively studied in at least three different fields: Temporal Databases (TDBs), Software Engineering (SE) and Computer Aided Design (CAD). TDBs [36,14] deal with issues that arise when data evolution and histories of temporal models have to be managed. In SE, Source Code Control System (SCCS) [34] was probably one of the precursors of version control systems. Here a *revision* of a file is created each time the file is modified. Revision Control Systems (RCS) [39] extended SCCS by introducing the concept of *variant* to capture branching evolution (e.g. in SCCS, evolutions are represented as a sequence, while in RCS they are represented as a tree). Space consumption is optimized by only storing textual differences (*deltas*) between subsequent versions. This is the same approach used by popular version control systems such as CVS and SVN. It is possible to use textual deltas to version control process models by considering XML based serializations of process models (e.g. EPML, XPDL, YAWL). However, such deltas only serve as a method to reconstruct different versions and do not facilitate other essential aspects of process model repositories as mentioned later in this section.

Within SE, approaches in the area of Software Configuration Management [9], propose to use database technology to enhance the underlying data model and make the notion of version explicit. *Damokles* [13] is probably one of the first database-based versioning environment for SE. It offers the notion of *revision* as a built-in datatype and a version-aware data modeling language. In [30] the authors present an object graph versioning system (HistOOry) which allows applications to store and efficiently browse previous states of objects. This approach keeps history of object graphs, while ours deals with version control of graphs. Moreover, our goals are different: we focus on graph fragment reusability and update propagation.

A version control method specifically designed for process models is proposed in [2]. This method is based on change operations: the differences between two process model versions are specified as a set of insert, delete and modify operations on tasks, links and attributes. The version history of a process model is stored as the initial version plus the set of change operations required to derive all subsequent versions. When a new process model version is checked in, the change operations required to derive this version from the last version of the same process model are computed and stored as the delta of the new version. Similarly, when a process model version is checked out, all change operations required to derive the requested version from the initial version are retrieved and applied to the initial version to construct the requested version. Another method for process model version control is to store all versions of a process model in a single graph by annotating the graph's nodes and edges with version numbers [45]. Once such a graph

is built, one can derive any version of its process model by following a set of derivation rules. Thus, deltas between process model versions are captured as a set of graph elements (i.e. nodes and edges). However, the types of deltas proposed in the above two methods, as well as the textual deltas used in SCCS, RCS, CVS and SVN discussed earlier, do not have any other purpose than reconstructing different versions. In contrast, we use process fragments as deltas, which are meaningful components of process models. In addition to reconstructing different versions, we use fragments to automatically propagate changes across process model versions and across different process models, and to reduce conflicting edit operations over these models. Further, fragments can be used as queries for searching specific process models in large repositories, as done in [40], or as compositional units to create new process models. For example, a fragment used in an old process model version can be reused in a new version of another process model. Hence, we argue that our fragment-based approach is better-suited for the management of process models, specially when other requirements such as change propagation, concurrency control and search are considered, in addition to pure version control.

Thomas [38] presents an architecture for managing different versions of reference process models. However this approach focuses on high-level aspects of versioning such as integration with different enterprise databases, inter-connections with external applications, attributes to be associated with versions and user interface design. Thus, this work is complementary to our research as our methods can be embedded in such an architecture.

## 7.2 Repositories

Repositories provide a shared database for artifacts produced or used by an enterprise, and also facilitate functions such as version control, check-in, check-out and configuration management [6]. The use of repositories for managing artifacts in different domains has been studied and different storage mechanisms have been proposed. The concept of managing complex artifacts as aggregations of lower level components has been discussed in the literature (e.g. [9][17][19][18]). In particular, version control and change propagation of such composite artifacts have been studied in the context of CAD repositories [17][19][18]. Accordingly, the highest degree of sharing is obtained when all software components are versioned including composite and atomic components, and their relationships. The storage technique that we propose extends such concepts in the context of process model management. Most of the research on composite artifact storage mechanisms assumes that lower level objects and their composition relationships are explicitly stated by users. In our technique, we use the RPST algorithm to automatically decompose process models into lower level fragments in linear time. Further, when storing process models we always decompose them into the smallest possible RPST fragments, thus increasing the advantages of space utilization, change propagation and concurrency control. We also share the structures and composition relations between such process models. This allows us to maximize the sharing of fragments among process models (i.e. identical structures are shared even if child mappings are not the same). Further, we share components (i.e. fragments) and structures across multiple versions (i.e. vertically) as well as across different process models (i.e. horizontally).

Business process model repositories stemming from research initiatives support process model-specific features in addition to basic insert, retrieve, update and delete functions [27][37][28][8][44], such as searching stored process models based on different

parameters. For example, the semantic business process repository [28] focuses on querying business processes based on ontologies while the process repository proposed in [8] also focuses on the lifecycle management of process models. Similar features can be found in commercial process model repositories, such as the ARIS platform [10]. However, both academic and commercial process model repositories only support basic version control at the level of process nodes. Moreover, none of these solutions adequately addresses the problems of change management and concurrency control. For example, in ARIS one can only propagate updates to node attributes.

A repository designed for storing process model fragments is proposed in [35]. The purpose is to develop new process models by reusing existing fragments. As such, this repository only stores individual fragments but not entire process models, nor the relations among their fragments. On the other hand, we focus on storage and management of process models, so we use fragments as a means to decompose process models hierarchically, and to identify commonalities among process models and their versions.

Redundant process fragments are identified as an issue when managing large process model repositories in [42]. If these fragments are not kept in sync, changes to the repository may lead to inconsistencies. Since we share redundant fragments only once, and we propagate changes across them, our technique can be seen as a way of solving the “redundant process fragments” issue described in [42].

### 7.3 Process Model Changes

Different classifications of process model changes have been proposed in the literature [43][112]. Weber et al. [43] propose a set of change patterns that can be applied to process models and process instances, in order to align these artifacts with changing requirements. These change patterns focus on fragment-level operations (e.g. inserting a new fragment into a process model, deleting a fragment or moving a fragment to a different position) as well as on control-flow changes (e.g. adding a new control-flow dependency and changing the condition of a conditional branch). The classification proposed by Dijkman [112] focuses on finer-grained changes including the insertion and removal of an activity, the refinement of an activity into a collection of activities and the modification of an activity’s input requirements. This classification also includes changes performed on resource-related aspects, such as allocating an activity to a different human role. These classifications are useful for many areas, such as developing and evaluating process model editors, identifying differences between process models, designing change operation based concurrency control techniques and developing version control systems. However, our storage and version control technique considers the final states of process models, and the operations applied to derive different process models are not required for our approach. As such, this work is complementary to ours. In fact, we do not impose any restriction on the type of changes that can be performed on our process models.

### 7.4 Concurrency Control

Fine-grained locking of generic objects and CAD objects has been studied in [29][34]. However, the possibility of fine-grained locking of process models at the process fragment level has not been studied in the literature. The issue of resolving conflicts in different process model versions has been explored both at design-time and at run-time. At run-time, the propagation of process model changes to running process instances

without causing errors and inconsistencies has been extensively studied in the literature [32][33][16][21]. Since our process models are design-time artifacts, this work is complementary to ours. At design-time, Küster et al. [23][22][1] propose a method for merging two versions of the same process model based on the application of *change operations* which can be automatically identified without the need for a change log. Similar to our approach, this solution relies on the decomposition of process models into SESE fragments. However, this approach focuses on resolving conflicts once overlapping modifications are detected, while our approach prevents conflicts before they occur through selective locking. Thus, it may be possible to combine both approaches in order to develop flexible collaborative environments.

## 8 Conclusion

This paper presents a novel versioning model and associated storage structure specifically designed to deal with (large) process model repositories. The focal idea is to store and version single SESE process fragments, rather than entire process models. The motivation comes from the observation that process model collections used in practice feature a great deal of redundancy in terms of shared process fragments.

The contribution of this technique is threefold. First, repository users can effectively keep track of the relations among different process models (horizontal sharing) and process model versions (vertical sharing). Second, sophisticated change propagation is achieved, since changes in a single fragment can be propagated to all process models and process model versions that share that fragment. This goes well beyond the change propagation provided by current process model repositories. This in turn allows users to automatically ensure consistency, and maximize standardization, in large process model repositories. Finally, locking can also be defined at the granularity of single fragments, thus fostering concurrent updates by multiple users, since it is no longer required to lock entire process models. To the best of our knowledge, fragment-based concepts have not been adopted to study these aspects of process model collections to date.

An important application of our technique is the management of variability in process model repositories. In fact, variants of a same process model, e.g. the “Home” and “Motor” variants of an “Insurance claim” process model, are never that dissimilar from each other, i.e. they typically share various fragments [24]. These variants can either be explicitly modeled as different branches of the same process, or their commonalities can be automatically detected when these variants are inserted into the repository. In both cases, our technique will trace these links among the variants, and keep the variants synchronized whenever they undertake changes.

This technique was implemented and its usefulness was evaluated on two industrial process model collections. In future work, we plan to combine our fragment-based locking method with operational merging [26] to provide more flexible conflict resolution in concurrent fragment updates. Currently, our technique is limited to the control flow of process models. In future, we plan to also version process’ data and resources. Moreover, we plan to further evaluate our technique from a qualitative point of view, by conducting usability tests with process model repository users.

**Acknowledgments.** This research is funded by the Smart Services Cooperative Research Centre (CRC) through the Australian Government’s CRC Programme.

## References

1. Gerth, C., Küster, J.M., Luckey, M., Engels, G.: Precise Detection of Conflicting Change Operations Using Process Model Terms. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6395, pp. 93–107. Springer, Heidelberg (2010)
2. Bae, H., Cho, E., Bae, J.: A Version Management of Business Process Models In Bpms. In: Chang, K.C.-C., Wang, W., Chen, L., Ellis, C.A., Hsu, C.-H., Tsoi, A.C., Wang, H. (eds.) APWeb/WAIM 2007. LNCS, vol. 4537, pp. 534–539. Springer, Heidelberg (2007)
3. Bancilhon, F., Kim, W., Korth, H.F.: A model of cad transactions. In: Pirotte, A., Vassiliou, Y. (eds.) VLDB, pp. 25–33. Morgan Kaufmann (1985)
4. Barghouti, N.S., Kaiser, G.E.: Concurrency control in advanced database applications. *ACM Comput. Surv.* 23(3), 269–317 (1991)
5. Berliner, B., Prisma, I.: CVS II: Parallelizing software development. In: Proceedings of the USENIX Winter 1990 Technical Conference, vol. 341, p. 352 (1990)
6. Bernstein, P.A., Dayal, U.: An overview of repository technology. In: VLDB, pp. 705–713. Morgan Kaufmann (1994)
7. Cardoso, J.: Poseidon: a Framework to Assist Web Process Design Based on Business Cases. *Int. J. Cooperative Inf. Syst.* 15(1), 23–56 (2006)
8. Choi, I., Kim, K., Jang, M.: An xml-based process repository and process query language for integrated process management. *KPM* 14, 303–316 (2007)
9. Conradi, R., Westfechtel, B.: Version models for software configuration management. *ACM Computing Surveys* 30(2), 232–282 (1998)
10. Davis, R., Brabänder, E.: ARIS design platform: getting started with BPM. Springer-Verlag New York Inc. (2007)
11. Dijkman, R.M.: A classification of differences between similar Business Processes. In: EDOC, p. 37. IEEE Computer Society (2007)
12. Dijkman, R.M.: Diagnosing Differences Between Business Process Models. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 261–277. Springer, Heidelberg (2008)
13. Dittrich, K.-R.: The damokles database system for design applications: its past, its present, and its future, pp. 151–171. Ellis Horwood Books (1989)
14. Dumas, M., Fauvet, M.-C., Scholl, P.-C.: TEMPOS: a platform for developing temporal applications on top of object DBMS. *IEEE TKDE* 16(3) (2004)
15. Ekanayake, C.C., La Rosa, M., ter Hofstede, A.H.M., Fauvet, M.-C.: Fragment-based version management for repositories of business process models. QUT ePrints 39059. Queensland University of Technology, Australia (2010)
16. Joeris, G., Herzoz, O.: Managing evolving workflow specifications. In: Proc. of IFCIS, pp. 310–321. IEEE (1998)
17. Katz, R.H.: Towards a unified framework for version modeling in engineering databases. *ACM Comput. Surv.* 22(4), 375–408 (1990)
18. Katz, R.H., Chang, E.E.: Managing change in a computer-aided design database. In: VLDB, pp. 455–462 (1987)
19. Katz, R.H., Chang, E.E., Bhateja, R.: Version modeling concepts for computer-aided design databases. In: SIGMOD Conference, pp. 379–386. ACM (1986)
20. Keller, G., Teufel, T.: SAP R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley (1998)
21. Kim, D., Kim, M., Kim, H.: Dynamic business process management based on process change patterns. In: Proc. of ICCIT, pp. 1154–1161. IEEE (2007)
22. Küster, J.M., Gerth, C., Engels, G.: Dependent And Conflicting Change Operations of Process Models. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 158–173. Springer, Heidelberg (2009)

23. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and Resolving Process Model Differences in the Absence of a Change Log. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) *BPM 2008*. LNCS, vol. 5240, pp. 244–260. Springer, Heidelberg (2008)
24. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.M.: Merging Business Process Models. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) *OTM 2010*. LNCS, vol. 6426, pp. 96–113. Springer, Heidelberg (2010)
25. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., Garcia-Banuelos, L.: Apromore: An advanced process model repository. In: *ESWA (2011)*
26. Lippe, E., van Oosterom, N.: Operation-based merging. *SIGSOFT Software Engineering Notes* 17(5), 78–87 (1992)
27. Liu, C., Lin, X., Zhou, X., Orłowska, M.E.: Building a repository for workflow systems. In: *TOOLS (31)*, pp. 348–357. IEEE Computer Society (1999)
28. Ma, Z., Wetzstein, B., Anicic, D., Heymans, S., Leymann, F.: Semantic business process repository. In: *SBPM. CEUR*, vol. 251 (2007)
29. Munson, J.P., Dewan, P.: A concurrency control framework for collaborative systems. In: *CSCW*, pp. 278–287 (1996)
30. Pluquet, F., Langerman, S., Wuyts, R.: Executing code in the past: efficient object graph versioning. In: *OOPSLA 2009*, Orlando, Florida, USA (2009)
31. 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)
32. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases* 16(1), 91–116 (2004)
33. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications. In: Meersman, R., Tari, Z. (eds.) *OTM 2004*. LNCS, vol. 3290, pp. 101–120. Springer, Heidelberg (2004)
34. Rochkind, M.-J.: The source code control system. *IEEE TSE* 1(4), 364–370 (1975)
35. Schumm, D., Karastoyanova, D., Kopp, O., Leymann, F., Sonntag, M., Strauch, S.: Process fragment libraries for easier and faster development of process-based applications. *Journal of Systems Integration* 2(1), 39–55 (2011)
36. Snodgrass, R.T.: Temporal databases. In: *Proc. of GIS (1992)*
37. Song, M., Miller, J.A., Arpinar, I.B.: Repox: An xml repository for workflow designs and specifications. Technical report, University of Georgia, USA (2001)
38. Thomas, O.: Design and implementation of a version management system for reference modeling. *JSW* 3(1), 49–62 (2008)
39. Tichy, W.-F.: Design implementation and evaluation of a revision control system. In: *Proc. of the 6th Int. Conf. on Software Engineering*, Tokyo, Japan (1982)
40. Uba, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Clone Detection in Repositories of Business Process Models. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) *BPM 2011*. LNCS, vol. 6896, pp. 248–264. Springer, Heidelberg (2011)
41. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. *Data Knowl. Eng.* 68(9), 793–818 (2009)
42. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. *Computers in Industry* 62(5), 467–486 (2011)
43. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. *DKE* 66(3), 438–466 (2008)
44. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Business process model repositories - framework and survey. Technical Report 232409, TU/e, The Netherlands (2009)
45. Zhao, X., Liu, C.: Version Management in the Business Process Change Context. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 198–213. Springer, Heidelberg (2007)

# Selecting and Ranking Business Processes with Preferences: An Approach Based on Fuzzy Sets

Katia Abbaci<sup>1</sup>, Fernando Lemos<sup>2</sup>, Allel Hadjali<sup>1</sup>, Daniela Grigori<sup>2</sup>,  
Ludovic Liétard<sup>3</sup>, Daniel Rocacher<sup>1</sup>, and Mokrane Bouzeghoub<sup>2</sup>

<sup>1</sup> IRISA/ENSSAT, Rue de Kérampont BP 80518 Lannion, France  
{katia.abbaci,allel.hadjali,daniel.rocacher}@enssat.fr

<sup>2</sup> PRISM Lab, 45 Av. des États Unis 78000 Versailles, France  
{fernando.lemos,daniela.grigori,mokrane.bouzeghoub}@prism.uvsq.fr

<sup>3</sup> IRISA/IUT, Rue Edouard Branly BP 30219 Lannion, France  
ludovic.lietard@univ-rennes1.fr

**Abstract.** Current approaches for service discovery are based on semantic knowledge, such as ontologies and service behavior (described as process model). However, these approaches still remain with a high selectivity rate, resulting in a large number of services offering similar functionalities and behavior. One way to improve the selectivity rate and to provide the best suited services is to cope with user preferences defined on quality attributes. In this paper, we propose and evaluate a novel approach for service retrieval that takes into account the service process model and relies both on preference satisfiability and structural similarity. User query and target process models are represented as annotated graphs, where user preferences on QoS attributes are modelled by means of fuzzy sets. A flexible evaluation strategy based on fuzzy linguistic quantifiers (such as *almost all*) is introduced. Then, two families of ranking methods are discussed. Finally, an extensive set of experiments based on real data sets is conducted, on one hand, to demonstrate the efficiency and the scalability of our approach, and on the other hand, to analyze the effectiveness and the accuracy of the proposed ranking methods compared to expert evaluation.

**Keywords:** web service retrieval, quality of services, preferences, fuzzy set theory, linguistic quantifier.

## 1 Introduction

Searching for a specific service within service repositories become a critical issue for the success of service oriented and model-driven architectures and for service computing in general. This issue has recently received considerable attention and many approaches have been proposed. Most of them are based on the matchmaking of process input/outputs [1], service behavior (described as process model) [2,3,4] or ontological knowledge [4]. However, these approaches have high selectivity rate, resulting in a large number of services offering similar functionalities and behavior [4].



One way to discriminate between similar services is to consider non-functional requirements such as quality preferences (response time, availability, etc.). Indeed, for a given query in a given context, there is no need to provide all possible services but only those satisfying user preferences and contextual constraints. A recent trend towards quality-aware approaches has been initiated [5,6,7], but it is limited to atomic services. Our goal is to go further these approaches into a unique integrated approach dealing with functional and non-functional requirements in service retrieval. Targeting this goal poses the following challenges: (i) At the description level, provide a model allowing to specify non-functional requirements at different granularity levels of the service functional description; (ii) At the discovery level, define an evaluation method that efficiently computes the satisfiability of a target service w.r.t. the functional and non-functional requirements of a user query.

More specific challenges related to non-functional characteristics should also be taken into account: (i) Services are deployed over dynamic and heterogeneous environments such that their non-functional properties are often given or derived with different accuracies; (ii) Users are not always able to precisely specify their non-functional constraints; (iii) Users have different points of view over what is a satisfactory service according to the same set of non-functional constraints; (iv) The service retrieval should avoid empty or overloaded answers due to the imprecision of the user's query.

Preferences are a natural way to facilitate the definition of non-functional constraints in user query. They are flexible enough, on one hand, to avoid empty returns caused by very strict user constraints and, on the other hand, to provide an adequate set of relevant results even when user specifies too general constraints. In addition, fuzzy logic has been used as a key technique to take into account human point of view in preference modelling and evaluations [8].

In [9], it is proposed a QoS-aware process discovery method whereas the user query is a graph annotated with QoS factors. Starting from [9], this paper investigates a novel approach for service selection and ranking taking into account both behavior specification and QoS preferences. User query and target process models are represented as graphs, where queries are annotated with preferences on QoS properties and targets are annotated with QoS attributes. Preferences are represented by means of fuzzy sets as they are more suitable to the interpretation of linguistic terms (such as *high* or *fast*) that constitute a convenient way for users to express their preferences. To avoid empty answers for a query, an appropriate flexible evaluation strategy based on fuzzy linguistic quantifiers (such as *almost all*) is introduced.

In the remainder of this paper, Section 2 provides some basic background and discusses related works. Section 3 describes process model specification with preferences. Section 4 addresses fuzzy preference modelling and evaluation. Section 5 presents our interpretation of process models similarity based on linguistic quantifiers. Section 6 discusses service ranking methods. Section 7 proposes an illustrative example and Section 8 presents a set of experiments conducted to evaluate our approach. Finally, Section 9 concludes the paper.

## 2 Background and Related Work

Here, we recall some notions on preference modelling (e.g., Pareto and fuzzy set based models) and we review preference-based service discovery approaches.

### 2.1 Preference Modelling

The semantics of preferences assumed in this work is the one provided by the databases area: preferences are used to help in reducing the amount of information returned in response to user queries and to avoid the happening of empty answers. Generally, two families of approaches can be distinguished to model preferences. The first one relies on commensurability assumption which leads to a total pre-order [10,11,8]. We highlight the *SQLf* proposal [11], which is based on the extension of the relational algebra to fuzzy set theory. The second family assumes that commensurability does not hold, in this case no compensation is allowed between criteria and only a partial order is obtained [12,13,14].

One popular approach of this last family is *Preference SQL* [13]. It provides foundations for a *Pareto*-based preference model for database systems. A preference is formulated as a strict partial order on a set of attribute values. It introduces a number of preference operators to express and compose preferences. Let us note that all tuples returned by a Preference SQL query satisfy the *Pareto* principle. A compensatory strategy between different atomic conditions is not possible due to the fact that Preference SQL makes use of different functions for evaluating the distance with which a tuple disagrees with an atomic condition. Moreover, the most preferred tuples are returned to the user without being capable to distinguish how better is one tuple compared to another.

Fuzzy sets were introduced in [15] for dealing with the representation of classes or sets whose boundaries are not well defined. Then, there is a gradual transition between the full membership and the full mismatch (an order relation on membership levels can be established). Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as *cheap*, *fast*, etc. Formally, a fuzzy set  $F$  on the universe  $X$  is described by a membership function  $\mu_F : X \rightarrow [0, 1]$ , where  $\mu_F(x)$  represents the **membership degree** of  $x$  in  $F$ . By definition, if  $\mu_F(x) = 0$  then the element  $x$  **does not belong at all** to the fuzzy set  $F$ , if  $\mu_F(x) = 1$  then  $x$  **fully belongs** to  $F$ . When  $0 < \mu_F(x) < 1$ , one speaks of **partial membership**. The set  $\{x \in F | \mu_F(x) > 0\}$  represents the **support** of  $F$  and the set  $\{x \in F | \mu_F(x) = 1\}$  represents its **core**.

In addition, the closer  $\mu_F(x)$  to the value 1, the more belonging to  $F$ . Therefore, given  $x, y \in F$ , one says that  $x$  is preferred to  $y$  iff  $\mu_F(x) > \mu_F(y)$ . If  $\mu_F(x) = \mu_F(y)$ , then  $x$  and  $y$  are equally preferred. In practice, the membership function associated to  $F$  is often represented by a *trapezoid*  $(\alpha, \beta, \varphi, \psi)$ <sup>1</sup>, where  $[\alpha, \psi]$  is its support and  $[\beta, \varphi]$  is its core. Among other forms (Gaussian, sigmoidal, bell, etc), this one is very easy to be defined and to manipulate.

<sup>1</sup> In our case, the quadruplet  $(\alpha, \beta, \varphi, \psi)$  is user-defined to ensure the subjectivity property.

A fuzzy set-based approach to preference queries proposed in [8] is founded on the use of fuzzy set membership functions that describe the preference profiles of the user on each attribute domain involved in the query. This is especially convenient and suitable when dealing with numerical domains, where a continuum of values is to be interfaced for each domain with satisfiability degrees in the unit interval scale. Then satisfiability degrees associated with elementary conditions are combined using fuzzy set connectives, which may go beyond conjunctive and disjunctive aggregations (by possibly involving fuzzy quantifiers, if the satisfiability of most of the elementary conditions in a query is required).

## 2.2 Preference-Based Service Discovery

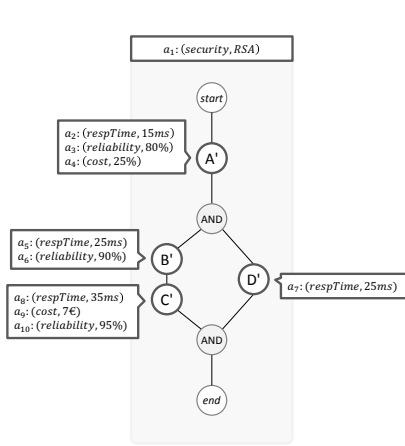
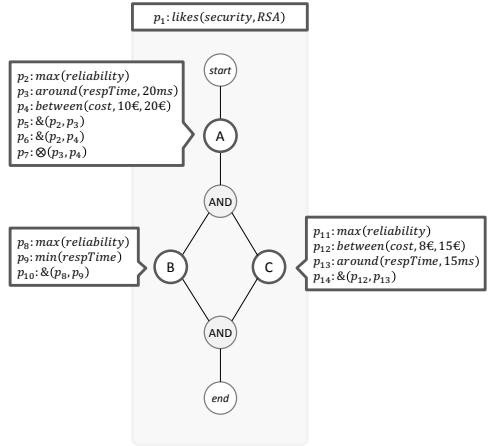
*Crisp Logic-based Approaches.* Most of the first approaches for service discovery using preferences were based on crisp logic solution and considered the services as black boxes [16,6,17]. With regard to the specification model, some of them do not deal with preferences; instead, they compute for each service a score based on set of the non-functional properties of the service [16]. The other approaches does not propose or use preference constructors to help user better define his preferences or interpret the results [6,17]. The models presented are not abstract enough to provide a widely use of the approach in different contexts; some of them imposes a restricted set of properties over which user can work.

*Fuzzy Logic-based Approaches.* In last decades, several service discovery approaches based on fuzzy set theory have been proposed [18,19]. In [19] the authors treat the web service selection for composition as a fuzzy constraint satisfiability problem. They assign to each QoS criterion five fuzzy sets describing its constraint levels. In [20,21], QoS based service selection is modelled as a fuzzy multiple criteria decision making problem. In [22], a service selection mechanism is presented allowing the service broker to select a set of services from a query specifying imprecise constraints defined by fuzzy sets. The query evaluation is based on the aggregation of the obtained degrees over constraints. Şora et al. [5] propose an approach to automatically generate fuzzy rules from user preferences and rank the candidate services using a fuzzy inference process. The global score of each web service is given in a scale of satisfiability levels instead of an aggregation of the satisfiability degrees of the preferences.

The aforementioned fuzzy approaches take into account only the satisfiability of preferences whereas they ignore the structural similarity of web services. Most of them do not verify the *subjectivity property*, which considers the user point of view when defining the membership functions. Moreover, these works deal only with services as black boxes. In this paper, user can also define preferences over the activities of the service behavior specification and both structural similarity and user preference satisfiability are considered.

## 3 Preferences in Process Model Specification

Many languages are currently available to describe service process models, e.g., WS-BPEL and OWL-S. They represent a process model as a set of atomic

Fig. 1. Target Graph  $t_1$ Fig. 2. Query Graph  $q_1$ 

activities combined using control flow structures. As a consequence, these languages can be abstracted as a direct graph  $G = (V, E)$ , where the vertices represent activities (e.g., *hotel reservation*, *payment*) or control flow nodes (e.g., *and*, *or*), while the edges represent the flow of execution between activities.

In this work, services are specified as graphs annotated with QoS properties and user queries are specified as graphs annotated with preferences. Figure 1 presents a global annotation indicating the security of the process model and activity annotations indicating other QoS attributes of some activities. Figure 2 shows a sample user query annotated with a global preference indicating user prefers services providing RSA encryption and some activity preferences involving reliability, response time and cost. It is worth mentioning that our model can be implemented by extension mechanisms in OWL-S.

We precise that, in this work, target models are considered already annotated with QoS attributes while the user is the one to define the preference annotations of his query. Techniques to obtain the QoS information of a process model can be found in [23]. Next, we present the formal definitions of our model:

**Definition 1.** An **annotation** is a pair  $(m, r)$ , where  $m$  is a QoS attribute and  $r$  is a value for  $m$ <sup>2</sup>. It can be specified over a process model graph (**global annotation**) or over an atomic activity (**activity annotation**).

**Definition 2.** A **preference** is an expression that represents a desire of the user over the QoS attributes of a process model or activity. It can be specified over a process model graph (**global preference**) or over an atomic activity (**activity preference**). It can be of one the following forms<sup>3</sup>:

<sup>2</sup> We abstract from the different units in which a value can be described.

<sup>3</sup> Based on a subset of preferences defined in [13].

- *around* ( $m, r_{desired}, \mu_{around}$ ): it favors the value  $r_{desired}$  for attribute  $m$ ; otherwise, it favors those close to  $r_{desired}$ . The membership function  $\mu_{around}$  evaluates the degree to which a value  $r$  satisfies  $r_{desired}$ ;
- *between* ( $m, r_{low}, r_{up}, \mu_{between}$ ): it favors the values inside the interval  $[r_{low}, r_{up}]$ ; otherwise, it favors the values close to the limits. The function  $\mu_{between}$  evaluates the degree to which a value  $r$  satisfies the interval  $[r_{low}, r_{up}]$ ;
- *max* ( $m, \mu_{max}$ ): it favors the highest value; otherwise, the closest value to the maximum is favored. For example, the maximum of availability is equal by default to 100%. The function  $\mu_{max}$  evaluates the degree to which a value  $r$  satisfies the highest value of  $m$ ;
- *min* ( $m, \mu_{min}$ ): it favors the lowest value; otherwise, the closest value to the minimum is favored, as example: the minimum of response time or cost is equal by default to 0.  $\mu_{min}$  evaluates to which degree a value  $r$  satisfies the lowest value of  $m$ ;
- *likes* ( $m, r_{desired}$ ): it favors the value  $r_{desired}$ ; otherwise, any other value is accepted;
- *dislikes* ( $m, r_{undesired}$ ): it favors the values that are not equal to  $r_{undesired}$ ; otherwise,  $r_{undesired}$  is accepted;
- Pareto  $\otimes$  ( $p_i, p_j$ ): it states that the two soft preference expressions  $p_i$  and  $p_j$  are equally important;
- Prioritized  $\&$  ( $p_i, p_j$ ): it states that the soft preference expression  $p_i$  is more important than the soft preference expression  $p_j$ .

The work in [13] distinguishes two types of preferences: *atomic* (*around*, *between*, *max*, *min*, *likes* and *dislikes*) and *complex* ( $\otimes$  and  $\&$ ). It also distinguishes two types of atomic preferences: *numerical* (*around*, *between*, *max* and *min*) and *non-numerical* (*likes* and *dislikes*). The values in non-numerical preferences are taken from a global ontology of a type “is-a”  $O$ , given by the user.

## 4 A Fuzzy Model to Evaluate Preferences

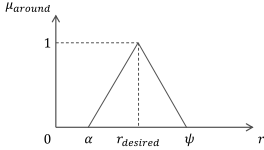
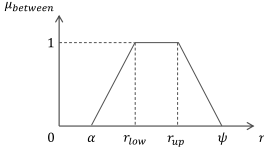
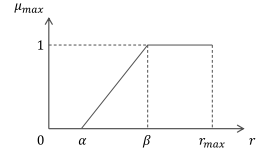
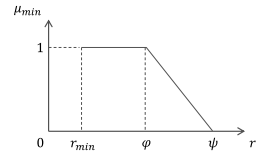
Here, we introduce a fuzzy semantics of the atomic preferences discussed in the Section 3, and show how they can be evaluated. In particular, we propose a metric, called *satisfiability degree* ( $\delta$ ), that measures how well a set of annotations of a target process model satisfies a set of preferences present in the query. The computation of this degree is done both for atomic and complex preferences.

### 4.1 Atomic Preferences

For numerical atomic preferences, the satisfiability degree is obtained thanks to user-specific membership functions. Table 1 summarizes the fuzzy modelling of numerical preferences of interest. Given a preference  $p$  and an annotation  $a : (m, r)$ , one is interested in computing the degree to which the annotation  $a$  satisfies the fuzzy characterization underlying  $p$ .

For example, consider the constructor *between*: a fuzzy preference  $p : \textit{between}(m, r_{low}, r_{up})$  is characterized by the membership function  $(\alpha, \beta, \varphi, \psi)$ , where

**Table 1.** Fuzzy modelling of numerical preferences

NUMERICAL PREFERENCE	FUZZY INTERPRETATION
$around(m, r_{desired}, \mu_{around})$	$\mu_{around}(r) = \begin{cases} 0, & r \leq \alpha, r \geq \psi \\ \frac{r-\alpha}{r_{desired}-\alpha}, & \alpha < r < r_{desired} \\ 1, & r = r_{desired} \\ \frac{\psi-r}{\psi-r_{desired}}, & r_{desired} < r < \psi \end{cases}$ 
$between(m, r_{low}, r_{up}, \mu_{between})$	$\mu_{between}(r) = \begin{cases} 0, & r \leq \alpha, r \geq \psi \\ \frac{r-\alpha}{r_{low}-\alpha}, & \alpha < r < r_{low} \\ 1, & r_{low} \leq r \leq r_{up} \\ \frac{\psi-r}{\psi-r_{up}}, & r_{up} < r < \psi \end{cases}$ 
$max(m, \mu_{max})$	$\mu_{max}(r) = \begin{cases} 0, & r \leq \alpha \\ \frac{r-\alpha}{\beta-\alpha}, & \alpha < r < \beta \\ 1, & \beta \leq r \leq r_{max} \end{cases}$ 
$min(m, \mu_{min})$	$\mu_{min}(r) = \begin{cases} 1, & r_{min} \leq r \leq \varphi \\ \frac{\psi-r}{\psi-\varphi}, & \alpha < r < \psi \\ 0, & r \geq \psi \end{cases}$ 

$\beta = r_{low}$ ;  $\varphi = r_{up}$ ;  $\alpha$  and  $\psi$  are two values from the universe  $X$ . Let  $a : (m, r)$  be an annotation of a target graph, the satisfiability degree of preference  $p$  according to  $a$  is given by: (i)  $p$  is completely satisfied iff  $r \in [r_{low}, r_{up}]$ :  $\mu_{between}(p, a) = 1$ , i.e.  $\delta(p, a) = 1$ ; (ii) the more  $r$  is lower (resp. higher) than  $r_{low}$  (resp.  $r_{up}$ ), the less  $p$  is satisfied:  $0 < \mu_{between}(p, a) = \delta(p, a) < 1$ ; (iii) for  $r \in ]-\infty, \alpha] \cup [\psi, +\infty[$ ,  $p$  is not satisfied:  $\mu_{between}(p, a) = \delta(p, a) = 0$ .

For non-numerical atomic preferences, the satisfiability degree is based on the semantic similarity between concepts. We applied the widely known semantic similarity proposed in [24], which states that given an ontology  $O$  and two concepts  $c_1$  and  $c_2$ , the semantic similarity  $wp$  between  $c_1$  and  $c_2$  is given by  $wp(O, c_1, c_2) = 2N_3 / (N_1 + N_2 + 2N_3)$ , where  $c_3$  is the least common super-concept of  $c_1$  and  $c_2$ ,  $N_1$  is the length of the path from  $c_1$  to  $c_3$ ,  $N_2$  is the length of the path from  $c_2$  to  $c_3$ , and  $N_3$  is the length of the path from  $c_3$  to the root of the ontology. Given a non-numerical atomic preference  $p$  and an annotation  $a$ , the satisfiability degree  $\delta(p, a)$  is given by:

- If  $p = likes(m, r_{desired})$ , then  $\delta(p, a) = \begin{cases} 1, & r_{desired} = r \\ wp(O, r_{desired}, r), & otherwise \end{cases}$
- If  $p = dislikes(m, r_{undesired})$ , then  $\delta(p, a) = 1 - \delta(likes(m, r_{undesired}), a)$

One can use other semantic similarity measures between business processes [25,26]. This issue is not discussed here and it is beyond the scope of this study.

## 4.2 Complex Preferences

To compute the satisfiability degree of complex preferences, we first construct a *preference tree*  $t_p$  that represents the complex preference structure of a set of preferences  $S_p$ . In that preference tree, the nodes represent atomic preferences and the edges represent a *more important than* relation (*prioritized preference*, denoted by  $\&$ ) from parent to child. Preferences belonging to the same level and having the same parent express *Pareto preference*, denoted by  $\otimes$ . Each level  $i$  of the tree is associated with a weight  $\omega_i = 1/i$  except the *level*0.

For example, consider the preference tree of  $q_1$  in Figure 3. Preference  $p_{11}$  is an atomic preference that is not component of any complex preference.  $p_5 : \&(p_2, p_3)$  is a complex preference composed of preferences  $p_2$  and  $p_3$ ; it means that  $p_2$  is more important than  $p_3$ .  $p_7 : \otimes(p_3, p_4)$  is a complex preference composed of preferences  $p_3$  and  $p_4$ ; it means that  $p_3$  and  $p_4$  are equally important.

Considering that each atomic preference  $p_i$  has a satisfiability degree  $\delta_i$ , a new satisfiability degree  $\delta'_i$  is computed taking into account the weight  $\omega_i$  underlying  $p_i$  in the spirit of [8].  $\delta'_i$  is defined using the formula (II) (we assume that  $\max_{i=1,n} w_i = 1$ ).

$$\delta'_i = \max(\delta_i, 1 - \omega_i) \quad (1)$$

This new interpretation of  $p_i$  considers as acceptable any value outside of its support with the degree  $1 - \omega_i$ . It means that the larger  $\omega_i$  (i.e.,  $p_i$  is important), the smaller the degree of acceptability of a value outside the support of  $p_i$ . At the end, we have calculated the satisfiability degree of user atomic preferences considering their constructors and the complex preferences composing them.

## 5 Process Model Similarity: A Linguistic Quantifier-Based Method

We describe here a method to compute preference satisfiability between process model graphs. We also discuss a method to assess the structural similarity

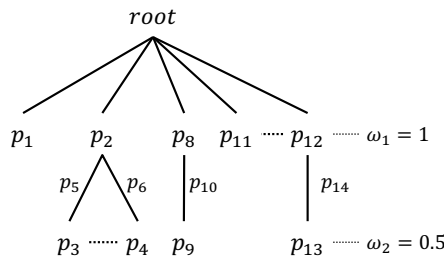


Fig. 3. Sample preference tree

between two process model graphs. Both degrees will be used to rank potential targets (see Section 6). We precise that this work is not interested in discovering a mapping between two process models; we suppose a mapping already exists such that we can compare matched activities annotations against user preferences. In this issue, please consider the work in [4] for an algorithm that returns a mapping between two process models.

To evaluate the structural similarity of two graphs  $q$  and  $t$ , we propose to use a graph matching algorithm like in [4]. This algorithm returns a mapping  $M$  and a set  $E$  of edit operations necessary to transform  $q$  into  $t$ . A mapping between  $q$  and  $t$  is a set of pairs  $(v, w)$ , such that  $v$  is an activity of  $q$  and  $w$  is an activity of  $t$ . The edit operations considered are simple graph operations: node/edge deletion and addition. Figure 4 illustrates a mapping between query graph  $q_1$  and target graph  $t_1$ . Let  $SS(v, w)$  denotes the structural similarity between activities  $v$  and  $w$ ; we use the metric proposed in [4]. Let  $\delta(q_1.S_p, t_1.S_a)$  be the satisfiability degree between global preferences and annotations and let  $\delta(v, w)$  be the satisfiability degree between activities  $v$  and  $w$  (see Section 4).

Next, we rely on the linguistic quantifier “almost all” for the similarity evaluation process. This quantifier is a relaxation of the universal quantifier “all” and constitutes an appropriate tool to avoid empty answers since it retrieves elements that would not be selected when using the quantifier “all”.

## 5.1 Preference Satisfiability between Process Models

A natural user interpretation of the similarity between query and target PMs according to preferences is given by the truth degree of the following proposition:

$\gamma_1$ : Almost all preferences of  $q$  are satisfied by  $t$

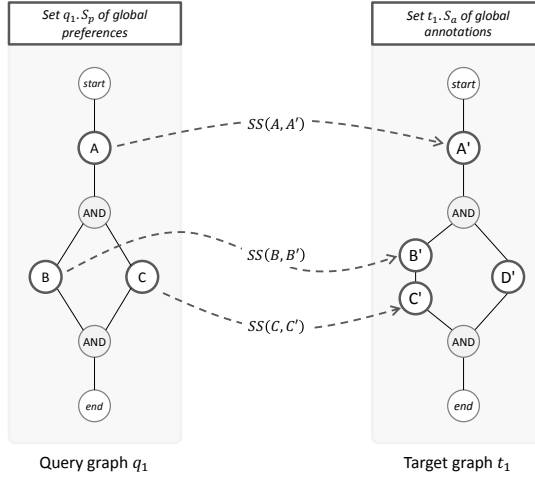
The above statement is a fuzzy quantified proposition of the form “ $Q X$  are  $P$ ”, where (i)  $Q$  is a relative quantifier (e.g., *almost all*, *around half*, etc.) [27] which is defined by a function  $\mu_Q$  such as  $\mu_Q(\varpi)$  is the degree of truth of “ $Q X$  are  $P$ ” when a proportion  $\varpi$  of elements of  $X$  fully satisfy  $A$  and the other elements being not satisfied; (ii)  $X$  is a set of elements; (iii)  $P$  is a fuzzy predicate. In [28], a decomposition method to compute the truth degree  $\delta_\gamma$  of  $\gamma : Q X$  are  $P$  is proposed. The method is a two-step procedure:

- Let  $\Omega = \{\mu_1, \dots, \mu_n\}$  be a set of degrees of the elements of  $X$  w.r.t.  $P$ , ordered in decreasing way; i.e.  $\mu_1 \geq \dots \geq \mu_n$ ;
- The truth degree  $\delta_\gamma$  is given by the equation (2), where  $\mu_Q(i/n)$  is a membership degree of the element  $i/n$  to  $Q$ .

$$\delta_\gamma = \max_{1 \leq i \leq n} \min(\mu_i, \mu_Q(i/n)) \quad (2)$$

In our case,  $\Omega = \{\mu_1 : \delta'_1, \dots, \mu_n : \delta'_n\}$  is the set of satisfiability degrees of all (global and activity) atomic preferences of query  $q$ , where  $\delta'_i$  is the satisfiability degree of an atomic preference  $p_i$  computed by formula (1). The semantics of





**Fig. 4.** Sample mapping  $M$  between query graph  $q_1$  and target graph  $t_1$

the linguistic quantifier *almost all* is given in Table 2. In this case, (i) the user is totally satisfied if at least 80% of preferences are satisfied and (ii) the user is not satisfied at all if at most 50% of preferences are satisfied.

## 5.2 Structural Similarity between Process Models

Similarly, we can apply the technique of fuzzy quantifiers to obtain a structural similarity degree between two process models. The structural similarity between a query and target process models can be given by the truth degree of the following propositions “ $\gamma_1, \gamma_2$  and  $\gamma_3$ ” (defined in Table 2):

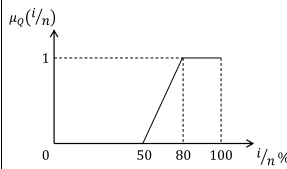
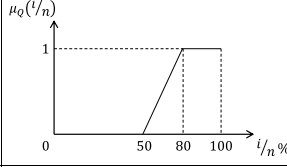
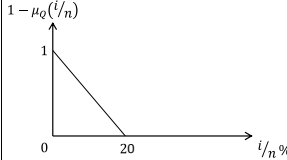
- $\gamma_2$ : Almost all the activities of  $q$  are mapped with activities of  $t$ , and  
 $\gamma_3$ : Almost no edit operation is necessary to transform  $q$  into  $t$

The truth degree of proposition  $\gamma_2$  is obtained from the formula (2), where  $\Omega = \{\mu_1 : SS_1, \dots, \mu_n : SS_n\}$  is the set of semantic similarity degrees of all mapped activities of  $q$ , and  $SS_i$  is the semantic similarity degree of a query activity  $v$  mapped with a target activity  $w$ . In the case of the proposition  $\gamma_3$ , the expression “almost no edit operation is necessary to transform  $q$  into  $t$ ” is equivalent to the expression “almost all edit operations are *not* necessary to transform  $q$  into  $t$ ”. Therefore, its truth degree is computed as follows:

$$\delta_\gamma = \max_{1 \leq i \leq n} \min(1 - \mu_i, 1 - \mu_Q^{(i/n)}) \quad (3)$$

In this case,  $\Omega = \{\mu_1 : C_1, \dots, \mu_n : C_n\}$  is the set of transformation costs of mapped target activities with the corresponding activities of  $q$ , and  $C_i$  is the transformation cost of a target activity  $w$  into a query activity  $v$ .

**Table 2.** Decomposition-based interpretations of propositions  $\gamma_1, \gamma_2, \gamma_3$

PROPOSITION	SET X	MEMBERSHIP FUNCTION $\mu_Q(i/n)$
$\gamma_1$	$X = \{\mu_1: \delta(v_1, w_1), \dots, \mu_n: \delta(v_n, w_n)\}$ , where $n$ is the number of mapping elements	
$\gamma_2$	$X = \{\mu_1: SS(v_1, w_1), \dots, \mu_n: SS(v_n, w_n)\}$ , where $n$ is the number of mapping elements	
$\gamma_3$	$X = \{\mu_1: C(v_1), \dots, \mu_n: C(v_n)\}$ , where $n$ is the number of query activities	

So, the structural similarity between  $q$  and  $t$  is evaluated as follows:

$$SS = \min(\delta_{\gamma_2}, \delta_{\gamma_3}) \tag{4}$$

In our approach, we consider particularly the formulae (2) and (3), where  $\mu_Q(i/n) = i/n$ . Thus, the meaning of delivered degrees has a simple and clear semantics for the user [29]. The evaluation of  $\gamma_1, \gamma_2$  and  $\gamma_3$  means that:

"At least  $\delta_{\gamma_1}^*$  % of preferences of  $q$  are satisfied by  $t$  to at least a degree of  $\delta_{\gamma_1}$ , at least  $\delta_{\gamma_2}^*$  % of the activities of  $q$  are mapped with  $t$  to at least a degree of  $\delta_{\gamma_2}$ , and at least  $\delta_{\gamma_3}^*$  % of  $q$  does not need edit operation to transform  $q$  into  $t$  to at least a degree of  $\delta_{\gamma_3}$ " (where  $\delta_{\gamma_i}^* = 100 \times \delta_{\gamma_i}$ ).

## 6 Process Model Ranking

Previous section has presented an fuzzy set-based approach to compute the similarity between one query and one target graphs. In this section, given a set of target graphs that are relevant to the query, we discuss some methods to rank-order these graphs according to their structural and preference similarities. Let  $\delta(q, t, M)$  be the satisfiability degree between query graph  $q$  and target graph  $t$  according to a mapping  $M$ . Similarly, let  $SS(q, t, M, E)$  be the structural similarity between  $q$  and  $t$  according to a mapping  $M$  and a set  $E$  of edit operations. We classify ranking methods into two categories:

*Ranking Methods based on Aggregation.* In this first category, ranking methods aggregate both structural and preference similarities into a unique degree used to rank-order the target graphs. Two kind of aggregations are considered:

*Weighted Average-Based Aggregation.* The weighted average of  $SS(q, t, M, E)$  and  $\delta(q, t, M)$  is given by:

$$rank(q, t) = \omega_{SS} \times SS(q, t, M, E) + (1 - \omega_{SS}) \times \delta(q, t, M) \quad (5)$$

where  $0 < \omega_{SS} < 1$  is a weight assigned to the structural similarity criterion.

*Min-Combination Based Aggregation.* The min-combination method [30] selects the smallest value of the two similarity degrees  $SS(q, t, M, E)$  and  $\delta(q, t, M)$ :

$$rank(q, t) = \min(SS(q, t, M, E), \delta(q, t, M)) \quad (6)$$

*Ranking Method without Aggregation.* The two distinct similarity degrees are used to rank-order target graphs. The answers are ranked by using the *lexicographic order*. A priority is given to the structural similarity while the preference similarity is only used to break ties.

## 7 Illustrative Example

We give here an example of service discovery for query  $q_1$  of Figure 2. We consider a set  $\{t_1, \dots, t_8\}$  of eight potential answers to  $q_1$  retrieved by a matchmaking algorithm as discussed in Section 5. First, we compute the preference satisfiability between  $q_1$  and the potential target graphs (see Section 5.1). Next, we compute the structural similarity between  $q_1$  and the potential targets (Section 5.2). Then, we apply the ranking methods described in Section 6. To illustrate, we evaluate the preference satisfiability and structural similarity between  $q_1$  and target  $t_1$  of Figure 1. We consider the mapping between them as depicted in Figure 4.

**Preferences Satisfiability.** First, the satisfiability degree  $\delta'_i$  of each preference  $p_i$  of  $q_1$  is calculated as shown in Table 3. For instance, the satisfiability degree  $\delta_2 = \delta(p_2, a_2)$  between preference  $p_2$  and annotation  $a_2$  is obtained by function  $\mu_{max}[reliability]$ . According to equation (II) and the generated preference tree, the new interpretation of the satisfiability degrees is presented in column  $\delta'_i$ . Second, we apply the truth degree described in Section 5.1 to obtain the global satisfiability degree between  $q_1$  and  $t_1$ , as follows:  $\delta_{\gamma_1}(q_1, t_1) = \max(\min(1, \mu_Q(1/9)), \dots, \min(0.5, \mu_Q(9/9))) = 0.67$ . This means that at least 67% of preferences of  $q_1$  are satisfied by  $t_1$  to at least a degree 0.67.

**Structural Similarity.** Assume now that the structural similarities between activities are given by  $SS(A, A') = 0.72$ ,  $SS(B, B') = 0.85$  and  $SS(C, C') = 0.66$ , and the costs of transformation of target activities are  $C(start) = C(end) = C(A') = 0$ ,  $C(AND-split) = 0.1$ ,  $C(B') = C(C') = 0.2$ ,  $C(D') = 0.4$ ,  $C(AND-join) = 0.1$ . In a similar way, the structural similarity degree between  $q_1$  and  $t_1$  is obtained as  $\delta_{\gamma_2}(q_1, t_1) = 0.66$  and  $\delta_{\gamma_3}(q_1, t_1) = 0.75$ . Now,

**Table 3.** Satisfiability degrees of each pair of matched activities

SATISFIABILITY DEGREE CALCULATION				
ATOMIC PREFERENCES			COMPLEX PREFERENCES	
PREF.	MEMBERSHIP FUNCTION	$\delta_i$	PREFERENCE TREE	$\delta'_i$
$p_1$	-	$\delta(p_1, a_1) = 1$		$\delta'_1 = 1$
$p_2$	$\mu_{\max}[\text{reliability}] = (50,80,100,100)$	$\delta(p_2, a_3) = 1$		$\delta'_2 = 1$
$p_3$	$\mu_{\text{around}}[\text{respTime}] = (10,20,20,30)$	$\delta(p_3, a_2) = 0.5$		$\delta'_3 = 0.5$
$p_4$	$\mu_{\text{between}}[\text{cost}] = (0,10,20,30)$	$\delta(p_4, a_4) = 0.5$		$\delta'_4 = 0.5$
$p_8$	$\mu_{\text{around}}[\text{respTime}] = (10,20,20,30)$	$\delta(p_8, a_6) = 1$		$\delta'_8 = 1$
$p_9$	$\mu_{\min}[\text{respTime}] = (0,0,20,60)$	$\delta(p_9, a_5) = 0.9$		$\delta'_9 = 0.9$
$p_{11}$	$\mu_{\max}[\text{reliability}] = (50,80,100,100)$	$\delta(p_{11}, a_{10}) = 1$		$\delta'_{11} = 1$
$p_{12}$	$\mu_{\text{between}}[\text{cost}] = (4,8,15,19)$	$\delta(p_{12}, a_9) = 0.75$		$\delta'_{12} = 0.75$
$p_{13}$	$\mu_{\text{around}}[\text{respTime}] = (5,20,20,30)$	$\delta(p_{13}, a_8) = 0$		$\delta'_{13} = 0.5$

**Table 4.** Structural similarity and preference satisfiability degrees of a set of target graphs.

TARGET GRAPH	STRUCTURAL SIMILARITY $SS$	SATISFIABILITY DEGREE $\delta$
$t_1$	0.66	0.67
$t_2$	0.29	0.72
$t_3$	0.85	0.40
$t_4$	0.78	0.35
$t_5$	0.78	0.21
$t_6$	0.68	0.72
$t_7$	0.66	0.72
$t_8$	0.66	0.35

**Table 5.** Ranking of target graphs according to weighted average, min-combination and lexicographic order methods.

WEIGHTED AVERAGE	MIN-COMBINATION	LEXICOGRAPHIC ORDER
$t_3$ $wa = 0.74$	$t_6$ $mc = 0.68$	$t_3$
$t_6$ $wa = 0.69$	$t_7$ $mc = 0.66$	$t_4$
$t_7$ $wa = 0.68$	$t_1$ $mc = 0.66$	$t_5$
$t_4$ $wa = 0.67$	$t_3$ $mc = 0.40$	$t_6$
$t_1$ $wa = 0.66$	$t_4$ $mc = 0.35$	$t_7$
$t_5$ $wa = 0.64$	$t_8$ $mc = 0.35$	$t_8$
$t_8$ $wa = 0.58$	$t_2$ $mc = 0.29$	$t_1$
$t_2$ $wa = 0.40$	$t_5$ $mc = 0.21$	$t_2$

$SS(q, t, M, E) = \min(\delta_{\gamma_2}, \delta_{\gamma_3}) = 0.66$ , which means that at least 66% of query activities are mapped to at least a degree 0.66 and at most 66% of target activities have transformation cost to at most 0.66.

**Ranking.** Consider the preference satisfiability and structural similarity degrees of each potential target presented in Table 4. Table 5 summarizes the results of the different ranking methods discussed in Section 6 (where  $\omega_{SS} = 0.75$ ).

The Lexicographic order ensures that the first in the ordered list is that having the best structural similarity and, in case of ties, that having the best preference satisfiability. For example  $t_3$  is better than all the other target graphs because its structural similarity is the greatest value. However, a drawback of this method is that the rank can be too drastic, as for the case of  $t_5$  : (0.78, 0.21) and  $t_6$  (0.68, 0.72). In a such case, the idea of a weighted average is more suitable since it allows for a compensation. Now, with the weighted average  $t_6$  is better than  $t_5$  but generally it does not provide a clear semantics of the induced order.

Finally, the min-combination method relies on the worst satisfiability for each service and does not highlight the structural similarity versus the preference satisfiability. The weighted min-combination can overcome the above limitation.

## 8 Complexity Analysis and Experimental Results

In what follows, we first study the complexity of our approach and then present the set of experiments conducted to (i) measure the time the preference evaluation task takes in the process model matchmaking and to (ii) evaluate the effectiveness of the results.

### 8.1 Complexity Analysis

The complexity of our solution can be analyzed in three steps. In the case of the evaluation of atomic preferences, it implies the time to find the relevant annotation and the time to evaluate the atomic preference itself. Considering the time to find the relevant annotation in a set of  $m$  annotations per activity, the time to evaluate all the  $n$  atomic preferences of a user query is  $O(n \cdot m)$ , if we consider that to evaluate an atomic preference is either trivial in the case of numerical preferences or *polynomial* in the case of non-numerical preferences [24]. The complexity remains *polynomial* even if we consider that each query activity defines as much atomic preferences as the number of considered non-functional properties.

In the case of the evaluation of complex preferences, the worst case is when all atomic preferences of each query activity are aggregated by complex preferences. Therefore, we have the time to evaluate each atomic preference and the time to construct and to evaluate the preference tree. The time to construct the tree is *linear*, since we only analyze the complex preferences, which are never more than half of the total of preferences. The time to evaluate the preference tree is also *linear* w.r.t. the quantity of preferences. Finally, the evaluation of the linguistic quantifiers is also *polynomial*, since it consists of an ordering of degrees plus the choosing for an element satisfying a condition. As a conclusion, we can see that the complexity of our solution is *polynomial*.

### 8.2 Experiments Setup

To run our experiments, we implemented a prototype that works over the system proposed by [4]. We adapted their business process model to consider non-functional annotations and their query model to consider preference annotations. We also reused their test set of process models and queries.

The main goals of our experiments are to: (i) Measure the overhead time w.r.t. the matchmaking time. It is important to note that matchmaking algorithms are NP-complete; (ii) Measure the effectiveness of our results by means of Normalized Discounted Cumulative Gain (NDCG) score; (iii) Compare the effectiveness of our results with the crisp logic-based approach presented in [9].

---

<sup>4</sup> The least common ancestor and the distances between concepts in an ontology can be calculated previously, off query time.

*Test set setup.* In our experiments, we considered two real-data sets containing target graphs: the first one is composed of 24 graphs of flight reservation domain having an average size of 18 activities, while the second has 32 graphs of hotel reservation domain having an average size of 12 activities, which means that the graphs have a quite considerable size. The graphs in each group have similar structure, which induces the matchmaking results to be close and not empty. We annotated the activities of each target with 10 annotations, one for each of the 10 considered QoS attributes. The attributed values were generated randomly.

Three different query process models were proposed: *FlightReservationQuery1* (FR-1), *FlightReservationQuery2* (FR-2) and *HotelReservationQuery1* (HR-1). The activities of these queries were annotated with *textual preferences* pertinent to the domain of each activities. These textual preferences were described using natural language and their semantics considered the concept of atomic and complex preferences.

We generated adapted versions of these queries according to the model proposed in our approach (Fuzzy logic-based approach) and in [9] (Crisp logic-based approach), since our objective is also to compare both approaches.

*Definition of the ideal ranking.* A group of experts was invited to manually analyze the satisfiability of each target graph w.r.t. to the textual queries considering the behavior specification and QoS preferences. After the analysis, the experts gave one single note to each target in a 1-7 Likert scale (1 for strongly different, 7 for strongly similar). At the end, an expert ranking was defined for each query.

*Experiment execution.* Five rankings were obtained after query evaluation:

1. (*Crisp AVG*) Results from crisp approach ordered by the weighted average of structural similarity and preference satisfiability;
2. (*Crisp LEX*) Results from crisp approach ordered by the lexicographic order of structural similarity and preference satisfiability;
3. (*Fuzzy AVG*) Results from our approach ordered by the weighted average of structural similarity and preference satisfiability;
4. (*Fuzzy LEX*) Results from our approach ordered by the lexicographic order of structural similarity and preference satisfiability;
5. (*Fuzzy MIN*) Results from our approach ordered by the min-combination of structural similarity and preference satisfiability;

From the results of each ranking, the top-k targets were selected and the NDCG scores were computed. The overhead time was calculated over the whole set of results. All the evaluations were conducted on a machine with an Intel i5 2.8GHz processor, 4GB of memory, running Windows 7 OS and Java VM version 1.6.

### 8.3 Experimental Results

As can be seen from the results presented in Table 6, the extra time taken to evaluate the hard preferences is insignificant w.r.t. the matchmaking time. It barely represents 1% of the matchmaking time.

**Table 6.** Matchmaking and preference evaluation times

Query/Time (ms)	AMT	APET
FR-1	82.8	0.9
FR-2	180.8	0.8
HR-1	50.9	0.8

Legend:

- *AMT*: Average Matchmaking Time

- *APET*: Average Preference Evaluation Time

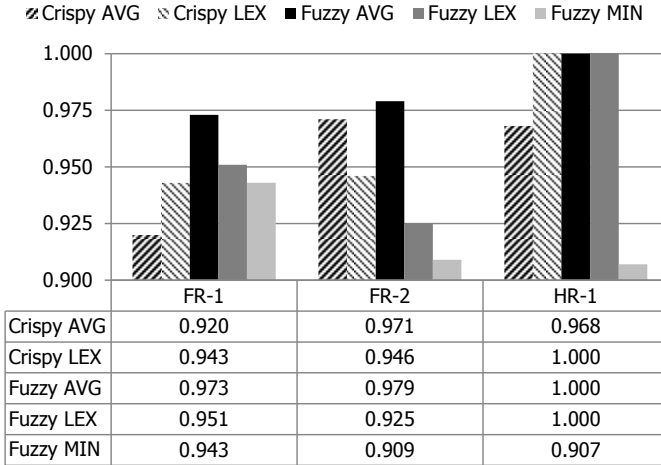
**Fig. 5.** Effectiveness using NDCG measure

Figure 5 presents the NDCG scores according to the different approaches and their proposed ranking methods. In this case, the closer the score is to 1, the closer the ranking proposed by the corresponding approach is to the ranking defined by the experts. For query FR-1, all scores of fuzzy approaches overcame the crisp ones. For query FR-2, fuzzy AVG score was better than crisp results. For query HR-1, some crisp and fuzzy approaches provided the expert ranking.

The results clearly show that both crisp and fuzzy approaches provided a good effectiveness, although the scores of fuzzy AVG method always overcome crisp scores. Fuzzy LEX score was very unstable w.r.t. to the expert ranking since the experts tried to find a compromise between structure and quality, whereas in lexicographic order, the priority is given to the structural similarity while the preference similarity is only used to break ties. The restrictiveness of Fuzzy MIN proved to be very ineffective, although the semantics of its results is very strong.

## 9 Conclusion

In this paper, we have proposed an approach for web service selection and ranking. In our approach, the evaluation process takes into account two aspects: (i)

structural similarity, and (ii) preference satisfiability. User preferences are modelled with fuzzy predicates. Both preference satisfiability and structural similarity are interpreted thanks to linguistic quantifiers. This makes the matchmaking process more flexible and realistic. Some ranking methods have been proposed as well. We also introduced a complexity analysis of our solution and we showed that the preference evaluation does not raise the complexity of process model matchmaking. Finally, we presented the set of experiments conducted over an implementation of our approach to measure the effectiveness of the results. These experiments showed that our approach gathered with the weighted average proposes a better ranking than the considered crisp solution.

As future work, we plan to apply fuzzy set-based techniques to evaluate hard constraints over QoS attributes (such as  $cost \geq 20$ ) in process model matchmaking. We also plan to investigate other fuzzy aggregation and ranking methods that minimize the restrictiveness of those presented in this work.

**Acknowledgment.** This work has received support from the French National Agency for Research (ANR) on the reference ANR- 08-CORD-009.

## References

1. 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, ser. AAMAS 2006, pp. 915–922 (2006)
2. Dijkman, R., 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)
3. van Dongen, B., Dijkman, R., Mendling, J.: Measuring similarity between business process models. In: Bellahsene, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008)
4. Grigori, D., Corrales, J.C., Bouzeghoub, M., Gater, A.: Ranking bpm processes for service discovery. IEEE Transactions on Services Computing 3, 178–192 (2010)
5. Şora, I., Lazăr, G., Lung, S.: Mapping a fuzzy logic approach for qos-aware service selection on current web service standards. In: ICCO-CONTI, pp. 553–558 (2010)
6. Zhang, Y., Huang, H., Yang, D., Zhang, H., Chao, H.-C., Huang, Y.-M.: Bring qos to p2p-based semantic service discovery for the universal network. Personal Ubiquitous Computing 13(7), 471–477 (2009)
7. Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: Proc. of ECOWS, pp. 265–274 (2006)
8. Dubois, D., Prade, H.: Using fuzzy sets in flexible querying: Why and how? In: Proc. of FQAS, pp. 89–103 (1996)
9. Lemos, F., Gater, A., Grigori, D., Bouzeghoub, M.: Adding preferences to semantic process model matchmaking. In: Proc. of GAOC (2011)
10. Hristidis, V., Koudas, N., Papakonstantinou, Y.: Prefer: A system for the efficient execution of multi-parametric ranked queries. In: SIGMOD Conference, pp. 259–270 (2001)
11. Bosc, P., Pivert, O.: Sqlf: a relational database language for fuzzy querying. IEEE Trans. on Fuzzy Systems 3(1), 1–17 (1995)



12. Chomicki, J.: Preference formulas in relational queries. *ACM Transactions on Database Systems* 28(4), 427–466 (2003)
13. Kießling, W.: Foundations of preferences in database systems. In: *VLDB. VLDB Endowment*, pp. 311–322 (2002)
14. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: *ICDE*, pp. 421–430 (2001)
15. Zadeh, L.A.: Fuzzy sets. *Information and Control* 8(3), 338–353 (1965)
16. D’Mello, D.A., Kaur, I., Ram, N., Ananthanarayana, V.S.: Semantic web service selection based on business offering. In: *Proc. of EMS*, pp. 476–481 (2008)
17. Agarwal, S., Lamparter, S., Studer, R.: Making Web services tradable: A policy-based approach for specifying preferences on Web service properties. *Web Semantics: Science, Services and Agents on The World Wide Web* 7(1), 11–20 (2009)
18. Sathya, M., Swarnamugi, M., Dhavachelvan, P., Sureshkumar, G.: Evaluation of qos based web- service selection techniques for service composition. *IJSE* 1, 73–90 (2010)
19. Lin, M., Xie, J., Guo, H., Wang, H.: Solving qos-driven web service dynamic composition as fuzzy constraint satisfaction. In: *Proc. of EEE*, pp. 9–14 (2005)
20. Chen, M.-F., Gwo-Hshiong, T., Ding, C.: Fuzzy mcdm approach to select service provider. In: *Proc. of ICFS* (2003)
21. Xiong, P., Fanin, Y.: Qos-aware web service selection by a synthetic weight. In: *Proc. of FSKD* (3), pp. 632–637 (2007)
22. Hafeez, O., Chung, S., Cock, M.D., Davalos, S.: Towards an intelligent service broker with imprecise constraints: Fuzzy logic based service selection by using sawsdl. *TCSS 702 Design Project in Computing and Software Systems, University of Washington* (2008)
23. Dumas, M., García-Bañuelos, L., Polyvyany, 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)
24. Wu, Z., Palmer, M.S.: Verb semantics and lexical selection. In: *Proc. of ACL*, pp. 133–138 (1994)
25. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: *Proc. of the Fourth APCCM*, vol. 67, pp. 71–80 (2007)
26. Koschmider, A., Oberweis, A.: How to detect semantic business process model variants? In: *Proc. of the 2007 ACM SAC*, pp. 1263–1264 (2007)
27. Glöckner, I.: *Fuzzy Quantifiers in Natural Language: Semantics and Computational Models*. Der Andere Verlag, Osnabrück (2004)
28. Yager, R.R.: General multiple-objective decision functions and linguistically quantified statements. *International Journal of Man-Machine Studies* 21, 389–400 (1984)
29. Liétard, L.: A new definition for linguistic summaries of data. In: *IEEE World Congress on Computational Intelligence, Fuzzy*. IEEE, Hong-Kong (2008)
30. Dubois, D., Prade, H.: Handling bipolar queries in fuzzy information processing. In: Galindo, J. (ed.) *Handbook of Research on Fuzzy Information Processing in Databases*. IGI Global, pp. 97–114 (2008)

# Efficient Retrieval of Similar Business Process Models Based on Structure

## (Short Paper)

Tao Jin<sup>1,2</sup>, Jianmin Wang<sup>2</sup>, and Lijie Wen<sup>2</sup>

<sup>1</sup> Department of Computer Science and Technology, Tsinghua University, China

<sup>2</sup> School of Software, Tsinghua University, China

**Abstract.** With the business process management technology being more widely used, there are more and more business process models, which are typically graphical. How to query such a large number of models efficiently is challenging. In this paper, we solve the problem of querying similar models efficiently based on structure. We use an index named TaskEdgeIndex for query processing. During query processing, we estimate the minimum number of edges that must be contained according to the given similarity threshold, and then obtain the candidate models through the index. Then we compute the similarity between the query condition model and every candidate model based on graph structure by using maximum common edge subgraph based similarity, and discard the candidate models that actually do not satisfy the similarity requirement. Since the number of candidate models is always much smaller than the size of repositories, the query efficiency is improved.

## 1 Introduction

The wide use of business process management technology results in a large number of business process models. These models are typically graphical. For example, there are more than 200,000 models in China CNR Corporation Limited. How to query such a large number of models efficiently is challenging. For example, before a designer creates a new business process model, if s/he can obtain the models nearly containing her/his draft model (which is always incomplete) as a subgraph, and then continue to work on these models instead of starting from scratch, it would save a lot of time and it is less error-prone. Since the number of models is large, the efficiency of similarity retrieval is very important.

The problem to be solved in this paper can be described as follows. Given a query condition model  $q$ , quickly find all the models (notated as  $S$ ) in the model repository  $R$  satisfying that, for every model  $m$  in  $S$ , we can find a subgraph  $sub$  in  $m$ , the similarity between  $sub$  and  $q$  based on structure must not be less than a specified threshold  $\theta$ .

There are many methods to compute the similarity between graphs based on their structure such as the method based on graph edit distance, maximum common subgraph. In this paper, we use the maximum common edge subgraph

(MCES) based similarity, which is widely used. For example, MCES based similarity was used in [12]. MCES based similarity is superior to graph edit based similarity in that no particular edit operations together with their costs need to be defined. Since the computation of MCES based similarity is NP-hard, it would be much time-consuming if we scan the repository sequentially and compute the similarity between the query condition model and every model in the repository. In this paper, we use a filtering-verification framework to reduce the number of times of MCES based similarity computation. Our contributions in this paper can be summarized as follows.

- We apply MCES based similarity algorithm to business process models.
- We use an index named *TaskEdgeIndex* to speed up the query processing.
- We implement our approach in the BeehiveZ system and do some experiments to evaluate our approach.

There are many different notations used to capture the business processes, such as *BPMN*, *BPEL*, *XPDL*, *EPC*, *YAWL*, *PNML* and so on. Among all the notations, Petri net has good formal foundation and simple graph notations, so that it can not only be understood and used easily but also can be used for analysis. Many researchers have worked on the transformation from other notations to Petri nets. You can refer to [3] for an overview. To deal with business process models with different formats in an uniform way, we assume that all the models in the repository are represented as or transformed to Petri nets.

## 2 Preliminaries

Petri net was introduced into business process management area for modeling, verification and analysis in [4]. The details of Petri net can be found in [5].

**Definition 1 (Petri net).** *A Petri net is a triple  $N = (P, T, F)$ , with  $P$  and  $T$  as finite disjoint sets of places and transitions ( $P \cap T = \emptyset$ ), and  $F \subseteq (P \times T) \cup (T \times P)$  is a set of arcs (flow relations). We write  $X = (P \cup T)$  for all nodes of a Petri net. For a node  $x \in X$ ,  $\bullet x = \{y \in X \mid (y, x) \in F\}$ ,  $x \bullet = \{y \in X \mid (x, y) \in F\}$ .*

In this paper, we measure the similarity between two Petri nets using MCES based similarity. To obtain the MCES of two given graphs, first we get the line graphs of the original graphs, and then get the modular product graph of the line graphs, finally, we get the maximum clique of the modular product graph. When we project the maximum clique back to the original graphs, we can get the MCES of the two given original graphs. The details can be found in [1]. In the following, we give the corresponding definitions for Petri net.

**Definition 2 (Task edge).** *Given a Petri net  $N = (P, T, F)$ , a task edge is a pair  $te = \langle t_1, t_2 \rangle$  satisfying that  $t_1, t_2 \in T \wedge \exists p \in P (p \in t_1 \bullet \wedge p \in \bullet t_2)$ . We distinguish the source and target of a task edge as  $s(te) = t_1$ ,  $t(te) = t_2$ , and all the task edges is denoted as  $TE(N)$ .*

**Definition 3 (Task edge graph).** A directed graph  $TEG(N) = (V, E)$  is the task edge graph of a Petri net  $N = (P, T, F)$  satisfying:  $V = TE(N)$ ,  $\forall te_1, te_2 \in V$ ,  $te_1$  is adjacent to  $te_2$  iff  $t(te_1) = s(te_2)$ . We denote the incident task as  $adj(te_1, te_2) = t(te_1) = s(te_2)$ , which means that the task edge  $te_1$  and  $te_2$  share the same task. If  $te_1$  is adjacent to  $te_2$ ,  $\langle te_1, te_2 \rangle \in E$ . We denote the set of vertices of a graph  $G$  as  $V(G)$  and the set of edges as  $E(G)$ .

**Definition 4 (Task edge graph modular product).** The modular product of two task edge graphs  $TEG_1$  and  $TEG_2$ ,  $MPG(TEG_1, TEG_2)$ , is defined on the vertex set  $V(TEG_1) \times V(TEG_2)$  where the respective vertex labels are compatible and two vertices  $(u_i, v_i)$  and  $(u_j, v_j)$  of modular product are adjacent when  $(u_i, u_j) \in E(TEG_1) \wedge (v_i, v_j) \in E(TEG_2) \wedge w(u_i, u_j) = w(v_i, v_j)$  or  $(u_i, u_j) \notin E(TEG_1) \wedge (v_i, v_j) \notin E(TEG_2)$ . Here  $w(u_i, u_j) = w(v_i, v_j)$  indicates that the labels of  $adj(u_i, u_j)$  and  $adj(v_i, v_j)$  are compatible. Compatible labels means that the labels are equal when label similarity is not considered or similar when label similarity is considered.

**Definition 5 (Sub-similarity).** Given two business process models represented as Petri nets  $N_1$  and  $N_2$ , we can construct the task edge graph modular product first, and then find the biggest clique in the modular product, which is the maximum common subgraph between the corresponding task edge graphs, denoted as  $\omega(teg(N_1), teg(N_2))$ . The MCES based similarity between  $N_1$  and  $N_2$  can be measured as:

$$subSim(N_1, N_2) = \frac{|V(\omega(teg(N_1), teg(N_2)))|}{\min(|TE(N_1)|, |TE(N_2)|)}. \quad (1)$$

### 3 Index Construction and Query Processing

To improve the query efficiency, we use an index named *TaskEdgeIndex* as a filter to obtain a set of candidate models which contain at least a specific number of task edges in the query condition model. Then in the verification stage, we calculate the MCES based similarity between query condition model and every candidate model and discard all the candidate models with the similarity less than the specified threshold  $\theta$ . Since the number of candidate models is always much smaller than the size of repository, the query efficiency can be improved. Moreover, the computation of task edges for the models in the repository is completed during the construction of *TaskEdgeIndex*, so during the query processing the time is saved.

#### 3.1 Index Construction

The *TaskEdgeIndex* sets up the relation between task edges and models, and it has two parts (we only discuss the index on logical level here, the information of implementation can be found in Section 4). One part is a forward index (notated as *FI*), which stores the mapping from models to task edges. The items indexed

in  $FI$  are like  $(m, TE)$ , in which,  $m$  is denoted as a model, represented as a Petri net, and  $TE$  is the set of task edges of the corresponding model.  $FI$  can be used to obtain the task edges of a model. The other part is an inverted index (notated as  $II$ ), which stores the mapping from task edges to models. The items indexed in  $II$  are like  $(te, te.list)$ , in which,  $te$  is denoted as a task edge, and  $te.list$  is denoted as a set of models where the corresponding  $te$  appears.  $II$  can be used to obtain all the models that contain a specific task edge.

Given a model represented as a Petri net, Algorithm 1 extracts all the task edges. Every place in the model is traversed and the corresponding task edges are extracted.

---

**Algorithm 1.** Task edges extraction (*getTaskEdges*)
 

---

**input** : a model  $m$  represented as a Petri net  
**output**: all the task edges in the given model  $m$

```

1 foreach  $p$  in  $P$  do
2   foreach  $tpre \in \bullet p$  do
3     foreach  $tsuc \in p \bullet$  do
4        $TE.add((tpre, tsuc));$ 
5 return  $TE$ ;
```

---

Based on the Algorithm 1, the *TaskEdgeIndex* can be constructed as described in Algorithm 2. From Algorithm 2, we can see that when a new model is added to the repository, the index can be updated incrementally. When one model is deleted from the repository, the mapping between the model and its task edges can be deleted from *TaskEdgeIndex* directly.

---

**Algorithm 2.** Add a model to *TaskEdgeIndex*


---

**input**: a model  $m$  represented as a Petri net

```

1  $TE = getTaskEdges(m);$ 
2 foreach  $te$  in  $TE$  do
3    $Fl.add(m, te);$ 
4    $Il.add(te, m);$ 
```

---

### 3.2 Query Processing

Based on *TaskEdgeIndex*, the query processing can be divided into two stages, namely, the filtering stage and the refinement stage. Firstly, in the filtering stage, we extract all the task edges from the query condition model  $q$  by using Algorithm 1, and use the inverted index ( $II$ ) to get the set of candidate models where at least  $\theta \times |TE(q)|$  task edges from the query condition model appear. Secondly, in the refinement stage, for every candidate model, we calculate the MCES based

similarity between it and the query condition model by using Equation 11. If the similarity is less than the specified threshold  $\theta$ , the corresponding candidate model is removed from the candidate set. Finally, all the models satisfying the user’s requirement are returned.

---

**Algorithm 3.** Retrieve the similar models
 

---

```

input : a query condition model  $q$ , and the model similarity threshold  $\theta$ 
output: all the models satisfying the requirement

// filtering stage
1 qTE = getTaskEdges(q);
2 foreach te in qTE do
3   ret.add(II.getModelSet(te));
4 foreach c in ret do
5   mTE = FI.getTaskEdges(c);
6   if  $|mTE \cap qTE| < \theta \times |qTE|$  then
7     ret.remove(c);
// refinement stage
8 foreach c in ret do
9   if subSim(c,q) <  $\theta$  then
10    ret.remove(c);
11 return ret;
```

---

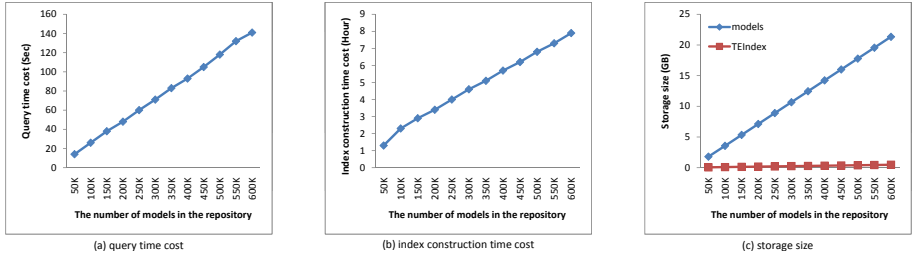
The above procedure is described as Algorithm 3. Here, we can see that the model similarity threshold has effect on the query efficiency, because it affects the size of candidate set after Line 7 executed.

## 4 Tool Support and Evaluation

To evaluate our approach, we implemented it in our system named BeehiveZ. BeehiveZ can be downloaded from <http://code.google.com/p/beehivez/>. BeehiveZ was developed in Java, all the models were stored as Text in MySQL RDBMS. The *TaskEdgeIndex* was managed by Apache Lucene. The model similarity threshold can be configured in BeehiveZ. Since the model similarity has effect on the query efficiency as shown in Section 3.2, to measure the query time cost at the worst case, we set the model similarity threshold to be 0 by default in our experiments, which means that even if there is only one common edge between one model and the query condition model, the model will be returned as a resulting model. During our experiments, we used a computer with Intel(R) Core(TM)2 Duo CPU E8500 @3.16GHz and 2GB memory. This computer ran Ubuntu 9.04 and JDK6. The heap memory for JVM was configured as 1GB.

All the models in our experiments were generated automatically using the rules from [5]. According to 7PMG proposed in [6], models should be decomposed

if they have more than 50 elements, so we generated models with the maximum number of transitions as 50, the number of places and arcs in a model is not configurable. Firstly, we generated 10 query condition models and added them to the repository. And then, we generated more models by batch. At last, there were more than 600,000 models in the repository. The number of models with different number of transitions followed the normal distribution. There were totally 15,605,621 transitions, and the number of transitions with different labels was 242,234.



**Fig. 1.** The performance of *TaskEdgeIndex*

The change of retrieval efficiency can be found in Fig. 1(a). With more and more models added to the repository, it is more time-consuming, but the time is still acceptable. Because for different query condition models, the change of query time is similar, we only show the query time on the query condition model with 26 transitions here. In the above experiment, we set the model similarity threshold to be 0. The query efficiency changes along with the change of model similarity threshold, which can be found in Table 1. There were more than 600,000 models in the repository, and we also used the same query condition model with 26 transitions. With the model similarity threshold increases, the query time cost decreases, which means that *TaskEdgeIndex* can discard more models with a higher model similarity threshold. The index construction time can be found in Fig. 1(b). It shows the accumulated time for index construction from scratch. Since our index can be constructed incrementally, when new models are added to the repository, the index can be updated immediately, the time for index updating is very short. The storage size of index can be found in Fig. 1(c). The storage size of index is less than 3% of the models.

**Table 1.** Query time (sec) with different model similarity thresholds

	0	0.1	0.2	0.4	0.6	0.9	1.0
time(sec)	145.43	11.46	7.87	7.84	7.81	7.69	7.68

## 5 Related Work

There are already some works on business process model query. In [7] the authors used an indexing technique to search for matched process models, it belongs to the category of exact query. BP-QL was proposed in [8], and it is a language used for BPEL model query. The VisTrails system [9] allows users to query business process by example and to refine business processes by analogies. A business process search engine, WISE, was proposed in [10], which returns the most specific business process hierarchies containing matched keywords. A framework was proposed in [11], which is based on a visual query language for business process models named BPMN-Q, and makes use of the robust indexing infrastructure available by RDBMS. In [12], the authors use path indexes to improve the exact business process model query efficiency. In [13], the authors use ordering relation indexes to improve the business process model query efficiency based on behavior. In [14], the authors use an index named TARIndex to improve the similar business process model retrieval based on behavior. But all the above works have not touched the area of similar business process model retrieval based on graph structure, while we solve the problem of efficient similar model retrieval based on structure in this paper. In [15], the authors used some structure features to estimate the similarity between models and an index technique was used, and then graph edit based similarity is measured between some selected models and the query condition model. But it is not guaranteed that all the models satisfying the requirement will be returned as a result, because the relation between the structure feature filtering and the graph edit based similarity measure is not clear. Some correct models may not pass the filtering stage. While through our approach, all the models satisfying the requirement will be returned as resulting models. If one model cannot pass our filtering stage, it means that the investigated model does not contain enough number of common edges with the query condition model, so it is impossible to have a subgraph sufficiently similar to the query condition model according to MCES based similarity.

## 6 Conclusion and Future Work

In this paper we focus on improving the efficiency of similar process model query based on graph structure. We use MCES based similarity to measure the similarity between business process models. To improve the query efficiency, we use an index named *TaskEdgeIndex* to support query processing. The *TaskEdgeIndex* works as a filter, so that we only need to compute the similarity between the query condition model and every model in the candidate set which is obtained with the use of *TaskEdgeIndex*. The size of candidate set is always much smaller than the size of the repository, that is why the query efficiency can be improved. Moreover, the task edges computation of models in the repository is completed during the *TaskEdgeIndex* construction, so during the query processing the time is saved. Analysis and experiments show that our approach is efficient.



However, we only consider the tasks and the execution order between tasks when we measure the similarity between business process models. We will also consider the data processing and the resource allocation in the future.

**Acknowledgments.** The work is supported by the National Science and Technology Major Project (HGJ) of China (No. 2010ZX01042-002-002-01), the National Basic Research Program (973 Plan) of China (No. 2009CB320700), the National High-Tech Development Program (863 Plan) of China (No. 2008AA042301) and an NSF Project of China (No. 61003099).

## References

1. Raymond, J.W., Gardiner, E.J., Willett, P.: RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *Comput. J.* 45(6), 631–644 (2002)
2. Yan, X., Yu, P.S., Han, J.: Substructure Similarity Search in Graph Databases. In: *SIGMOD Conference*, pp. 766–777 (2005)
3. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri Net Transformations for Business Processes - A Survey. *T. Petri Nets and Other Models of Concurrency* 2, 46–63 (2009)
4. 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)
5. Murata, T.: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77(4), 541–580 (1989)
6. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven Process Modeling Guidelines (7PMG). *Information & Software Technology* 52(2), 127–136 (2010)
7. Mahleko, B., Wombacher, A.: Indexing Business Processes based on Annotated Finite State Automata. In: *ICWS*, pp. 303–311 (2006)
8. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying Business Processes. In: *VLDB*, pp. 343–354 (2006)
9. Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.T.: Querying and Re-Using Workflows with VisTrails. In: *SIGMOD Conference*, pp. 1251–1254 (2008)
10. Shao, Q., Sun, P., Chen, Y.: WISE: A Workflow Information Search Engine. In: *ICDE*, pp. 1491–1494 (2009)
11. Sakr, S., Awad, A.: A Framework for Querying Graph-Based Business Process Models. In: *WWW*, pp. 1297–1300 (2010)
12. Jin, T., Wang, J., Wu, N., La Rosa, M., ter Hofstede, A.H.M.: Efficient and Accurate Retrieval of Business Process Models Through Indexing. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) *OTM 2010. LNCS*, vol. 6426, pp. 402–409. Springer, Heidelberg (2010)
13. Jin, T., Wang, J., Wen, L.: Querying Business Process Models Based on Semantics. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) *DASFAA 2011, Part II. LNCS*, vol. 6588, pp. 164–178. Springer, Heidelberg (2011)
14. Jin, T., Wang, J., Wen, L.: Efficient Retrieval of Similar Business Process Models Based on Behavior. Technical report. Tsinghua University (2011)
15. Yan, Z., Dijkman, R., Grefen, P.: Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) *OTM 2010. LNCS*, vol. 6426, pp. 60–77. Springer, Heidelberg (2010)

# Preservation of Integrity Constraints by Workflow

Xi Liu<sup>1,2,3,\*</sup>, Jianwen Su<sup>3,\*\*</sup>, and Jian Yang<sup>4</sup>

<sup>1</sup> State Key Laboratory for Novel Software Technology, Nanjing University, China

<sup>2</sup> Department of Computer Science and Technology, Nanjing University, China

<sup>3</sup> Department of Computer Science, University of California at Santa Barbara, USA

<sup>4</sup> Department of Computing, Macquarie University, Australia

liux@seg.nju.edu.cn, su@cs.ucsb.edu, jian.yang@mq.edu.au

**Abstract.** Integrity constraints on data are typically defined when workflow and business process models are developed. Keeping data consistent is vital for workflow execution. Traditionally, enforcing data integrity constraints is left for the underlying database system, while workflow system focuses primarily on performing tasks. This paper presents a new mechanism that turns a workflow into an equivalent one that will preserve integrity constraints. For a given workflow schema (or model) and a given set of data integrity constraints, an algorithm developed in this paper *injects* additional conditions into the workflow schema that restricts possible execution paths. The modified workflow will guarantee data consistency (i.e., satisfaction of the integrity constraints) whenever the workflow updates the database(s). In addition, we show that our injection mechanism is “conservative complete”, i.e., the conditions inserted are weakest possible. By making workflow execution self-behaving, enforcing integrity constraints over multi-databases is avoided, and constraints specific to a workflow can also be enforced effectively. Mechanisms such as this enhance independence of workflow executions from the environment—a much desired property.

## 1 Introduction

Data integrity is the assurance of data correctness, consistency and completeness. From the database perspective, data integrity can be imposed within a database at its design stage through the use of standard rules and procedures, and maintained through the use of error checking and validation routines [2]. Data is the most important asset for any business to make decisions and gain global competitiveness. Decisions made on data that lack integrity can result in losing opportunities and even losing business.

Database management systems (DBMSs) are developed for storing and managing data that is generated and updated by various applications. Workflow systems are an important class of software systems that manage organizational business processes and normally utilize database systems for storing data, executing tasks, and logging. Workflow has been studied for over a decade [16]. Recently with the emerging web service technology, notations and specifications for workflows have been developed such as

---

\* Supported in part by National Natural Science Foundation of China (No.90818022 and No.61021062) and a grant from China Scholarship Council.

\*\* Supported in part by NSF grant IIS-0812578 and a grant from IBM.

BPMN, BPEL, YAWL, etc. These workflow models mostly focus on the aspect of task flow control and completely rely on the underlying database systems to take care of data integrity. However an enterprise workflow system can run across different agencies, departments and organizations, thus it needs to interact with different databases. Take an online shopping workflow as an example, it may need to communicate with a customer database that is only logically integrated from databases of different branches and other partner companies. Distributed DBMS technology does not provide a satisfactory solution in enforcing effectively data integrity defined across multiple database systems. Even when a DBMS detects a violation, it is often difficult to locate the origin in a workflow that causes the error. Also, these underlying databases can be shared by many applications and workflow systems. On the other hand, there are data integrity constraints specific to individual workflow, i.e., they are “local” to the workflow in question. It is not appropriate to enforce such local constraints on databases shared among different applications including other workflows. With the current trend of using “cloud” as the outsourcing facility for data storage and management, pushing local data integrity constraints into a shared database system may result in undesirable effects. Moreover, it is unclear that cloud would realize mechanisms to maintain data consistency in loosely coupled databases [8, 13].

A database system can only check/validate data integrity. It still relies on applications and workflow to produce the correct data and updates, i.e., adhered to integrity constraints. Therefore, in a complex workflow system interacting with distributed databases, it will become an obstacle to always let the database systems check data integrity and come back to the workflow to make necessary corrections for it to proceed.

To overcome the above discussed problems, we propose a mechanism to make a workflow self-behaving in terms of data integrity. The key novelty is to modify a workflow schema by *injecting* certain conditions according to the defined integrity constraints to guard against inconsistent updates. The data integrity is therefore guaranteed within the workflow, and we further gain the independence of workflow execution from the underlying database systems concerning workflow related data updates.

We develop Integrity Preservation Mechanism (IPM) based on a recent artifact-centric workflow model of [17]. The concept of artifact-centricity in workflow modeling was introduced in [24]. There have been increased studies on design and modeling using artifact-centric [5, 6, 17, 22] or other data-aware approaches [12, 21]. The technical development of this paper uses the artifact-centric modeling language GSM (*Guard-Stage-Milestone*) [17]. The language is a declarative meta-model using event-condition-action rules to capture business stakeholders’ view [9, 18]. We develop a formal model to specify the execution of GSM workflow based on transition systems and the Z notation [25]. Specified integrity constraints are ensured by strengthening the guard of the operations in the execution workflow schema to prohibit updates that may violate integrity. This process is called *guard injection* in this paper.

To make guard injection work properly, the injection must be *strong enough* to prevent any integrity violations and *weak enough* to allow some or even all correct executions to proceed. The technical challenge is to formulate the appropriate balance in the injection algorithm design.

<pre> Customer(<i>custid</i> PRIMARY KEY,         <i>email</i> NOT NULL,         <i>addr</i>,         UNIQUE(<i>email</i>)) Ship(<i>shipid</i> PRIMARY KEY,      <i>ordid</i> NOT NULL,      <i>addr</i> NOT NULL,      <i>name</i> NOT NULL,      <i>from</i> NOT NULL,      <i>ship_stat</i>,      FOREIGN KEY(<i>ordid</i>)      REFERNECES <i>Order</i>) </pre>	<pre> Inventory (<i>invid</i> PRIMARY KEY,           <i>prod</i>, <i>avail_qty</i>, <i>loc</i>) Order(<i>ordid</i> PRIMARY KEY,      <i>custid</i> NOT NULL,      <i>invid</i> NOT NULL,      <i>shipid</i>, <i>qty</i>, <i>ord_stat</i>,      FOREIGN KEY(<i>custid</i>)      REFERNECES <i>Customer</i>      FOREIGN KEY(<i>invid</i>)      REFERNECES <i>Inventory</i>      FOREIGN KEY(<i>shipid</i>)      REFERNECES <i>Ship</i>) </pre>
---	---

**Fig. 1.** Key artifacts in EzMart

This paper makes the following technical contributions.

1. We formulate a new technical problem of preserving integrity constraints by modifying workflow specifications, develop an algorithm for solving this problem, and prove the correctness of the algorithm.
2. We introduce the concept of “conservative runs” and show that our solution is also “conservative complete”, i.e., injections are always weakest possible.
3. In carrying out this work, we also define a formal transition-system semantics for GSM (whose alternative semantics were developed recently [9, 18]).

We note here that although IPM is based on GSM, the methodology and techniques developed in this paper can be easily applied to other workflow specification languages supporting logical data models. In particular, IPM works as long as the action effect can be formulated as a transition system (and the workflow execution is guarded).

The remainder of the paper is organized as follows. Section 2 motivates the problem and illustrates GSM with an example. Section 3 sketches a formal semantics for GSM. Sections 4 and 5 are devoted to the injection algorithm and correctness proof, resp., with the concepts of soundness and conservative completeness included in Section 5. Section 6 reports on related work, and Section 7 concludes the paper. Due to space limitation, we omit detailed formalisms and technical proofs in the paper, and include them in an online appendix [23].

## 2 A Motivating Example and GSM

In this section, we illustrate the main problem with an example workflow. The example is specified in the declarative artifact-centric workflow model GSM [17], which provides the technical setting for this paper.

### 2.1 The EzMart Workflow

In an online shopping center “EzMart”, a registered customer can buy products and the purchased items are delivered to the customer’s address. Modeled with an artifact-centric approach [6], EzMart contains four artifact classes: *Customer*, *Order*, *Ship*, and

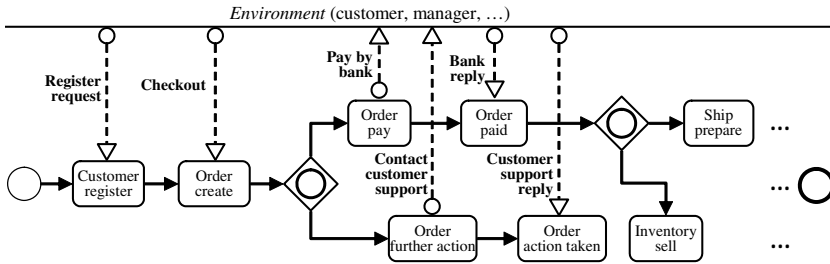


Fig. 2. The EzMart workflow in BPMN-like notation

*Inventory*. The artifacts are structured as relations as shown in Fig. 1, where *ord\_stat* can be one of “CREATE”, “INVUPD”, “CANCEL”, or “RETURN”, and *ship\_stat* can be one of “PREPAR”, “SHIPIN”, “FINISH”, or “FAILED”.

Fig. 2 shows a part of EzMart which is a typical online store process. The customer first registers, can then select products and proceed to checkout. An order is created when the checkout request is made and the customer pays the order using an online bank service. When the order is paid, a shipment process starts and it completes when the package is delivered to the customer. After an order is made, the customer may contact the customer support to take further action(s) on the order, and the order may be returned or canceled (and also possibly changed to other status), the order status is updated accordingly. The back-end inventory management will calculate the available quantity as an order is paid. When the quantity is too low, the inventory manager is notified and a replenishment process starts that will eventually update the quantity (not showed in Fig. 2).

## 2.2 GSM Specification of EzMart

We now specify key components of EzMart using the workflow language GSM [17]. GSM models complex business process with globalization and out-sourcing in a declarative fashion. The behavior and constraints of business operations are specified in event-condition-action rules.

There are two key constructs in GSM: the information model and the lifecycle model. The former consists of the artifacts and their data attributes (as described in Subsection 2.1). The latter is specified using “stages” consisting of guards, milestones, and stage bodies. Intuitively, a *stage* represents a phase of processing of an artifact. A stage is entered if its *guard* is true, and ends when a *milestone* is accomplished (a condition becomes true). Fig. 3 shows a specification of EzMart in GSM that extends the BPMN workflow shown in Fig. 2. In Fig. 3, a stage (body) is shown as a rectangular with round-corners, a diamond on a stage is the guard (diamond with a “+” in the middle represents the corresponding stage will create a new artifact instance), and a circle on a stage is the milestone (a circle with a bullet indicates a *finish* milestone, a milestone that can complete a lifecycle).

<sup>1</sup> In Fig. 2 we use the inclusive gateway of in BPMN, denoted by a diamond with a cycle in the middle. Such a gateway allows one or more branches following to be taken.

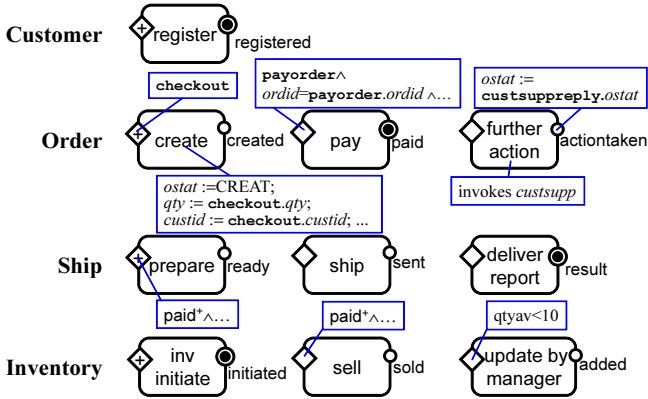


Fig. 3. GSM lifecycle model of EzMart

We focus on artifact class *Order* to illustrate how the GSM model of EzMart works. When the customer is ready to checkout, a *checkout* event is sent to EzMart. An *Order* artifact is then created by stage *create*, where *ord\_stat* is set to “CREATE”, *custid* and *ordid* (reference to associated customer and inventory, resp.) and the *qty* are set according to the content of the *triggering* event *checkout*.

To pay the order, the customer sends a *payorder* event with *ordid* matching the artifact ID. The stage *pay* then opens, and bank service is invoked to pay the order. When the bank replies, the finish milestone *paid* is achieved.

After the order is made, the customer can request some further actions on the order. If the customer does so, stage *further\_action* opens and customer support human task *custsupp* is invoked. The milestone *actiontaken* is achieved by the reply event of customer support. This reply takes the immediate effect on milestone *actiontaken* to change the status of the order, e.g. mark the status of the order as canceled.

When the order is paid, the *control event*  $paid^+$  (achieving the milestone *paid*) can trigger stages *prepare* of *Ship* and *sell* of *Inventory*. Furthermore, in stage *sell*, the *ord\_stat* of the *Order* artifact identified by the ID retrieved from the event  $paid^+$  is assigned to “INVUPD” (i.e.  $Order(paid^+.ordid).ostat := INVUPD$ ).

Stage guard conditions and milestones are formulas on events and attributes (stage and milestone status is not used). In the remainder of the paper, the term “sentry” is used to refer to conditions for both guard and milestones.

### 2.3 Integrity Constraints

Artifacts are stored in a database (conceptually). Data in workflow are a representation of the reality and thus integrity constraints arise naturally [6]. Some integrity constraints in EzMart (e.g. not-null, keys and foreign keys) are already shown in the artifact relation definitions in Fig. 1. There are additional integrity constraints. The constraints on *attribute content* restrict the domains of attribute values. For each *Order* artifact, the quantity is greater than 0, i.e.  $qty > 0$ ; for *Ship* artifacts, sending address must be

different from the delivery address, i.e.  $from \neq addr$ ; for *Inventory* artifacts, the available quantity is non-negative,  $avail\_qty \geq 0$ .

In addition to the constraints on attributes of a single artifact, there are further *business specific constraints*:

- **Status constraint:** Given an artifact  $s$  of *Ship*, if the shipment has started but not yet finished, the associated order cannot be canceled nor returned. That is, if  $s.ship\_stat$  is neither “FINISH” nor “FAILED”, and there is an artifact  $o$  of *Order*, such that  $s.ordid = o.ordid$  and  $o.shipid = s.shipid$ , then the order status  $o.ord\_stat$  must not be “RETURN” nor “CANCEL”.
- **Address-name constraint:** Given an artifact of *Order*, the delivery address  $addr$  and recipient’s name  $name$  of the associated *Ship* artifact must match the address  $addr$  and  $name$  of the associated *Customer* artifact.
- **Ship-from constraint:** Given an artifact of *Order*, the sending address of the shipment must match the inventory warehouse location  $loc$ .
- **Ship-order reference circle:** Given an artifact of *Order*, the *Ship* artifact referenced by attribute  $o.shipid$  must also reference back to  $o$ , and *vice versa*.

## 2.4 Enforcing Integrity Constraints: A Challenge

Traditionally workflow systems rely on the underlying database system to ensure data consistency. However, in reality the data are quite likely stored and managed distributedly. Artifacts in a single artifact class may be stored in several databases. Assume that EzMart combines two old shopping centers that maintain their own customer databases. Some integrity constraints in EzMart, e.g. the candidate key on *Customer*, cannot be handled properly and easily by one database system alone [14].

The recent trend of cloud computing and SOA brings other opportunities: (1) data management of EzMart can be outsourced and the service provider may “pack” similar data from EzMart and other applications together, (2) the customer data EzMart uses may be owned by a separate data service provider who may not respect data integrity constraints from EzMart. Consider the repository for *Order* of EzMart that shares the same actual data with electronic order database of other companies. The data service provider has to keep the data of EzMart from the other companies as well as maintaining constraints from different applications. This results in high complexity and expenses. In general, elevating integrity constraints local to one workflow to global for the data service provider is problematic. For example, other applications may not require quantity in the order to be strictly positive (cf. attribute content constraint of EzMart).

It is desirable for a workflow to block its own updates if they violate integrity constraints. Consider the attribute content constraint on *Order* that requires for each *Order* artifact  $o$ ,  $o.qty > 0$ . In Fig. 4, the stage *create* uses the triggering event *checkout* to assign  $qty$ . Then if we strengthen the sentry of the guard to allow only the event with  $checkout.qty > 0$  to pass, the constraint on *Order* cannot be violated by the update from the stage. To generalize this idea, associated data integrity constraints should be preserved *within* EzMart by strengthening the guard condition—the *guard injection*.

As an extreme, the simplest and effective injection is to inject FALSE to the guard of every stage. Then no execution would violate the constraints—because there will be

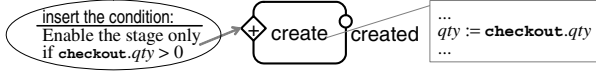


Fig. 4. Prevent the violation by strengthen the sentry condition

no execution at all. To make the injection useful, we need to make sure the injected constraints must be weaker or even the “weakest”. The injection should block all executions that can violate the constraints but should also allow as many executions as possible that preserve data consistency.

This paper develops a technical approach that calculates injection according to the constraint set and actions in stage bodies. The approach is proved to be both “sound” (strong enough to prevent violations) and “conservative complete” (weak enough to have a useful workflow).

### 3 A Formal Semantics of GSM

In order to analyze GSM workflow for possible injection, it is necessary to formalize its semantics. In this section, we define the execution model (an operational semantics) of GSM specifications. First we give an intuitive explanation of GSM execution. Then we present a transition system semantics for GSM. The formalism is inspired (but not restricted) by the Z notation [25]. Our transition system semantics is complementary to recent GSM operational semantics presented in [9, 18]. Our semantics focuses on manipulation of data and system variables, and forbids the concurrency between atomic stages. A brief comparison can be found in the online appendix [23].

#### 3.1 Intuitive Explanation

The GSM execution model was initially described in [17], and further developed in [18]. A workflow starts with no artifacts. An artifact is created when a create-instance stage opens. A stage opens if its sentry is satisfied and then actions defined in the stage body starts to execute. For example, stage `update_by_manager` in EzMart opens when attribute `avail_qty` of the `Inventory` artifact is less than 10. If the stage is to create instance, a new artifact is created, e.g. create stage creates a new `Order` artifact.

A milestone is achieved when its achieving sentry is satisfied and its belonged stage is open. For example, action `taken` is achieved when stage `further_action` is opened and the reply event from task `custsupp` comes (i.e. the head event in the external event queue is reply from `custsupp`). A milestone is invalidated when its invalidating sentry is satisfied, i.e. the milestone changes to or stays in the status of not achieved.

A stage can also reopen, when its sentry is satisfied again. For example, stage `further_action` can run several times as long as the event `takeaction` is received. Our workflow model extends the GSM slightly with the notion of “finish” milestones, such as `registered`, `paid`, etc. And only when all of the finish milestones are achieved can we say a workflow execution finishes (formal and strict definition in Section 5).

When the status of a stage or a milestone changes, a control event is generated. The control event can also be used as a triggering event. An example is the triggering event



of sell. The stage needs a control event  $\text{paid}^+$  to open, and  $\text{paid}^+$  denotes the paid milestone's achieving.

### 3.2 GSM Transition Systems

In our execution model, a GSM workflow is a transition system consisting of a state space (a set of states), an initial state and a set of operations (or transitions). The state space is a set of all possible “snapshots” of artifacts, each is called a *state* which specifies, at a specific time during the execution, the attribute value, the status of stage and milestones and event queues. The *initial state* is a special state that the transition system starts on, where no artifacts exist in the transition system, and event queues are empty.

*Operations* are transitions from one state to the next, identified by “operation signatures”, and the transition enabling condition (called the “guard”) and state changes made by the transition (called the “actions”) are specified. There are the following six types of operations:

- *Open* : opens a stage, if the stage sentry is satisfied and no other stage is open. When the stage opens, all of its milestones are set to be not achieved. New instance is created if the stage is a create-instance stage.
- *Body* : executes the stage body.
- *AchieveClose* : achieves a milestone. The operation is enabled when the achieving sentry of the milestones is satisfied, and the stage containing the milestone is opened. When the milestone is achieved, its stage is closed. And if the milestone is triggered by some external events, immediate effect is taken on the attributes.
- *Invalid* : invalidates a milestone (changes the milestone status to not achieved).
- *DeCQ* : removes the head event from control event queue. This operation has a lower priority than the above four types of operations.
- *DeEQ* : removes the head event from external even queue. This operation has the lowest priority. That is, only when none of the above five types of operations can be enabled, can *DeEQ* be enabled.

Status change of stage opening and closing, milestone achieving and invalidating generates control events, which are also control events to be added to the event queue (used to trigger other stages or milestones in the workflow). Parameters of the signature for each *Open* operation are the stage name and (if the stage is a create-instance stage) the artifact ID, for *Body* are the stage name and artifact ID, for *AchieveClose* and *Invalid* are the milestone name and artifact ID, and no parameter for *DeCQ* and *DeEQ*.

Due to space limitation, only the state space and *Open* operation (for a create-instance stage) are presented, an example is given to briefly explain the operations *AchieveClose*. A complete formalism of state space and operations is included in [23].

Given a GSM workflow  $AP$ , its transition system is denoted by  $TS_{AP}$ . The state space is specified by the construct  $STATE$  in Z notation, shown in the left column of Fig. 5.

In the state space each artifact class  $\alpha_i$  is represented as a table consisting the artifact ID, the data attribute value (denoted by  $x, y, \dots$ ), and stage and milestone status (denoted by stage or milestone names, such as  $s, m, \dots$ ). And the set of all artifact classes in  $AP$  is  $\mathbf{A} = \{\alpha_i \mid i \in 1..n\}$ . For any artifact class  $\alpha$ , we use  $S(\alpha)$  and  $M(\alpha)$  to denote the sets of all stages and milestones of  $\alpha$ , resp.

The state variable  $XOp$  is a finite set of signatures of possible next operations, where  $OPSIG$  is the type of operation signatures. Variable  $eq$  and  $cq$  are two queues of external events and control events, resp., where  $ExtEv$  and  $IntEv$  are respectively the type of external events and control (internal) events. State variables that do not immediately help in understanding the execution model are omitted.

<u>STATE</u>	<u>Open(create)</u>
$\alpha_1(id, x, y, \dots) : ArtifactClass$	$new = Order(newid(), null, \dots,$
$\alpha_2(\dots, s, m, \dots) : ArtifactClass$	$TRUE /*create*/, FALSE, FALSE, \dots)$
⋮	<hr/>
$\alpha_n(\dots) : ArtifactClass$	$Open(create) \in XOp$
$XOp : \mathbb{F} OPSIG$	$\forall \beta : \mathbf{A}; id : ID; t : S(\beta) . \beta(id).t = FALSE$
$eq : seq ExtEv$	$(head eq) \text{ isevent checkout}$
$cq : seq IntEv$	⋮
⋮	<hr/>
⋮	$Order := Order \cup \{new\}$
	$XOp := (XOp - \{Open(create)\}) \cup$
	$\{Body(create, new.id)\}$

**Fig. 5.** State space and an example operation

We give an example state  $s$  of  $TS_{EzMart}$ :

- $s.A$  is the set of tables of artifacts of classes *Customer*, *Order*, *Ship* and *Inventory*.
- $s.XOp = \{Open(create)\}$ , the set of the open stage operation of *create* of *Order*.
- $s.eq = \langle checkout(\dots), \dots \rangle$  and  $s.cq = \langle \rangle$  (empty queue).

On the initial state, all artifact class tables are empty, next operation set ( $XOp$ ) is the set of *Open* operations for create-instance stages, external and control event queues ( $eq$  and  $cq$ ) are both empty. As for EzMart,  $init.XOp$  is the set of *Open(register)*, *Open(create)*, *Open(prepare)* and *Open(inv\_initiate)*.

Operations are defined using the schema extended from Z notation [25].<sup>2</sup> Operation *Open* is responsible to handle stage opening. We use *Open(create)* of *Order*, specified in the right column of Fig. 5 as an example of the operation to open a create-instance stage. Let  $s$  be the current state as the example state given above, and  $s'$  be the next state after the transition specified by the operation *Open(create)*. Suppose  $s.Order$  has one row: *Order(ord001, cust002, ...)*. If there is no stage being open on  $s$ , then  $s$  satisfies the guard of *Open(create)*. A new *Order* artifact, denoted by the local variable  $new$ , is created, where the ID is assigned using a system new ID generator, all other attributes are assigned to null and statuses of all stages and milestones are set to FALSE except the status of *create* is set to TRUE. Let's assume the new ID generator gives ord002 on  $s$ . Then, after this operation,  $s'.Order$  has two rows,

$Order(ord001, cust002, \dots); Order(ord002, null, \dots, TRUE, FALSE, FALSE, \dots)$

and  $s'.XOp = \{Body(create, ord002)\}$ . Since the open stage event of *create* is not used in EzMart, control event queue  $cq$  is kept unchanged.

<sup>2</sup> Readers who are familiar with Z can find out our notation still follows the fundamental idea of Z, and the extension is only “syntactical” to make our specification easier.

We group achieving of a milestone and closing of its stage in *AchieveClose*. Consider the operation of achieving milestone *actiontaken*. The signature of the operation is *AchieveClose(actiontaken, ordid)*. Suppose that on the current state  $s$ ,  $s.XOp$  contains *AchieveClose(actiontaken, ord002)*,  $s.Order(002).pay = \text{TRUE}$ ,  $s.Order(002).paid = \text{FALSE}$ , and  $s.eq = \langle \text{custsuppreply}(\dots) \rangle$ , where event *custsuppreply* is the reply from customer support task *custsupp*. Then this operation can be enabled with *ordid* taking the value of *ord002*. As a result, on the next state, artifact  $s'.Order(\text{ord002})$  is updated as  $\text{pay} = \text{FALSE}$ ,  $\text{paid} = \text{TRUE}$ , and *ostat* is set according to *custsuppreply.ostat* — the immediate effect of the event.

## 4 Guard Injection

Based on the GSM execution model, we explore in this section our approach to enforce integrity constraints. The key idea is to inject conditions according to the specified constraints and the stage to “block” possible violations. We first define constraints and some needed notions, and then present the algorithm for guard injection.

Our problem is similar to but different from checking integrity constraints in distributed databases where the exact changes to the database are known [14, 19]. Our problem considers workflow specifications when the updates to the database are unknown but parameterized. A key idea of [14, 19] is to “look forward” at the “post-condition” of the update and check locally if the constraint with respect to the post-condition can be satisfied without looking at database(s). (If not, databases are consulted to check the integrity constraints.) The injection technique developed in this paper “looks backward” to calculate “weakest” precondition of stage and ensures that *potentially* executed updates would never violate the constraints.

### 4.1 Integrity Constraints

In this paper, each integrity constraint  $\kappa$  is defined in the following form (cf [2]):

$$\kappa = \forall \mathbf{x} . (\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} . \psi(\mathbf{x}, \mathbf{y})) \quad (1)$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are finite vectors of variables with no repetition,  $\mathbf{x}$  is nonempty (while  $\mathbf{y}$  can be empty), and Formula  $\phi$  and  $\psi$  are nonempty conjunction of artifact relation atoms and comparison atoms of the form  $x \circ y$ , where  $x$  is a variable and  $y$  is a variable or constant and  $\circ$  denotes operators:  $=$ ,  $\neq$ ,  $\geq$ ,  $\leq$ ,  $>$  and  $<$ . Variables in  $\mathbf{x}$  and  $\mathbf{y}$  are artifact IDs or data attributes of artifacts in *AP*. There is *at least one* artifact relation atoms in  $\phi$ . Formula  $\phi$  uses all variables in  $\mathbf{x}$  and  $\psi$  uses all variables in  $\mathbf{y}$ .

The conjunctive atoms in  $\phi$  are *premises* and the right hand side of the arrow is referred to as the *consequent*. For simplicity, we assume all constraints are free from trivial atoms (equivalent to *TRUE* or *FALSE*). Variables in  $\mathbf{x}$  are referred to as  $\forall$ -quantified variables, while the ones in  $\mathbf{y}$  as  $\exists$ -quantified variables.

For example, the attribute content and not-null constraint on *Order* and status constraint are given in Section 2 and we repeat here:

**Attribute content and not-null constraint on Order,  $\kappa_{attr}$ :** For each *Order* artifact, neither of the references *custid* and *invid* is null, and the quantity is larger than 0, i.e.  $qty > 0$ ;

**Status constraint,  $\kappa_{stat}$ :** Given an artifact *s* of *Ship*, if *s.ship\_stat* is neither FINISH nor FAILED, and there is an artifact *o* of *Order*, s.t. *s.ordid* = *o.ordid* and *o.shipid* = *s.shipid*, then the order status *o.ord\_stat* must not be RETURN or CANCEL.

Written in the form of Equation (II), these two constraint formulas are:

$$\begin{aligned} \kappa_{attr} &= \forall ordid, custid, invid, shipid, qty, ord\_stat. \\ &\quad Order(ordid, custid, invid, shipid, qty, ord\_stat) \rightarrow \\ &\quad\quad\quad custid \neq null \wedge invid \neq null \wedge qty > 0 \\ \kappa_{stat} &= \forall ordid, custid, invid, shipid, qty, ord\_stat, addr, name, from, ship\_stat. \\ &\quad Order(ordid, custid, invid, shipid, qty, ord\_stat) \wedge \\ &\quad Ship(shipid, ordid, addr, name, from, ship\_stat) \wedge \\ &\quad ship\_stat \neq FINISH \wedge ship\_stat \neq FAILED \rightarrow \\ &\quad\quad\quad ord\_stat \neq RETURN \wedge ord\_stat \neq CANCEL \end{aligned}$$

The complete list of formulas for all constraints in EzMart in the form of Equation (II) can be found in [23].

We say a *concerning attribute* of constraint  $\kappa$  is some attribute  $\alpha.x$  of some artifact class  $\alpha$ , where  $\alpha.aid$  is in  $\forall$ -quantified variables of  $\kappa$ ,  $\alpha$  is an artifact relation atom in  $\kappa$ , and any one of the following holds:

- there is a constant appearing at the column of  $x$  in artifact relation atom of  $\alpha$  in  $\kappa$ ;
- there is a variable  $x'$  appearing at the column of  $\alpha.x$  in artifact relation atom of  $\alpha$ , and  $x' \circ y$  also appears in  $\kappa$ , where  $y$  is a constant or any other variables; or
- there is some variable  $x'$  appearing more than once in  $\kappa$  and one of its appearances is at the column of  $\alpha.x$  in artifact relation atom of  $\alpha$ .

The set of concerning attributes of  $\kappa$  is denoted by  $CA(\kappa)$ .

The sets of concerning attributes of  $\kappa_{attr}$  and  $\kappa_{stat}$  are

$$\begin{aligned} CA(\kappa_{attr}) &= \{Order.custid, Order.invid, Order.qty\} \\ CA(\kappa_{stat}) &= \{Ship.ordid, Ship.ship\_stat, Order.shipid, Order.ord\_stat\}. \end{aligned}$$

Given a constraint  $\kappa$  and an attribute  $x \in CA(\kappa)$ , the set of *writing stages* of  $x$  is denoted by  $WS(x)$ . A writing stage of  $x$  is such a stage  $\mathfrak{s}$  that  $x \in WriteSet(\mathfrak{s})$ , where  $WriteSet(\mathfrak{s})$  denotes the set of attributes that can be written by the body of stage  $\mathfrak{s}$ . The set of *writing milestones* of  $x$  is denoted by  $WM(x)$ . A writing milestone of  $x$  is such a milestone  $\mathfrak{m}$  that uses the reply event to update attribute  $x$ .

In EzMart, the writing stage of concerning attributes of  $\kappa_{attr}$  is create; and the writing stages and the writing milestone of concerning attributes of  $\kappa_{stat}$  are

$$\begin{array}{ll} Ship.ordid & : \text{prepare} & Ship.ship\_stat & : \text{prepare, ship} \\ Order.shipid & : \text{ship} & Order.ord\_stat & : \text{create, sell, actiontaken} \end{array}$$

## 4.2 Calculating Injected Conditions

The algorithm to calculate the injection is now presented. The intuition of the injection is first described. Then the algorithm is given along with examples using attribute content and not-null constraint on *Order* ( $\kappa_{attr}$ ) and status constraints ( $\kappa_{stat}$ ).

The intuition of the algorithm is to “inject” properly converted constraints into guards of *Open* operations of writing stages and stages of writing milestones of concerning attributes of the constraints. As a result, the guard is strengthened to block all updates that may violate the integrity constraints, but to allow updates that preserve the data integrity.

For writing stages, we analyze the stage body to understand how the updates are made. For example, stage *create* uses the triggering external event checkout to set *custid*, *invid* and *qty* — the concerning attribute of  $\kappa_{attr}$ . The injection is a substitution for the concerning attribute according to the stage update. The injection is replacing concerning variable in  $\kappa_{attr}$  by corresponding content of event checkout. Assume there is no violation before the update made by *create*, if the current state satisfies the injection, the execution of *create* under such a checkout event preserves the data consistency regarding to  $\kappa_{attr}$ . Moreover, such an injection is also weak enough only to block the updates that result in violation. As for the writing milestones, because there is no information about the task reply in the workflow, the injection to its stage’s *Open* operation has to be made to the strongest—FALSE.

In the following we present the detail of the algorithm. To begin with, implicit foreign key references are made explicit in the constraint set. A stage, e.g. *ship*, of one artifact class  $\alpha$ , e.g. *Ship*, can write attributes of artifact class  $\beta$ , e.g. *Order.shipid*. Then the foreign key constraint is added to the integrity constraint set, if it is not already specified:

$$\forall aid, bid, \mathbf{x}. \alpha(aid, bid, \mathbf{x}) \wedge bid \neq \text{null} \rightarrow \exists \mathbf{y}. \beta(bid, \mathbf{y})$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are disjoint vectors of other “unrelated” attributes in  $\alpha$  and  $\beta$ , resp. In EzMart, all of the foreign key dependencies are already specified.

---

### Algorithm 1. Guard injection

---

**Input:**  $TS_{AP}$  and  $\mathbf{K}$

**Output:**  $Inj : \mathbf{S} \rightarrow FO$

Set  $Inj(\mathbf{s}) := \text{TRUE}$  for each stage  $\mathbf{s}$  in  $AP$ ;

**foreach**  $\kappa \in \mathbf{K}$  and  $x \in CA(\kappa)$  **do**

**foreach**  $\mathbf{s} \in WS(x)$  **do**

    1      $Inj(\mathbf{s}) := Inj(\mathbf{s}) \wedge \text{SUB}(\kappa, \mathbf{s} [, aid])$ , where  
         $aid$  is the artifact ID in  $Open(\mathbf{s} [, aid])$  in  $TS_{AP}$ ;

**endfch**

**foreach**  $m \in WM(x)$  **do**

$Inj(\mathbf{s}) := \text{FALSE}$

**endfch**

**endfch**

---

The procedure of injection is given in Algorithm 1. It takes the transition system  $TS_{AP}$  of GSM specification  $AP$  and the set of integrity constraint  $\mathbf{K}$  as input. The output is the injection function  $Inj$  which maps each stage in  $AP$  to a first order formula. The idea of the algorithm is simple. For each constraint  $\kappa$  in  $\mathbf{K}$  and for each concerning variable  $x$  of  $\kappa$ , if  $x$  is going to be written by a stage  $\mathbf{s}$ , the algorithm injects the converted  $\kappa$  by

replacing  $x$  with the assignment in the body of  $\mathbf{s}$ , where the substitution is accomplished by function SUB (given below); if  $x$  is going to be written by a reply event triggering a milestone, then FALSE is injected.

The function SUB used in Line 1 of Algorithm 1 is to convert the constraint formula  $\kappa$  according to the stage  $\mathbf{s}$  and (if  $\mathbf{s}$  is not a create-instance stage) the artifact ID  $aid$ . Concerning attributes and artifact relations are replaced according to the assignments in the body of  $\mathbf{s}$ .

First, in Line 2 of SUB, by function  $explicitref(\kappa)$ , reference dependency premises are added when the stage writes attributes of another artifact class. The substitution procedure then starts by taking care of each  $\forall$ -quantified  $\alpha$  IDs. The variable  $x$  is replaced by the assignment in the stage body (Line 3), and the ID is replaced (Line 4), by  $newid()$  if the stage creates new instance; otherwise, by the artifact ID of the updated artifact in the stage assignment (denoted by  $lhsid$ , which is ID field of head event if the artifact is identified by the event or  $aid$  otherwise). After the variables are properly replaced, they are removed from the  $\forall$ -quantified variable list. Note that when the stage is creating an instance, replacing the  $id$  with  $newid()$  also replace the relation atom to TRUE because the new instance is going to be inserted and therefore the relation atom holds. Note that after the substitution,  $\beta(\dots, newid(), \dots)$  is further replaced by FALSE for any artifact relation  $\beta$ , and  $id \circ newid()$  is further replaced by FALSE for any  $id$  unless  $\circ$  is  $\neq$ . More technical details of SUB can be found in the online appendix [23].

---

**Function.** SUB( $\kappa, \mathbf{S}[, aid]$ ) returns the constraint formula after substitution

---

```

 $ID^\kappa :=$  all  $\forall$ -quantified artifact IDs of the artifact whose attribute is updated in  $\mathbf{s}$ ;
2  $con := explicitref(\kappa)$ ;
   $ret := TRUE$ ;
  foreach  $id$  in  $ID^\kappa$  do
3    $con := con[\mathbf{exp}/\beta(id).\mathbf{x}]$ , where  $\mathbf{x}$  and  $\mathbf{exp}$  are vectors of the same length
     and for any  $0 \leq i < \#\mathbf{x}$ ,  $\mathbf{s}.Body$  contains  $\beta(id).\mathbf{x}_i := \mathbf{exp}_i$ ;
4   if  $\mathbf{s}.creatInst = TRUE$  then
      $con := con[TRUE/\alpha(id, \dots)]$ , where  $\mathbf{s} \in S(\alpha)$ ;
      $con := con[newid()/id]$ 
   else
      $con := con[lhsid/id]$ ;
   endif
   $ret := ret \wedge con$ 
endfch
return  $ret$ 

```

---

In EzMart, for the attribute content and not-null constraint on  $Order$  ( $\kappa_{attr}$  in Subsection 4.1), all of the three concerning variables are replaced by the corresponding content of event checkout in stage create. And since the stage creates a new instance, the  $Order$  artifact relation atom is replaced by TRUE. This gives the substitution result, after trivial reduction,

$$SUB(\kappa_{attr}, create) = TRUE \rightarrow (head\ eq).custid \neq null \wedge (head\ eq).invid \neq null \wedge (head\ eq).qty > 0$$

Then for the status constraint ( $\kappa_{stat}$  in Subsection 4.1), the stage *sell* of *Inventory* sets *Order*((*head cq*).*ordid*).*ord\_stat* to “INVUPD”, where by the stage sentry, *head cq* is a *paid*<sup>+</sup> event. Therefore we have,

$$\begin{aligned} \text{SUB}(\kappa_{stat}, \text{sell}, \text{invid}) = & \forall \text{custid}, \text{shipid}, \text{qty}, \text{ord\_stat}, \text{addr}, \text{name}, \text{from}, \text{ship\_stat}. \\ & \text{Order}((\text{head } cq).\text{ordid}, \text{custid}, \text{invid}, \text{shipid}, \text{qty}, \text{INVUPD}) \wedge \\ & \text{Ship}(\text{shipid}, (\text{head } cq).\text{ordid}, \text{addr}, \text{name}, \text{from}, \text{ship\_stat}) \wedge \\ & \text{ship\_stat} \neq \text{FINISH} \wedge \text{ship\_stat} \neq \text{FAILED} \rightarrow \\ & \text{INVUPD} \neq \text{RETURN} \wedge \text{INVUPD} \neq \text{CANCEL} \end{aligned}$$

where *ordid* is replaced by (*head cq*).*ordid*, and *ord\_stat* is replaced by the constant INVUPD. And obviously, this is equivalent to TRUE.

After Algorithm 1 completes, injection to stages of *Order* is given as follows. Note that trivial expressions are directly removed and the injected formula is reduced.

$$\begin{aligned} \text{Inj}(\text{create}) = & \text{SUB}(\kappa_{attr}, \text{create}) \wedge \\ & \exists \text{email}, \text{addr}, \text{name}, \text{info}, \text{prod}, \text{avail\_qty}, \text{loc}, \text{price}. \\ & \text{Customer}((\text{head } eq).\text{custid}, \text{email}, \text{addr}, \text{name}, \text{info}) \wedge \\ & \text{Inventory}((\text{head } eq).\text{invid}, \text{prod}, \text{avail\_qty}, \text{loc}, \text{price}) \\ \text{Inj}(\text{further\_action}) = & \text{FALSE} \end{aligned}$$

The second conjunct injection on stage *create* is the one for foreign key constraint (to *Customer* and *Inventory*); injections for ship-order reference circle, address-name, ship-from and status constraint are all reduced to TRUE. Injection on *further\_action* is FALSE because *actiontaken* milestone of this stage uses the reply event to update *ord\_stat*, a concerning attribute of the status constraint  $\kappa_{stat}$ .

The complete list of constraints and injections on EzMart can be found in [23].

## 5 Soundness and Conservative Completeness

To state the correctness of the injection algorithm, we first define some technical notions. Examples from EzMart are given along the definitions and the technical results of “soundness” and “conservative completeness”.

**Definition 1 (run and complete run).** Let *AP* be a GSM specification, *TS<sub>AP</sub>* its transition system. A *run* of *TS<sub>AP</sub>* is an alternating sequence of states and operations

$$\rho = s_0 t_0 s_1 t_1 \dots t_{n-1} s_n$$

where  $s_0, s_1, \dots$  are states (specified by *STATE*),  $s_0$  is the initial state, and  $t_0, t_1, \dots$  are operations (specified by *OPER*), such that for each  $t_i$  ( $i \geq 0$ ),  $s_i \models \text{guard}(t_i)$ .

Let **A** be the set of artifact classes in *AP*. A run  $\rho$  is said to be *finished* iff state  $s_n$  satisfies that for each artifact of  $\alpha : \mathbf{A}$  with *id* : *ID*,

- there is a finish milestone achieved, i.e.  $\exists \mathbf{m} : M(\alpha) . \mathbf{m}$  is a finish milestone  $\wedge \alpha(\text{id}).\mathbf{m} = \text{TRUE}$ ; and
- there is *no* stage being open, i.e.  $\forall \mathbf{s} : S(\alpha) . \alpha(\text{id}).\mathbf{s} = \text{FALSE}$ .

In EzMart, each run starts with the initial state and followed by an open stage operation of register, create, prepare or inv\_initiate. On state  $s_1$ , only the stage body operation of the just opened stage in  $t_0$  can be enabled. State  $s_2$  is the result of the stage body operation. The run is finished if on the last state in the run, milestones registered of all artifacts of *Customer*, paid of all artifacts of *Order*, result of all artifacts of *Ship* and the initiated of all artifacts of *Inventory*, are all achieved, and there is no stage being open.

Given a run, if it does not violates any constraint, we say this run is sound.

**Definition 2 (Sound run).** Let  $AP$  be a GSM specification,  $TS_{AP}$  be its transition system and  $\mathbf{K}$  a set of integrity constraints on artifacts of  $AP$ . A run  $\rho$  of  $TS_{AP}$  is said to be  $\mathbf{K}$ -sound iff for each  $\kappa \in \mathbf{K}$ ,  $\kappa$  holds in every state in  $\rho$ . When  $\mathbf{K}$  is clear from the context, we simply say  $\rho$  is sound.

Consider run  $\rho_1 = s_0t_0s_1t_1s_2t_2s_3$  in EzMart where  $t_0$  is *Open(register)* and creates a new *Customer* artifact with ID cust001,  $t_1$  is *Body(register, cust001)* which sets *Customer(cust001).email* to abcdef.com, and  $t_2$  is *AchieveClose(registered, cust001)*. We can see that  $\rho_1$  is a finished run, and also sound. Because *email* is not empty, attribute content constraint on *Customer* is satisfied on all of the states in  $\rho_1$ . There is only one artifact in the system, the candidate key and foreign keys are also satisfied, and all business specific constraints are also satisfied.

Now consider another run  $\rho_2 = s_0t_0 \cdots t_i s_{i+1} \cdots$ . Suppose on  $s_i$ , there is an artifact *Ship(ship005).ship\_stat* = SHIPIN and  $t_i$  is *AchieveClose(action\_taken, ord002)*. If *Order(ord002).shipid* = ship005 and the immediate effect of reply event is to set the *Order(ord002).ord\_stat* to CANCEL, then status constraint ( $\kappa_{stat}$ , see Subsection 2.1) is violated, because the order ord002 is canceled when the purchased item is still shipping. Therefore  $\rho_2$  is not a sound run.

In business processes, many tasks are third-party services or human tasks. It is not reasonable to assume all tasks will strictly follow some contract. We have to be prepared that external tasks may give unpredictable reply in the domain. To ensure the constraints are never violated, we need to be cautious or conservative. Therefore, in the Algorithm 1, if the reply event may update a concerning attribute of a constraint, FALSE is injected to the associating stage.

**Definition 3 (Conservative run).** Let  $AP$ ,  $TS_{AP}$  and  $\mathbf{K}$  be the same as in Definition 2. A finished run  $\rho$  of  $TS_{AP}$  is said to be  $\mathbf{K}$ -conservative iff  $\rho$  is sound and for any constraint  $\kappa \in \mathbf{K}$ , there is no reply event being used to update any attribute in  $CA(\kappa)$ . When  $\mathbf{K}$  is clear from the context, we simply say  $\rho$  is conservative.

Consider a run  $\rho_3 = s_0t_0 \cdots s_k t_k s_{k+1} \cdots$ , and  $t_k$  is *Body(further\_action, ord002)*. The milestone actiontaken belongs to stage further\_action, and is a writing milestone of concerning attributes *ord\_stat* of status constraint. Therefore  $\rho_3$  is not conservative.

The transition system of  $AP$  with injection according to integrity constraint set  $\mathbf{K}$  is denoted by  $InjTS_{AP}(\mathbf{K})$ . When  $\mathbf{K}$  is clear from the context,  $InjTS_{AP}(\mathbf{K})$  is simply written as  $InjTS_{AP}$ . It is constructed by concatenating in conjunction  $Inj(s)$  with the original guard of *Open* operation of each stage  $s$ . (If  $Inj(s)$  is equivalent to TRUE, then the operation guard after injection is equivalent to the one before.) The definition of “correct” injection, is given by the notion of “soundness” and “conservative completeness” of



transition system with injection, where soundness captures no violation—the injection is strong enough, while conservative completeness allows the maximal behavior under conservative strategy—the injection is weak enough.

**Definition 4 (Sound and conservative complete injection).** Let  $AP$ ,  $TS_{AP}$  and  $\mathbf{K}$  be the same as in Definition 2 and  $Inj$  be the guard injection function. We say the injection  $Inj$  is *sound* iff each finished run of  $InjTS_{AP}$  is sound, *conservative complete* iff each conservative run of  $TS_{AP}$  is also a conservative run of  $InjTS_{AP}$ .

The main property of our algorithm is now stated, a proof can be found in the online appendix [23].

**Theorem 1.** Given a GSM specification  $AP$  and a set of integrity constraints  $\mathbf{K}$ , the transition system with injection,  $InjTS_{AP}$ , is both sound and conservative complete.

Again, we take advantage of EzMart to illustrate the idea of injection correctness. First, for any run of  $InjTS_{EzMart}$ , there is no violation. Take stage `create` as an example. If the head event of  $eq$  satisfies the sentry and injection of `create`, then, because  $Inj(\text{create})$  uses the assignment in `create`, after the update made in the body of `create`, the constraint is still satisfied. And because  $Inj(\text{further\_action}) = \text{FALSE}$ , this injection can block any possible updates made by the incoming reply event, and therefore ensures the integrity constraints.

Then, suppose there is a conservative run  $\rho$  of  $TS_{EzMart}$  that is not in  $InjTS_{EzMart}$ . If this is because the *Open* operation of `create` cannot be enabled in the injected workflow, say  $(\text{head } eq).qty = 0$ , then  $Inj(\text{create})$  fails; in  $\rho$  after the update made by `create`, the  $qty$  of newly created artifact of *Order* is 0 which violates the attribute content constraint on *Order*, and thus  $\rho$  cannot be sound. If it is because of the blocking in the injected system of *Open* operation of `further\_action`, then  $\rho$  is not conservative. Therefore, the injected workflow  $InjTS_{EzMart}$  is both sound and conservative complete.

## 6 Related Work

Triggers are a powerful tool to “fix” constraint violations as a reactive means, e.g. [7]. In distributed databases, checking constraints involving remote databases is expensive. It was discussed in [15] to maintain distributed integrity constraint efficiently by reducing the necessity to look at remote databases. It was investigated in [14] to use local data to test conjunctive query constraints with arithmetic comparison. In [19], a similar problem is discussed on conjunctive query constraints with negations. The approach used to generate complete local tests is basically to check constraint containment and calculate local tests with respect to “post-condition” of specific updates. To the contrary, our injection is to calculate the weakest precondition conservatively of potential updates and to ensure that updates never result in violation.

It is also studied that by enhancing the underlying systems, data consistency can be maintained in loosely coupled databases. A framework was developed that uses several communication protocols between different sites to maintain data consistency [13]. When strict consistency cannot be ensured, enforcing the weakened integrity constraints

is possible by a rule-based configurable toolkit presented in [8]. While these work construct strong data management systems, our work makes a minimum requirement on underlying DBMSs.

Using preconditions can also be found in [4] and [11]. The idea of finding weakest precondition rooted in [10]. The difference of using preconditions between our work and program verification is that the variable states and properties are on databases. In [4], the authors explored appropriate transaction languages to ensure integrity constraints using weakest preconditions. Calculation of weakest precondition was not discussed. In [11], authors studied the automated construction of artifact-centric workflows so that the workflow execution results are consistent with requirement defined on data, where weakest precondition of each task is calculated.

Rules were given in [20] to derive a set of functional dependencies that hold on query results on given relations. Decidability of dependency implication problem on Datalog programs was studied in [1]. Preservation of integrity constraints by a set of parameterized transactions was studied for relational databases [3] and for semantic databases [26].

## 7 Conclusion

This paper develops an approach to ensure data integrity within workflow execution by injecting converted constraints into guard of updates. The injection is proved to ensure data integrity while allowing all conservative runs of the original workflow.

The problem raised in this paper is hardly solved, there are many interesting issues to explore further. In one direction, it is desirable to extend this method for constraints with aggregations and arithmetic. Also, conservative requirement for injection can be relaxed by considering more accurate task models such as semantic web services and task contracts in the workflow. Another area of interest is to consider workflow with concurrent executions, which will require new injection techniques. Finally, it is interesting to investigate how techniques such as guard injection can be combined with mechanisms in federated databases [13].

**Acknowledgment.** The authors are grateful to Richard Hull (IBM T.J. Watson Research Center) for his informative discussions on the GSM semantics.

## References

1. Abiteboul, S., Hull, R.: Data functions, datalog and negation. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (1988)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
3. Abiteboul, S., Vianu, V.: A transaction-based approach to relational database specification. Journal of the ACM 36(4), 758–789 (1989)
4. Benedikt, M., Griffin, T., Libkin, L.: Verifiable properties of database transactions. In: Proc. ACM Symposium on Principles of Database Systems (PODS), pp. 117–127 (1996)
5. 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)

6. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling. Information Science Publishing (2008)
7. Ceri, S., Widom, J.: Deriving production rules for constraint maintainance. In: Proc. Int. Conf. on Very Large Data Bases (VLDB), pp. 566–577 (1990)
8. Chawathe, S., Garcia-Molina, H., Widom, J.: A toolkit for constraint management in heterogeneous information systems. In: Proc. Int. Conf. on Data Engineering (1996)
9. Damaggio, E., Hull, R., Vaculín, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 396–412. Springer, Heidelberg (2011)
10. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Communications of the ACM 18(8), 453–457 (1975)
11. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Proc. Int. Conf. on Database Theory, ICDT (2009)
12. Glushko, R.J., McGrath, T.: Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services. The MIT Press (2008)
13. Grefen, P., Widom, J.: Protocols for integrity constraint checking in federated databases. Distrib. Parallel Databases 5, 327–355 (1997)
14. Gupta, A., Sagiv, Y., Ullman, J.D., Widom, J.: Constraint checking with partial information. In: Proc. ACM Symp. on Principles of Database Systems (PODS), pp. 45–55 (1994)
15. Gupta, A., Widom, J.: Local verification of global integrity constraints in distributed databases. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 49–58 (1993)
16. Hollingsworth, D.: The workflow reference model: 10 years on. In: Workflow Handbook. Workflow Management Coalition, pp. 295–312 (2004)
17. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath III, F(T.), Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles (Invited talk). In: Bravetti, M. (ed.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011)
18. Hull, R., Damaggio, E., Masellis, R.D., Fournier, F., Gupta, M., Heath III, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: Proc. ACM Int. Conf. on Distributed Event-Based Systems, DEBS (2011)
19. Huyn, N.: Maintaining global integrity constraints in distributed databases. Constraints 2, 377–399 (1997)
20. Klug, A.: Calculating constraints on relational expression. ACM Trans. Database Syst. 5, 260–290 (1980)
21. Künzle, V., Weber, B., Reichert, M.: Object-aware business processes: Fundamental requirements and their support in existing approaches. Int. Journal of Information System Modeling and Design (IJISMD) 2(2), 19–46 (2011)
22. Liu, G., Liu, X., Qin, H., Su, J., Yan, Z., Zhang, L.: Automated realization of business workflow specification. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 96–108. Springer, Heidelberg (2010)
23. Liu, X., Su, J., Yang, J.: Preservation of Integrity Constraints by Workflow: Online Appendix, [http://seg.nju.edu.cn/~liux/pub/CoopIS11\\_appendix.pdf](http://seg.nju.edu.cn/~liux/pub/CoopIS11_appendix.pdf)
24. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
25. Spivey, J.M.: The Z Notation: A Reference Manual, 2nd edn. Prentice-Hall (1992)
26. Su, J.: Dependency preservation in semantic databases. Acta Inf. 31, 27–54 (1994)

# Monitoring Business Process Compliance Using Compliance Rule Graphs

Linh Thao Ly<sup>1</sup>, Stefanie Rinderle-Ma<sup>2</sup>, David Knuplesch<sup>1</sup>, and Peter Dadam<sup>1</sup>

<sup>1</sup> Institute of Databases and Information Systems, Ulm University, Germany

<sup>2</sup> Faculty of Computer Science, University of Vienna, Austria

{thao.ly,david.knuplesch,peter.dadam}@uni-ulm.de,  
stefanie.rinderle-ma@univie.ac.at

**Abstract.** Driven by recent trends, effective compliance control has become a crucial success factor for companies nowadays. In this context, compliance monitoring is considered an important building block to support business process compliance. Key to the practical application of a monitoring framework will be its ability to reveal and pinpoint violations of imposed compliance rules that occur during process execution. In this context, we propose a compliance monitoring framework that tackles three major challenges. As a compliance rule can become activated multiple times within a process execution, monitoring only its overall enforcement can be insufficient to assess and deal with compliance violations. Therefore, our approach enables to monitor each activation of a compliance rule individually. In case of violations, we are able to derive the particular root cause, which is helpful to apply specific remedy strategies. Even if a rule activation is not yet violated, the framework can provide assistance in proactively enforcing compliance by deriving measures to render the rule activation satisfied.

## 1 Introduction

In many application domains of information systems, business process compliance is increasingly gaining importance. In healthcare, for example, medical guidelines and clinical pathways should be followed during patient treatments. In the financial sector, regulatory packages such as SOX or BASEL III have been introduced to strengthen customers' confidence in bank processes. Finally, collections of quality controls, e.g., Six Sigma or ITIL, are of particular importance for the internal control of business processes. In this paper we assume an example scenario that is set in bank accounting, where a variety of rules and policies exists (e.g., as a result of risk management). A selection of such rules is listed below:

- $c_1$  Conducting a payment run creates a payment list containing multiple items that must be transferred to the bank. Then, the bank statement must be checked for payment of the corresponding items. In order to avoid fraud and errors, the payment list must be transferred to the bank only once.

- $c_2$  For payment runs with amount beyond €10,000, the payment list has to be signed before being transferred to the bank and has to be filed afterwards for later audits.
- $c_3$  When payment of an open item is confirmed in the bank statement, the item has to be marked as cleared eventually.
- $c_4$  Once marked as cleared, the item must not be put on any payment list.
- $c_5$  If an open item is not marked as cleared within 30 days, the bank details may be faulty. Thus, the bank details have to be (re)checked.

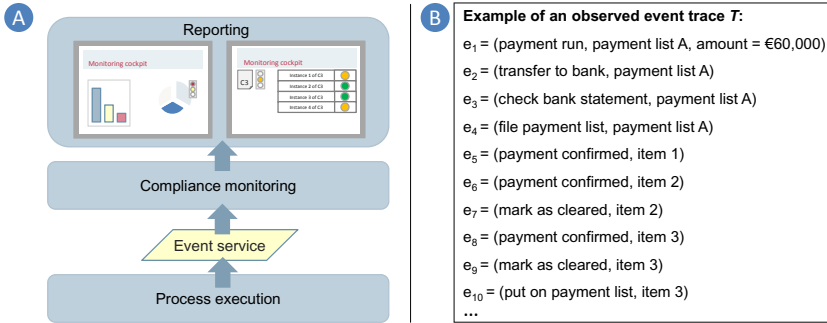
One way to ensure business process compliance is to verify the models of the affected business processes implemented within process-aware information systems against such imposed rules in order to achieve compliance by design. For that purpose, a multitude of approaches for business process model checking has been proposed in literature. However, design time checks are not quite sufficient as in many application domains, business process models are rarely documented or adhered to. To support business process compliance in such scenarios, it must be possible to monitor whether process executions (regardless whether an explicit process model exists) comply with imposed rules.

## 1.1 Challenges for Compliance Monitoring

Fig. 1A depicts a general compliance monitoring architecture. The events observed from process execution provide the basis for compliance monitoring. The actual low-level execution events may also be aggregated to meaningful events and then provided to the monitoring tier using event processing frameworks (e.g., complex event processing [1]). The monitoring tier, in turn, monitors compliance with imposed rules. It provides input to the reporting tier, where the results are visualized in accordance with the needs of the stakeholders involved. We assume that events such as the events from the bank accounting case listed in Fig. 1B can be provided by the event service. Clearly, the information that the monitoring tier is capable of providing constitutes the basis for presentation and visualization to stakeholders such as the process supervisor. This raises three fundamental challenges that we will discuss using the bank accounting case.

**Challenge 1: Identification and Monitoring of Individual Activations of a Compliance Rule.** Consider again compliance rule  $c_4$ . Then, a closer look at trace  $T$  from Fig. 1B reveals that  $c_4$  is violated over  $T$  as *item 2* and *item 3* are already marked as cleared but *item 3* is still put on a payment list again. In order to effectively deal with such violations, it is not sufficient for the monitoring tier just to identify that  $c_4$  is violated. Imagine that multiple other items are also marked as cleared within  $T$ . Then, it becomes very difficult to pinpoint the item(s) causing violations. Hence, the monitoring tier must provide fine-grained compliance feedback such that it becomes possible to pinpoint the violation.

In this example, two *activations* of  $c_4$  are present in the execution, namely for *item 2* and *item 3*. We refer to an event pattern that activates a compliance rule as a rule activation. For example,  $c_4$  becomes activated when an item is marked



**Fig. 1.** General compliance monitoring architecture and events from the bank accounting case

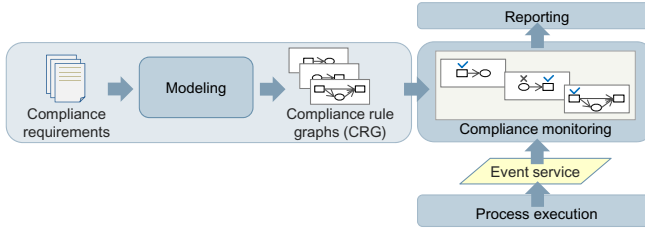
as cleared. As activities can be carried out multiple times, e.g., for different items, there can be multiple activations of a compliance rule in a process execution<sup>1</sup>. As the example also shows, the activations can be in different compliance states. For example, the activation for *item 3* is violated while the one for *item 2* is not. Therefore, the monitoring tier must be able to identify the rule activations and to provide information on their individual compliance state as the basis for effectively assessing and dealing with incompliance.

**Challenge 2: Proactive Prevention of Violations.** Generally, each activation of a compliance rule can be in one of three compliance states in a stage of process execution: satisfied, violated, or violable. Satisfied and violated are permanent states. For example, the activation of  $c_4$  for *item 3* is violated while the activation of  $c_3$  for *item 2* is satisfied. A violable activation, however, can become both violated or satisfied depending on the events observed in the future.

The state violable can have very different semantics depending on the rule activations. Consider for example the activation of  $c_3$  for *item 1* (referred to as ACT1) and the activation of  $c_4$  for *item 2* (referred to as ACT2). Then, both are violable as ACT1 can become violated if *item 1* is not marked as cleared in the future and ACT2 can become violated if *item 2* is put on a payment list again. Obviously, different measures are necessary to render ACT1 and ACT2 satisfied. In order to proactively prevent violations, the monitoring tier has to make the state violable more transparent to process supervisors. In particular, support with regard to how to render an activation satisfied is desirable. Being aware that ACT1 can be rendered satisfied by marking *item 1* as cleared, a process supervisor may, for example, schedule this task. To our best knowledge, challenge 2 has not been addressed yet.

**Challenge 3: Root Cause Identification in Case of Violations.** The identification of violations is only a first step. A rule activation can become violated,

<sup>1</sup> Note that there can also be rules that are only activated once, e.g., cardinality rules. We consider them as being activated upon the process start.



**Fig. 2.** Fundamental architecture of the SeaFlows compliance monitoring framework

when required events were not observed or prohibited events were observed within a particular scope. Particularly for more complex rules, a violation can have multiple causes, each of which may require different compensation mechanisms. Therefore, being able to identify the root cause of a violation would facilitate the application of adequate compensation.

## 1.2 Contributions

In this paper, we propose an approach developed in the SeaFlows project<sup>2</sup> that tackles these three challenges. The basic architecture of the framework is depicted in Fig. 2. We adopt a graph-based compliance rule language referred to as Compliance Rule Graphs (CRG) [2]. Our approach enables to “instantiate” a CRG each time a new activation of it is observed and thus to individually monitor the activations. As a result, feedback can not only be provided on the overall enforcement of a compliance rule but also on its activations. Our monitoring approach is based on a pattern matching mechanism that exploits the structure of CRGs for monitoring and introduces markings to indicate observed event patterns that are relevant to the compliance rule to be checked. From the markings, it is possible to derive measures to proactively prevent violations in case of violable rule activations, for example to derive pending activities. For violated activations, we can derive the root cause from the markings.

In the following, we first introduce necessary fundamentals on CRGs in Sect. 2. The monitoring framework is then presented in Sect. 3. Sect. 4 introduces our proof-of-concept implementation. In Sect. 5, we discuss related work. Sect. 6 summarizes the paper and provides an outlook on future research.

## 2 Compliance Rule Graph Fundamentals

For our monitoring framework we adopted compliance rule graphs (CRG), a graph-based compliance rule modeling language [2], as the graph structure has some advantages that we can exploit for monitoring as we will later show. The CRG language adopts the typical rule structure (i.e., if some conditions apply, then some consequence must also apply). As we address compliance rules on

<sup>2</sup> [www.seaflows.de](http://www.seaflows.de)

the occurrence, absence, and ordering relations of events, the rule antecedent and rule consequences are constituted by event patterns, respectively. Their explicit structure makes it easier to comprehend CRGs than for example complex logic formulas. For brevity reasons, we will focus on compliance rules with only one consequence event pattern in this paper. Our approach is, however, also applicable to compliance rules with multiple consequence patterns.

Based on the assumption adopted from graph-based process modeling languages that a graph is a suitable representation for expressing occurrence and ordering relations of events, the event patterns associated with the rule antecedent and the rule consequence are modeled by means of directed graphs. In order to distinguish between the antecedent and the consequence, they are modeled using designated node types (condition and consequence node types). Thus, from their looks, CRGs are acyclic directed graphs with different node types with a graph fragment describing the rule antecedent pattern and a graph fragment describing the rule consequence pattern.

These event patterns can be modeled using occurrence nodes, i.e., nodes that represent the occurrence of events of associated type and properties (e.g., data conditions) and edges constraining their ordering (similar to what we are used to from process modeling)<sup>3</sup>. Beside occurrence nodes, absence nodes representing the absence of certain events can be used to further refine the event patterns. By combining these modeling primitives, it is possible to model sophisticated event patterns, which can serve as antecedent or consequence of a CRG. Def. [11](#) provides a basic definition of CRGs.

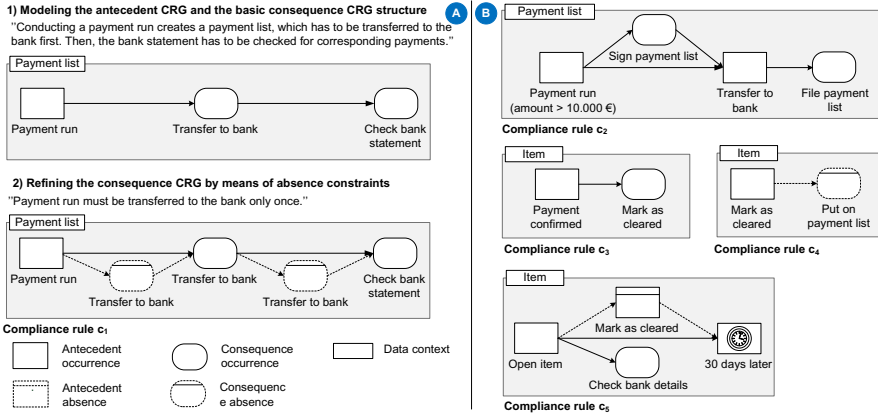
**Definition 1 (Compliance rule graph).** *A compliance rule graph is a 7-tuple  $R = (N_A, N_C, E_A, E_C, E_{AC}, nt, p)$  where:*

- $N_A$  is a set of nodes of the antecedent graph of  $R$ ,
- $N_C$  is a set of nodes of the consequence graph of  $R$ ,
- $E_A$  is a set of directed edges connecting nodes of  $N_A$ ,
- $E_C$  is a set of directed edges connecting nodes of  $N_C$ ,
- $E_{AC}$  is a set of directed edges connecting nodes of the antecedent and the consequence graph of  $R$ ,
- $nt : N_A \cup N_C \rightarrow \{\text{ANTEOCC}, \text{ANTEABS}, \text{CONSOCC}, \text{CONSABS}\}$  is a function assigning a node type to the nodes of  $R$ , and
- $p$  is a function assigning a set of properties (e.g., activity type, data conditions) to each node of  $R$ .

Assuming an existing event model, Fig. [31A](#) depicts the modeling of the CRG for compliance rule  $c_1$  in two steps. Apparently,  $c_1$  is activated by the *payment run* creating a *payment list*. This is modeled through the corresponding **ANTEOCC** node. As  $c_1$  requests the *payment run* to be followed by the event *transfer to bank* and the subsequent event *check bank statement* for the created *payment list*, **CONSOCC** nodes are used to model this consequence pattern. In a second step, the

<sup>3</sup> Note that the antecedent pattern can also be left empty to model for example cardinality constraints.





**Fig. 3.** Step by step modeling CRG for  $c_1$  (A) and CRGs for  $c_2 - c_5$  (B)

consequence CRG is refined to capture the condition that the *payment list* must be transferred to the bank only once. This condition is captured by **CONSABS** nodes signifying the requested absence of additional events of type *transfer to bank*. In this manner, we can also model the other compliance rule examples by means of CRG as depicted in Fig. 3 B. For example, the CRG for compliance rule  $c_5$  expresses that in case an *open item* event is followed by a time event representing 30 days later without having recorded *mark as cleared*, the bank details have to be checked. Note that we assume an event processing framework that can deliver such time events.

Intuitively, an event pattern of a CRG (regardless of whether it is an antecedent or a consequence pattern) matches a set of events if the occurrence nodes and the ordering relations match a set of nodes and the absence constraints expressed through the absence nodes are satisfied. If the antecedent pattern is composed from only **ANTEABS** nodes, then there can be only one match of the antecedent (if the absence constraints apply). Each match of the antecedent event pattern constitutes an *activation* of the corresponding CRG. For example, the sequence of the events  $e_1$  (*payment run* with amount beyond €10,000) and  $e_2$  (*payment list A* is transferred to the bank) from Fig. 1 constitutes a match of the antecedent of  $c_2$ . CRGs with empty antecedent pattern are activated upon the process start.

**Definition 2 (Semantics of CRGs).** Let  $R = (N_A, N_C, \dots)$  be a CRG and  $\sigma = \langle e_1, \dots, e_n \rangle$  be an execution trace. Then,  $R$  is satisfied over  $\sigma$  iff:

- for  $R$  with non-empty antecedent pattern holds: for each match of the antecedent pattern of  $R$  in  $\sigma$ , there is also a corresponding match of  $R$ 's consequence pattern in  $\sigma$  and
- for  $R$  with empty antecedent pattern holds: there is also a match of  $R$ 's consequence pattern in  $\sigma$ .

Due to their explicit structure, verbalization, a technique known from business rule modeling, can be easily realized for CRGs. While CRG is a compositional

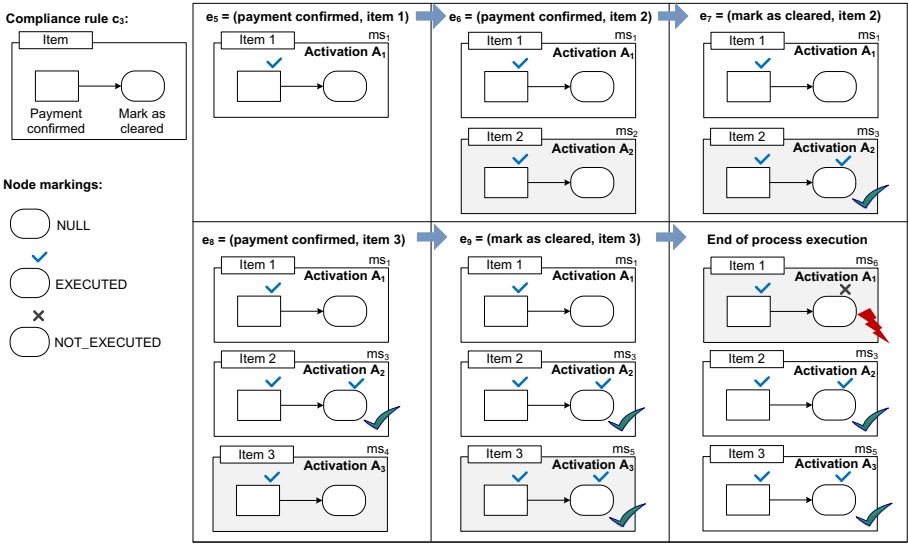


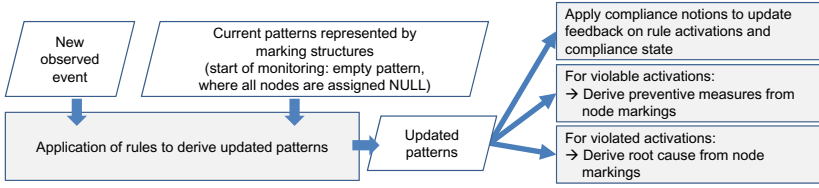
Fig. 4. Observed patterns with regard to CRG  $c_3$  when processing events  $e_5 - e_9$

language, we can also use CRGs to model frequent rule patterns, for example [3]. Due to space limitations, we abstain from going into further details on the properties of CRGs (e.g., syntactic correctness, further modeling primitives). Further details can be found in [2]. In this paper, we only focus on a subset of the CRG language that is sufficient to illustrate our monitoring framework.

### 3 Compliance Monitoring

Generally, it would be possible to monitor compliance with an imposed CRG by transforming it into an automaton or by generating event queries (e.g., for complex event processing [1]) from it. The benefits and drawbacks of these approaches are discussed in Section 5. For example, addressing challenge 1 is cumbersome when employing the automaton approach [4]. The beauty of our approach is that no transformation of the modeled compliance rules (in the form of CRGs) into other representations is necessary in order to enable monitoring. A CRG is instead monitored by exploiting its graph structure. Thus, feedback can be provided specifically on the basis of the structure of the very CRG leaving no gap between the modeled rule and the feedback mechanism.

The basic idea behind the approach is illustrated by Fig. 4 using the example of CRG  $c_3$ . Fig. 4 depicts event patterns that are relevant to  $c_3$  and that become observable in different stages of process execution when processing the events of trace  $T$  from Fig. 1. Instead of textually describing these patterns, we use the graph structure of CRGs and node markings that indicate whether or not an event was observed to capture these observable patterns. New patterns are marked in grey color.



**Fig. 5.** Updating compliance feedback when a new event is observed

*Example 1.* Consider Fig. 4 and  $c_3$ . Then, after observing  $e_1-e_5$ , the pattern “*payment confirmed for item 1 without subsequent mark as cleared yet*” becomes observable in the execution trace. This pattern is captured by  $ms_1$  using the node state EXECUTED to indicate that *payment confirmed* was already observed. When observing  $e_6$ , a similar pattern can be formed for *item 2*. Obviously,  $ms_1$  and  $ms_2$  each constitutes an activation of  $c_3$ . When observing  $e_7$ ,  $ms_2$  is no longer current but instead, the situation can be represented by  $ms_1$  and  $ms_3$ , where  $ms_3$  reflects the pattern “*payment confirmed for item 2 with subsequent mark as cleared*”. Thus,  $ms_3$  constitutes a satisfied activation of  $c_3$ . Observation of  $e_8$  yields a new pattern for *item 3* represented by  $ms_4$ . This pattern is replaced by  $ms_5$  after *item 3* is marked as cleared (event  $e_9$ ). If the process execution would be terminated,  $ms_1$  would be no longer current for activation  $A_1$  as *item 1* will not be marked as cleared. Thus,  $ms_1$  is updated to  $ms_6$ . Altogether, after execution of  $e_1-e_9$  and the termination of the process execution, the compliance with  $c_3$  is reflected by the patterns  $ms_6$ ,  $ms_3$ , and  $ms_5$ , where each of this represents an activation of  $c_3$ . While the activations  $A_2$  and  $A_3$  are satisfied as the required events were observed, activation  $A_1$  for *item 1* is violated as it was not marked as cleared as indicated by the CONSOCC node marked with NOTEXECUTED.

In the example, we built the observable patterns manually. Inspired by pattern matching, our monitoring framework automatically identifies all activations of a CRG to be checked and further tries to identify a match of the consequence pattern in the execution trace. For that purpose, it builds such patterns based on the graph structure of the CRG to be checked that become observable in the partial execution trace as illustrated in the example. However, in the monitoring framework, these patterns are not built from scratch each time a new event is observed. Instead, new observable patterns are *derived* from existing patterns by applying defined rules when observing a new event. Fig. 5 summarizes the overall procedure when a new event is observed.

Each relevant observable pattern represented by marking the CRG as illustrated in the example is stored in a data structure called *marking structure* (in brief MS). From the node semantics and the node markings, it can be derived whether a MS represents a rule activation. Also the individual compliance state of the rule activation can be determined this way (challenge 1). For a violable rule activation, measures can be derived from the node markings to proactively

enforce the satisfaction, for example, the pending activities can be identified (challenge 2). In case of violation, the node markings further enable root cause analysis without additional cost (challenge 3).

In the following, we first formalize the MSs and introduce formal notions to assess them with regard to compliance in Section 3.1. In Section 3.2, we introduce the algorithm for deriving updated observed patterns from old patterns when a new execution event is observed. Finally, we further show how the challenges 2 and 3 can be dealt with in Section 3.3.

### 3.1 CRG Markings and Compliance Notions

In Fig. 4, we already introduced the node markings that are used to indicate whether or not an event was observed: A CRG node  $n$  marked with `NULL` signifies that no matching event is observed yet. Regardless of the node type, a CRG node  $n$  in a pattern marked with `EXECUTED` means that a matching event has been observed. A CRG node  $n$  marked with `NOTEXECUTED` means that the associated event has not been and will not be observed (e.g., when the window for an event to occur has elapsed). Using the node markings, we can use the specific CRG structure to express relevant patterns that become observable in the execution trace. During compliance monitoring for a CRG, each such pattern is captured in a data structure called *marking structure* (MS), e.g.,  $ms_1$  in Fig. 4. In particular, a MS captures a (potential) activation of a CRG observable from the trace. It contains a marking for each antecedent node of the CRG and multiple markings for the CRG’s consequence pattern<sup>4</sup>. Def. 3 formalizes the notion of MSs.

**Definition 3 (CRG MS).** *Let  $R = (N_A, N_C, \dots)$  be a CRG and  $NodeStates := \{\text{NULL}, \text{EXECUTED}, \text{NOTEXECUTED}\}$  be the set of execution states of CRG nodes. Then, a CRG MS of  $R$  is defined as a 3-tuple  $ms := (ns_A, ev_A, \{(ns_C^1, ev_C^1), \dots, (ns_C^k, ev_C^k)\})$  where*

- $ns_A : N_A \rightarrow NodeStates$  is a function assigning an execution state to each node of  $A$ .
- $ev_A$  is a function assigning an observed execution event (or a dummy event in case  $ns_A(n) \in \{\text{NULL}, \text{NOTEXECUTED}\}$ ) to each node of  $A$ . We denote  $(ns_A, ev_A)$  as `ANTEMARK` of  $R$ .
- $ns_C^k : N_C \rightarrow NodeStates$  is a function assigning an execution state to each node of  $C$ .

<sup>4</sup> The rationale behind this is that depending on the particular CRG, the pattern matching procedure may have to try different options to form a match of the consequence pattern out of the events contained in the execution trace. Consider for example the rule “After  $A$ , there has to be a  $B$  that is not followed by a  $C$ ”. Then, assuming that an  $A$  is present in the trace, the first subsequent  $B$  may not lead to a match of the consequence as it can still be followed by a  $C$ . In this case, it becomes necessary to also explore other options, which results in multiple markings for the consequence pattern. This only becomes necessary for the particular case of `CONSOCC` nodes with direct `CONSABS` successors and is taken into account by our pattern matching mechanism (cf. Section 3.2).

- $ev_C^i$  is a function assigning an observed execution event (or a dummy event in case  $ns_C^i(n) \in \{\text{NULL}, \text{NOTEXECUTED}\}$ ) to each node of  $C$ . We denote  $(ns_C^i, ev_C^i)$  as **CONSMARK** of  $R$ .

*Example 2.* Consider  $ms_1$  from Fig. 4

- $ms_1 = (ns_A, ev_A, \{(ns_C, ev_C)\})$  with
- $ns_A(\text{payment confirmed}) = \text{EXECUTED}$ ,  $ev_A(\text{payment confirmed}) = e_5$ ,
- $ns_C(\text{mark as cleared}) = \text{NULL}$ , and  $ev_C(\text{mark as cleared}) = \text{no event}$ .

As illustrated by Fig. 4, in each stage of process execution, the compliance with an imposed CRG can be reflected by a set of MSs. To assess these MSs, we can benefit from the node semantics and the node markings. Generally, for a CRG's antecedent or consequence pattern composed from occurrence and absence nodes, a match in the execution trace is found if all events associated with occurrence nodes are observed (i.e., marked as **EXECUTED**) and for all absence nodes, no matching events were observed (i.e., marked as **NOTEXECUTED**). If the antecedent is marked accordingly, the corresponding MS constitutes an activation of the CRG. Recall that a rule activation is satisfied if a match of the CRG's consequence can also be found in the execution trace. Thus, the activation is satisfied if the MS also contains a **CONSMARK** that is marked as described. Def. 4 formalizes this intuition:

**Definition 4 (Compliance Notions for MSs).** Let  $R = (N_A, N_C, \dots)$  be a CRG and  $ms = (ns_A, ev_A, \{(ns_C^1, ev_C^1), \dots, (ns_C^k, ev_C^k)\})$  be a MS of  $R$ . Then,

- we will say  $ms$  is **ACTIVATED** if the following holds:
  - $\forall n \in N_A : nt(n) = \text{ANTEOCC} \Rightarrow ns_A(n) = \text{EXECUTED} \wedge$   
 $\forall n \in N_A : nt(n) = \text{ANTEABS} \Rightarrow ns_A(n) = \text{NOTEXECUTED}$

For an **ACTIVATED**  $ms$ , we further distinguish between the following states:

- $ms$  is **SATISFIED** if the following holds:
  - $\exists ns_C^i, i \in \{1, \dots, k\} :$   
 $(\forall n \in N_C : nt(n) = \text{CONSOCC} \Rightarrow ns_C^i(n) = \text{EXECUTED}) \wedge$   
 $(\forall n \in N_C : nt(n) = \text{CONSABS} \Rightarrow ns_C^i(n) = \text{NOTEXECUTED})$
- $ms$  is **VIOLATED** if the following holds:
  - $\forall ns_C^i, i \in \{1, \dots, k\} :$   
 $(\exists n \in N_C : nt(n) = \text{CONSOCC} \Rightarrow ns_C^i(n) = \text{NOTEXECUTED}) \vee$   
 $(\exists n \in N_C : nt(n) = \text{CONSABS} \Rightarrow ns_C^i(n) = \text{EXECUTED})$
- Otherwise,  $ms$  is considered **VIOLABLE**.

*Example 3.* Consider again Fig. 4. Then,

- $ms_1$ ,  $ms_3$ , and  $ms_6$  are all **ACTIVATED**, i.e., they constitute activations of  $c_3$ .
- $ms_1$  is **VIOLABLE**,  $ms_3$  is **SATISFIED**, while  $ms_6$  is **VIOLATED**.

Def. 4 enables us to assess obtained MSs. Altogether, this enables the process supervisor to get an overview on rule activations in the process execution and provides basic information on their compliance state. In Section 3.3, we will further discuss how the monitoring framework can be used to assist the process supervisor in identifying the root cause for violations and even in preventing violations. Before that, we first introduce the pattern matching mechanism operating on MSs behind our framework in Section 3.2.

### 3.2 The Pattern Matching Mechanism

As mentioned, each MS represents a (potential) activation of the CRG. As indicated in Fig. 5, the monitoring of a CRG starts with a MS where all nodes are assigned NULL (i.e., no events observed yet). How to derive updated MSs from existing MSs when a new event is observed? The pattern matching mechanism of the framework is based on three considerations:

**1:** The objective is to identify all rule activations present in the execution trace. For that purpose, it becomes necessary to explore different options to form a match of the CRG’s antecedent pattern out of the events in the trace in the pattern matching process.

**2:** For each MS, the objective is further to identify a match of the consequence CRG (cf. Def. 2). For that purpose, we try to explore only one option if possible to increase the efficiency. Alternative options are only explored if necessary.

**3:** Exploit the ordering of nodes for pattern matching: A node can only match an event, if the node and event specification match and the node is not yet assigned to another event. Additionally, also matching events for relevant predecessors must have already been found. In particular, for ANTEOCC and ANTEABS nodes, ANTEOCC predecessors must be already marked as EXECUTED. For CONSOCC and CONSABS nodes, ANTEOCC and CONSOCC predecessors must be marked as EXECUTED.

Based on these considerations, algorithm 1 derives updated patterns from a MS when an event is observed. The outer loop implements consideration 1. The inner loop in line 19 updates the observable patterns with regard to the consequence CRG (represented by the CONSMARKS). It further implements consideration 2, as only for particular nodes, namely CONSOCC nodes with direct CONSABS successors, alternative options have to be explored.

*Example 4.* Fig. 6A applies algorithm 1 to  $c_2$  over the events  $\langle e_1, e_2, e_4 \rangle$ .<sup>5</sup> New MSs are highlighted. The monitoring starts with  $ms_1$ . When  $e_1$  is observed, application of algorithm 1 yields both  $ms_1$  and  $ms_2$ . Here,  $ms_1$  enables the recognition of future activations of  $c_2$ , while  $ms_2$  explores whether  $e_1$  leads to a rule activation.<sup>6</sup> So far, no activation of  $c_2$  is observed yet. When observing

<sup>5</sup>  $e_3$  is irrelevant to  $c_2$ .

<sup>6</sup> Note that when manually conducting pattern matching (cf. Fig. 4), we would typically not identify  $ms_1$ . However, such not yet matching patterns are important to enable automatically deriving updated patterns from existing patterns.

---

**Algorithm 1.** Deriving updated patterns from a MS over an event
 

---

```

1:  $R = (A, C, \dots, \dots)$  is a CRG;  $e$  is an observed event;  $MS_{Res} = \emptyset$ ;
2:  $ms = (ns_A, ev_A, \{(ns_C^i, ev_C^i), \dots, (ns_C^k, ev_C^k)\})$  is a MS of  $R$ ;
   {CRG nodes that match  $e$  (cf. consideration 3)};
3:  $N_{AnteOcc}$  is the set of ANTEOCC,  $N_{AnteAbs}$  is the set of ANTEABS nodes matching  $e$ ;
4:  $N_{ConsOcc}^i, i = 1, \dots, k$ ; is the set of CONSOCC nodes matching  $e$  of  $(ns_C^i, ev_C^i)$ ;
5:  $N_{ConsAbs}^i, i = 1, \dots, k$ ; is the set of CONSABS nodes matching  $e$  of  $(ns_C^i, ev_C^i)$ ;
6: for all  $Q \in \mathcal{P}(N_{AnteOcc})$  do
7:   create a copy  $ms'$  of  $ms$ ;
8:   for all  $n \in Q$  do
9:      $ns'_A(n) = \text{EXECUTED}$ ;  $ev'_A(n) = e$ ;
10:    mark all ANTEABS predecessors of  $n$  with  $ns'_A(n) = \text{NULL}$  as NOTEXECUTED;
11:    for all CONSMARKS  $(ns_C^i, ev_C^i)$  of  $ms'$  do
12:      mark all CONSOCC predecessors of  $n$  with  $ns_C^i(n) = \text{NULL}$  as NOTEXECUTED;
13:      mark all CONSABS predecessors of  $n$  with  $ns_C^i(n) = \text{NULL}$  as NOTEXECUTED;
14:    end for
15:  end for
16:  for all  $n$  with  $n \in N_{AnteAbs} \wedge ns'_A(n) = \text{NULL}$  do
17:     $ns'_A(n) = \text{EXECUTED}$ ;  $ev'_A(n) = e$ ;
18:  end for
19:   $CM = \emptyset$ ;
20:  for all CONSMARKS  $cm = (ns_C^i, ev_C^i)$  of  $ms'$  do
21:     $N = N_{ConsOcc}^i \setminus \{n \in N_C \mid nt(n) = \text{CONSOCC} \wedge ns_C^i(n) = \text{NOTEXECUTED}\}$ ;
22:     $D = \{n \in N \mid n \text{ has no direct CONSABS successor}\}$ ;
23:     $I = N \setminus D$ ;
24:    for all  $Q = D \cup T, T \in \mathcal{P}(I)$  do
25:      create a copy  $cm' = (ns_C^i, ev_C^i)$  of  $cm$ ;
26:      for all  $n \in Q$  do
27:         $ns_C^i(n) = \text{EXECUTED}$ ;  $ev_C^i(n) = e$ ;
28:        mark all CONSABS predecessors of  $n$  with  $ns_C^i(n) = \text{NULL}$  as NOTEXECUTED;
29:      end for
30:      for all  $n$  with  $n \in N_{ConsAbs}^i \wedge ns_C^i(n) = \text{NULL}$  do
31:         $ns_C^i(n) = \text{EXECUTED}$ ;  $ev_C^i(n) = e$ ;
32:      end for
33:       $CM = CM \cup \{cm'\}$ ;
34:    end for
35:  end for
36:  set CONSMARKS of  $ms' = CM$ ;
37:   $MS_{Res} = MS_{Res} \cup \{ms'\}$ ;
38: end for
39: return  $MS_{Res}$ ;

```

---

$e_2$ ,  $ms_1$  remains unaffected. However,  $ms_2$  results in  $ms_2$  and  $ms_3$ . Here,  $ms_2$  enables the recognition of possible future rule activations in combination with  $e_1$ . The compliance notions (cf. Def. 4) reveal that  $ms_3$  constitutes an activation of  $c_2$ ,  $A_1$ , that is already VIOLATED as the payment list was not signed before being transferred to the bank. In this case, the process supervisor can be notified. If in practice the activity *transfer to bank* can be put on hold, the system could even suspend its execution being aware that the activity leads to incompliance. Despite the violation, the monitoring of  $A_1$  can still be continued. Thus, when observing  $e_4$ ,  $ms_3$  yields  $ms_4$ . Activation  $A_1$  is still violated, but nevertheless due to one but not two causes as we will later discuss in Section 3.3.

*Example 5.* Fig. 6B applies algorithm 1 to  $c_4$  over the events  $\langle e_7, e_9, e_{10} \rangle$ . As *mark as cleared* occurs twice the execution (as  $e_7$  for *item 2* and as  $e_9$  for *item 3*), compliance monitoring reveals two activations of  $c_4$ , namely  $A_1$  and

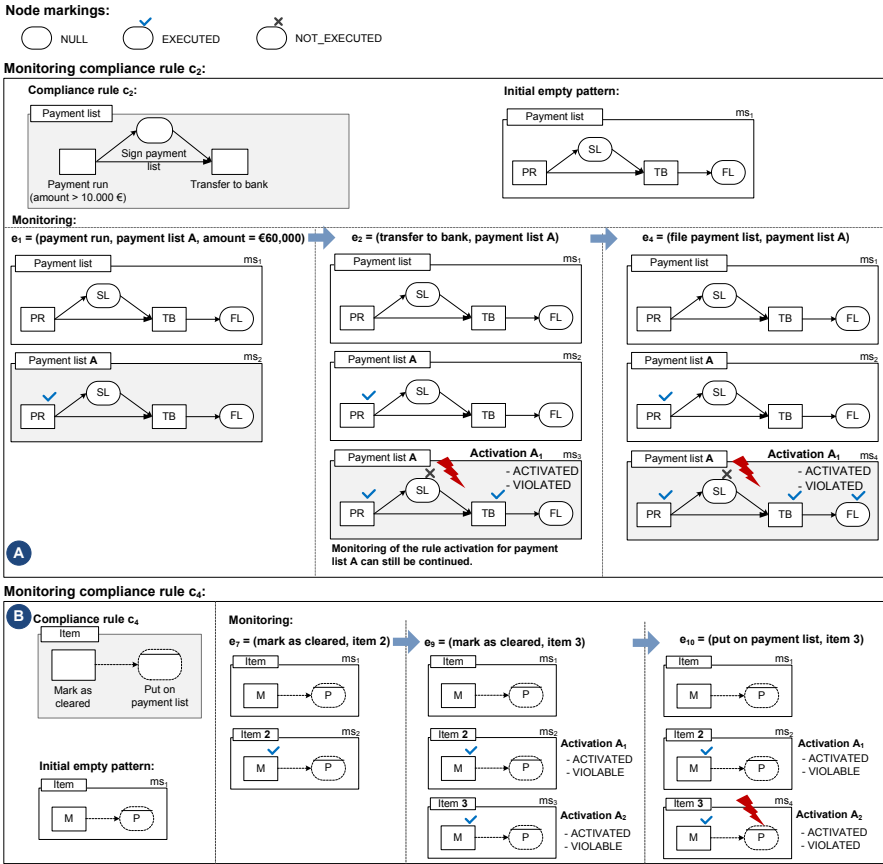


Fig. 6. Monitoring  $c_2$  over  $\langle e_1, e_2, e_4 \rangle$  (A) and  $c_4$  over  $\langle e_7, e_9, e_{10} \rangle$  (B)

$A_2$ , after observing  $e_7$  and  $e_9$ . At that stage, both activations are **VIOLABLE**. However, *item 3* is later put on a payment list again (indicated by  $e_{10}$ ). As a result, activation  $A_2$  becomes **VIOLATED** as the absence constraint is violated.

### 3.3 Prevention of Violations and Root Cause Analysis

*Prevention of violations* A **VIOLABLE** rule activation can become both **SATISFIED** or **VIOLATED** depending on future events. Due to its graph notation, such a MS can be presented to the process supervisor if required in order to identify measures to prevent a violation. Additionally, the system can assist in preventing violations by deriving concrete actions in order to render the activation **SATISFIED**. Based of Def. 4, the rule activation becomes **SATISFIED** when a match of the consequence is found. Thus, from a **CONSMARK** ( $ns_C, ev_C$ ) that can still lead to a match of the consequence, we can derive actions to satisfy the corresponding activation as follows:



- Each **CONSOCC** node  $n$  with  $ns_C(n) = \text{NULL}$  represents a still **pending activity**.

**Possible action:** Schedule the pending activity<sup>7</sup>.

Example: Consider  $ms_1$  from Fig. 4. Then, **CONSOCC** node **mark as cleared** is pending as it is still marked as **NULL**. To satisfy this activation, the corresponding activity can be scheduled, for example, by putting it into an agent’s worklist.

- Each **CONSABS** node  $n$  with  $ns_C(n) = \text{NULL}$  that does not have any **CONSOCC** predecessors still in state **NULL** represents an *active absence constraint*. The absence of the corresponding event is necessary in order for this **CONSMARK** to constitute a match of the consequence **CRG**.

**Possible action:** Deactivate the corresponding activity until  $n$  is marked as **NOTEXECUTED** (e.g., when the window of  $n$  elapsed).

Example: Consider  $ms_2$  from Fig. 6B. Then, **CONSABS** node *put on payment list* represents an active absence constraint. As no **CONSOCC** nodes are pending, immediate end of process execution would render this activation **SATISFIED**. To enforce compliance, the activity *put on payment list* can be deactivated for *item 2*.

As the pattern matching mechanism employs a rather greedy strategy to identify a match of the consequence, the thus derived action chains constitute the “shortest” ways to enforce compliance.

*Root cause identification.* In a similar manner to violation prevention, the root cause of a **VIOLATED** rule activation can be easily derived from a **CONSMARKS**. Generally, an activation can become **VIOLATED** if required events do not occur or / and prohibited events occur during process execution. These causes are also reflected in the **MS**. For a **CONSMARK** ( $ns_C, ev_C$ ) of a **VIOLATED MS**, we can identify why it does not constitute a match of the consequence **CRG**:

- Each **CONSOCC** node  $n$  with  $ns_C(n) = \text{NOTEXECUTED}$  represents a **required activity missing** in the pattern.

Example: Consider  $ms_4$  from Fig. 6A. Then, the missing event *sign payment list* before transferring *payment list A* to the bank can be precisely identified as the root cause for the violation of rule activation  $A_1$ .

- Each **CONSABS** node  $n$  with  $ns_C(n) = \text{EXECUTED}$  represents an **prohibited activity observed** in the pattern.

Example: Consider  $ms_4$  from Fig. 6B. Then, the prohibited event *put on payment list* for *item 3* is identified as the root cause for the violation.

## 4 Implementation

We implemented our monitoring approach in the SeaFlows Toolset [5] that comprises a variety of tools for supporting compliance throughout the process

<sup>7</sup> Since **CRGs** are acyclic, we can further derive a process to be scheduled by adopting the ordering relations of the occurrence nodes in case multiple activities are pending.

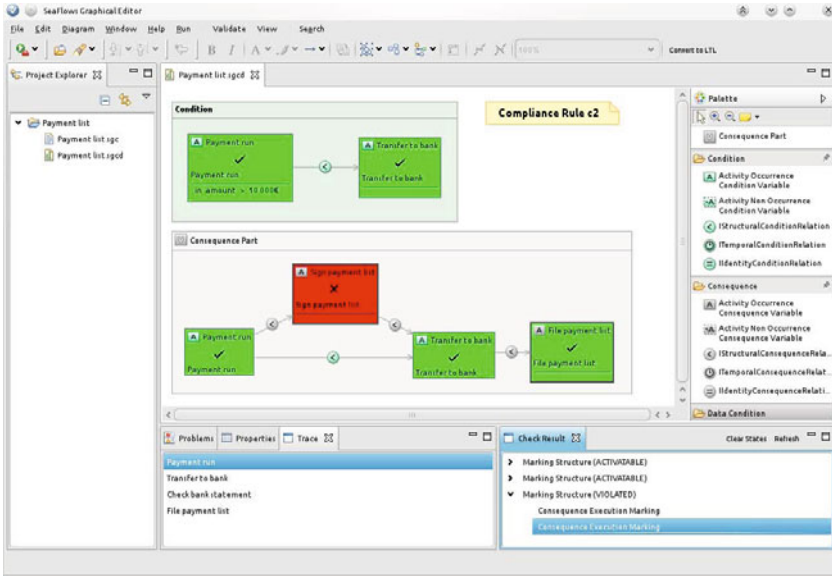


Fig. 7. Execution of  $c_2$  over  $\langle e_1, e_2, e_4 \rangle$  and  $ms_4$  as one of the resulting MSs

lifecycle. The SeaFlows Compliance Monitor is integrated into the AristaFlow BPM Suite that is based on ADEPT [6]. CRGs are modeled using a graphical editor and stored as XML files. For convenient compliance rule modeling, the SeaFlows Compliance Rule Editor allows for modeling parametrized compliance rules patterns that can be reused whenever a rule with a similar structure is required. Fig. 7 shows the rule activation captured by MS  $ms_4$  (cf. Fig. 6A) obtained when observing the event sequence  $\langle e_1, e_2, e_4 \rangle$  from Fig. 1. The root cause of the violation is visualized directly in the CRG by using the node markings and the color highlighting. This enables process supervisors to easily pinpoint violations and to apply root cause specific remedies, for example initiate an audit activity as the transferred payment list was not signed before being transferred to the bank.

## 5 State-of-the-Art

Most work addressing process monitoring focus on data consistency or process performance (e.g., KPI monitoring or business activity monitoring). Several approaches address root cause analysis in the context of design time verification [7,8,5]. Generally, for a predefined set of rule patterns, possible types of violations can be anticipated, which can be used for root cause analysis. However, runtime monitoring of complex rules on the occurrence, absence, and ordering of particular events necessitates more advanced strategies. We distinguished three classes of monitoring approaches addressing constraints on the behavior of events. In addition, compliance monitoring is also related to *conformance checking*.

*Automaton-based monitoring.* One approach to monitor compliance with imposed rules is to use an automaton that reaches an accepting state if the rule to be checked is satisfied. As compliance rules are typically not modeled as automaton, they first have to be modeled using a formalism, such as linear temporal logic (LTL), from which an automaton can be generated. To hide the complexity of LTL from the modeler, graph notations for frequently used constraint patterns based on the work of Dwyer and Corbett [3], such as ConDec [9], were suggested. Maggi et al. [10] suggest a monitoring approach based on LTL and colored automata. It includes information about the accepting states of the automata of the individual constraints in a global automaton representing the conjunction of all imposed constraints. The latter is important to identify whether constraints are conflicting. In case a violation occurs, the monitoring can still be continued. Generally, challenge 1, the support of individual rule activations, is cumbersome to tackle using automaton-based approaches as this would require an additional instantiation mechanism. In addition, it is a non-trivial task to derive meaningful information from a non-accepting state of a generated automaton in case of violations (e.g., root cause).

*Logic-based monitoring.* Some approaches employ logic formalisms to conduct monitoring. In [4], Montali et al. introduce an event calculus formalization for ConDec [9] constraints, which supports the identification of constraint activations. While this approach can also deal with temporal scopes, the formalization was done for existence, absence, and response constraints. Alberti et al. [11] report on monitoring contracts expressed as rules using the notion of happened and expected events. At runtime, events are aggregated in a knowledge base and reasoning is employed to identify violations. It seems that proactive prevention of violations and root cause analysis were not addressed by these approaches.

*Violation pattern based monitoring* Incompliance with rules on the occurrence, absence, and ordering of events can also be detected by querying the (partial) execution trace for violation patterns. To conduct the querying, existing frameworks and technologies such as complex event processing (CEP) [1] are applicable. In [8], Awad et al. introduce anti-patterns for basic rule patterns such as precedence. While this approach addresses design time verification of process models, the anti-patterns can also be applied to query the execution trace. For simple compliance rules or basic relations (as for example introduced in [12]), all violation patterns can be anticipated. However, for more complex compliance rules on the occurrence, absence, and ordering of events that can be violated in multiple ways, automatic computation of violation patterns to identify all possible violations becomes a real challenge. This has not been addressed yet. In their work, Giblin et al. [13] developed the REALM rule model. For REALM patterns, such as “y must occur within time t after x”, they provide transformations into ACT correlation rules, which can be used for detecting relevant event patterns. Event processing technologies are further used by numerous compliance monitoring frameworks to detect violations, e.g., in the COMPAS project [14][15]. How the event queries are generated from complex compliance rules is not the focus of these approaches.

*Conformance checking.* Conformance checking investigates whether a process model and process logs are conform to each other. Generally, the conformance can be tested for example by replaying the log over the process model. To tackle this, several approaches were proposed [16,17] that introduce techniques and notions such as fitness and appropriateness to also quantify conformance. Implementations of these approaches (e.g., the Conformance Checker) are available in the process mining framework ProM [18]. Conformance checking and compliance rule monitoring exhibit major differences that require different techniques. For example, compliance rules are typically declarative while process models are mostly procedural. In addition, most of the work on conformance checking operates a posteriori. However, these approaches provide good inspiration, for example to develop metrics for quantifying compliance. Weidlich et al. show in [12] how to derive event queries for monitoring process conformance from a process model. They employ a behavioral profile that serves as an abstraction of the process model. The profile captures three relations among the activities of a process model (e.g., strict order relation). For these relations, event monitoring queries can be generated (cf. discussion on violation pattern based monitoring).

## 6 Summary and Outlook

In this paper, we addressed three major challenges in the context of monitoring the compliance with imposed rules. Our framework enables the identification of all activations of a compliance rule. In case of a compliance rule is violated, it becomes possible to pinpoint the rule activations involved. In addition, as our framework does not require any transformations into other representations in order to conduct the monitoring, feedback and diagnosis can be given specifically based on the corresponding rule structure. In particular, we can derive the root cause for a violation from the node markings of the particular rule. Even for rule activations that are not yet permanently violated, we can derive actions (in particular pending activities and active absence constraints) that can be helpful to process supervisors to proactively prevent violations. The validity of the approach was shown based on our proof-of-concept implementation. Our monitoring approach is not restricted to CRGs but can also be adapted to deal with other graph-based rule languages. We also conducted research to further increase the efficiency of our approach, for example by pruning paths to be explored using domination rules. In future work, we will further address the interplay of CRGs, for example conflicting rules.

## References

1. Jacobsen, H.A., Muthusamy, V., Li, G.: The PADRES event processing network: Uniform querying of past and future events. *IT - Information Technology*, 250–261 (2009)
2. Ly, L.T., Rinderle-Ma, S., Dadam, P.: Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems. In: Pernici, B. (ed.) *CAiSE 2010*. LNCS, vol. 6051, pp. 9–23. Springer, Heidelberg (2010)

3. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proc. ICSE 1999, pp. 411–420 (1999)
4. Montali, M., Maggi, F., Chesani, F., Mello, P., van der Aalst, W.: Monitoring business constraints with the event calculus. Technical report. Universita degli Studi di Bologna (2011)
5. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., Dadam, P.: SeaFlows toolset – compliance verification made easy for process-aware information systems. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 76–91. Springer, Heidelberg (2011)
6. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases* 16, 91–116 (2004)
7. Elgammal, A., Turetken, O., van den Heuvel, W.-J., Papazoglou, M.: On the formal specification of regulatory compliance: A comparative analysis. In: Maximilien, E.M., Rossi, G., Yuan, S.-T., Ludwig, H., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6568, pp. 27–38. Springer, Heidelberg (2011)
8. Awad, A., Weske, M.: Visualization of compliance violation in business process models. In: Proc. BPI 2009 (2009)
9. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
10. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011)
11. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: Expressing and verifying business contracts with abductive logic. In: *Normative Multi-agent Systems*. Number 07122 in Dagstuhl Seminar Proceedings (2007)
12. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-Based Monitoring of Process Execution Violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 182–198. Springer, Heidelberg (2011)
13. Giblin, C., Müller, S., Pfitzmann, B.: From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Technical Report Research Report RZ-3662. IBM Research GmbH (2006)
14. Holmes, T., Mulo, E., Zdun, U., Dustdar, S.: Model-aware monitoring of soas for compliance. In: Dustdar, S., Li, F. (eds.) *Service Engineering*, pp. 117–136. Springer, Heidelberg (2011)
15. Birukou, A., D’Andrea, V., Leymann, F., Serafinski, J., Silveira, P., Strauch, S., Tluczek, M.: An integrated solution for runtime compliance governance in SOA. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 122–136. Springer, Heidelberg (2010)
16. van der Aalst, W.M.P., de Medeiros, A.K.A.: Process mining and security: Detecting anomalous process executions and checking process conformance. *Electr. Notes Theor. Comput. Sci.* 121, 3–21 (2005)
17. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* 33, 64–95 (2008)
18. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W(E.), Weijters, A.J.M.M.T.: ProM 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)

# History-Aware, Real-Time Risk Detection in Business Processes

Raffaele Conforti<sup>1</sup>, Giancarlo Fortino<sup>2</sup>,  
Marcello La Rosa<sup>1</sup>, and Arthur H.M. ter Hofstede<sup>1,3</sup>

<sup>1</sup> Queensland University of Technology, Australia  
{[raffaele.conforti](mailto:raffaele.conforti@qut.edu.au),[m.larosa](mailto:m.larosa@qut.edu.au),[a.terhofstede](mailto:a.terhofstede@qut.edu.au)}@qut.edu.au

<sup>2</sup> Università della Calabria, Italy  
[fortino@unical.it](mailto:fortino@unical.it)

<sup>3</sup> Eindhoven University of Technology, The Netherlands

**Abstract.** This paper proposes a novel approach for identifying risks in executable business processes and detecting them at run-time. The approach considers risks in all phases of the business process management lifecycle, and is realized via a distributed, sensor-based architecture. At design-time, sensors are defined to specify risk conditions which when fulfilled, are a likely indicator of faults to occur. Both historical and current process execution data can be used to compose such conditions. At run-time, each sensor independently notifies a sensor manager when a risk is detected. In turn, the sensor manager interacts with the monitoring component of a process automation suite to prompt the results to the user who may take remedial actions. The proposed architecture has been implemented in the YAWL system and its performance has been evaluated in practice.

## 1 Introduction

According to the AS/NZS ISO 31000 standard, a business process risk is the chance of something happening that will have an impact on the process objectives, and is measured in terms of likelihood and consequence [26]. Incidents such as scandals in the finance sector (the 4.9B Euros fraud at Société Générale), in the health sector (Patel Inquiry) and in the aviation industry (failed terrorist attacks) have shown that business processes are constantly exposed to a wide range of risks.

Failures of process-driven risk management can result in substantial financial and reputational consequences, potentially threatening an organization's existence. Legislative initiatives such as the Sarbanes-Oxley Act [1] and Basel II [2] in the finance sector have highlighted the pressing need to better manage business process risks. As a consequence of these mandates, organizations are now seeking new ways to *control* process-related risk and are attempting to incorporate it as a distinct view in their operational management. However, whilst conceptually appealing, to date there is little guidance as to how this can best be done. Currently the disciplines of process management and risk management are largely disjoint and operate independently of one another. In industry they are usually handled by different organizational units. Within academia, recent research has centered on the identification of process-related risks. However the incidents

---

<sup>1</sup> [www.gpo.gov/fdsys/pkg/PLAW-107publ204](http://www.gpo.gov/fdsys/pkg/PLAW-107publ204)

described above demonstrate that a focus on risk analysis alone is no longer adequate, and an active, real-time risk detection and controlling is required.

We propose a novel approach for operationalizing risk management in Business Process Management automation Suites (BPMSs). The aim of this approach is to provide a concrete mechanism for identifying risks in executable business process models and detecting them during process execution. This is achieved by considering risks throughout the BPM lifecycle, from process model design where risk conditions are defined, through to process diagnosis, where risks are monitored. By automating risk detection, the interested users (e.g. a process administrator) can be notified as early as a risk is detected, such that remedial actions can be taken to rectify the current process instance, and prevent an undesired state of the process (*fault* for short), from occurring. Based on historical data, we can also compute the probability of a risk at run-time, and compare it to a threshold, so as to notify the user only when the risk's criticality is no longer tolerable. To the best of our knowledge, this is the first attempt to incorporate risks into executable business processes and enable their automatic detection at run-time.

The proposed approach is realized via a distributed, sensor-based architecture. A *sensor* is an independent software component which monitors a risk condition capturing the situation upon which the risk of a given fault may occur. Conditions can be determined via a query language that can fetch both historical and current execution data from the logs of the BPMS. At run-time sensors are registered with a central sensor manager. At a given sampling rate, or based on the occurrence of a (complex) event, the sensor manager retrieves and filters all data relevant for the various sensors (as it is logged by the BPMS engine), and distributes it to the relevant sensors. If a sensor condition holds, i.e. if the probability of the associated risk is above a given threshold, the sensor alerts the sensor manager which in turn notifies the monitoring component of the BPMS. The distributed nature of the architecture guarantees that there is no performance overhead on the BPMS engine, and thus on the execution of the various process instances. We implemented this architecture on top of the YAWL system. We extended the YAWL Editor to cater for the design of risk sensors, and equipped the run-time environment with a sensor manager service that interacts with YAWL's monitoring service and execution engine.

To prove the feasibility of the proposed approach, we used fault tree analysis [4] (a well-established risk analysis method) to identify risk conditions in a reference process model for logistics, in collaboration with an Australian risk consultant. These risks embrace different process aspects such as tasks' order dependencies, involved resources and business data, and relate to historical data where needed, to compute risk probabilities. We expressed these conditions via sensors in the YAWL environment, and measured the time needed to compute these conditions at run-time. The tests showed that the sensor conditions can be computed in a matter of milliseconds without impacting on the performance of the running process instances.

This paper is organized as follows. Section 2 illustrates the running example in the logistics domain. Section 3 describes our risk-aware BPM approach while Sect. 4 presents the sensor-based architecture to implement this approach. The architecture is evaluated in Sect. 5. Section 6 covers related work while Sect. 7 concludes the paper.

## 2 Running Example

In this section we use an example to illustrate how the risk of possible faults to occur during a business process execution can be identified as early as possible. In particular, we show how risks can be expressed in terms of process-specific aspects such as tasks occurrence, data or available resources. The example, shown in Figure 1, describes the payment subprocess of an order fulfillment business process which is inspired by the VICS industry standard for logistics [30]. This standard is endorsed by 100+ companies worldwide, with a total sales volume of \$2.3 Trillion annually [30]. The notation used to represent this example is that of YAWL [13], although a deep knowledge of this language is not required.

This process starts after the freight has been picked up by a carrier and deals with the shipment payment. The first task is the production of a Shipment Invoice containing the shipment costs related to a specific order for a specific customer. If shipments have been paid in advance, all that is required is for a Finance Officer to issue a Shipment Remittance Advice specifying the amount being debited to the customer. Otherwise, the Finance Officer issues a Shipment Payment Order that needs to be approved by a Senior Finance Officer (who is the superior of this Finance Officer). At this point, a number of updates may be made to the Shipment Payment Order by the Finance Officer that issued it, but each of these needs to be approved by the Senior Finance Officer. After the document is finalized and the customer has paid, an Account Manager can process the shipment payment by specifying the balance. If the customer underpaid, the Account Manager needs to issue a Debit Adjustment, the customer needs to pay the balance and the payment needs to be reprocessed. A customer may also overpay. In this case the Account Manager needs to issue a Credit Adjustment. In the latter case and in case of a correct payment, the shipment payment process is completed.

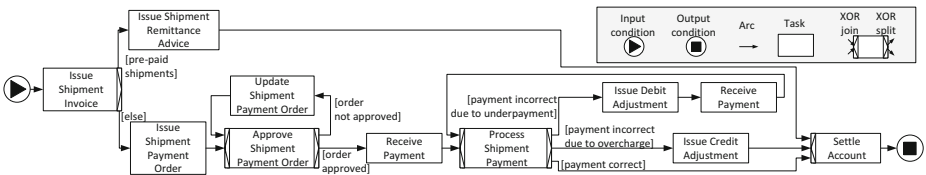


Fig. 1. Order-Fulfillment: Payment subprocess

In collaboration with a risk analyst of an Australian consulting company, we identified four faults that can occur during the execution of this payment subprocess. In order to prevent the occurrence of these faults, for each of them we also defined an associated risk condition by using fault tree analysis [4]. Accordingly, each risk condition is expressed as a set of lower-level boolean events which are organized in a tree via logical connectives such as ORs, ANDs and XORs. Below we describe the four risk conditions identified. However, for space reasons, in Fig. 2 we only show the fault tree for two of them.

The first fault is an *overtime process* fault. A Service Level Agreement (SLA) for a process or for a given task within a process, may establish that the process (or task) may



not last longer than a Maximum Cycle Time  $MCT$ , otherwise the organization running the process may incur a pecuniary penalty. In our case, an overtime fault occurs if an instance of the payment subprocess is not completed within an  $MCT$  of five days.

To detect the risk of overtime fault at run-time, we should check the likelihood that the running instance does not exceed the  $MCT$  based on the amount of time  $T_e$  expired at the current stage of the execution. Let us consider  $T_e$  as the remaining cycle time, i.e. the amount of time estimated to complete the current instance given  $T_c$ . Then the probability of exceeding  $MCT$  can be computed as  $1 - MCT / (T_e + T_c)$  if  $T_e + T_c > MCT$  and is equal to 0 if  $T_e + T_c \leq MCT$ . If this probability is greater than a tolerance value (e.g. 60%), we notify the risk to the user. The estimation of the remaining cycle time is based on past executions of the same process and can be computed using the approach in [29] (see Section 5 for more details).

The second fault is related to the resources participating in the process. The Senior Finance Officer who has approved a Shipment Payment Order for a given customer, must have not approved another order by the same customer in the last  $d$  days, otherwise there is an *approval fraud*. This fault is thus generated by the violation of a four-eye principle across different instances of the Payment subprocess.

To detect the risk of this fault we first have to check that there is an order, say order  $o$  of customer  $c$ , to be approved. This means checking that an instance of task Approve Shipment Payment Order is being executed. Moreover, we need to check that either of the following conditions holds: i)  $o$  has been allocated to a Senior Finance Officer who has already approved another order for the same customer in the last  $d$  days; or ii) at least one Senior Finance Officer is available who approved an order for customer  $c$  in the last  $d$  days and all other Senior Finance Officers who never approved an order for  $c$  during the last  $d$  days are not available. The corresponding fault tree is shown in Fig. 2.

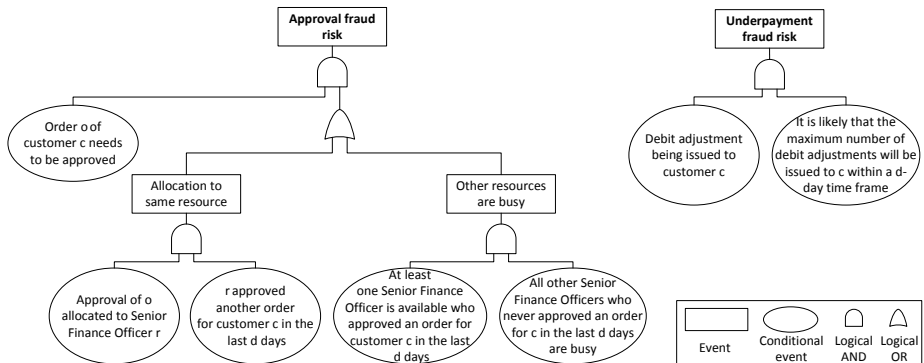


Fig. 2. The fault trees for Approval Fraud and Underpayment Fraud

The third fault relates to a situation where a process instance executes a given task too many times. This situation typically occurs in the context of loops. Not only could this lead to a process slowdown but also to a “livelock” if the task is in a loop whose exit condition is purposefully never met. In general, given a task  $t$  a maximum number

of allowable executions of  $t$  per process instance  $MAE^i(t)$  can be fixed as part of the SLA for  $t$ . With reference to the Payment subprocess, this can occur for example if task Update Shipment Payment Order is re-executed five times within the same process instance. We call this an *order unfulfillment* fault.

To detect the risk of this fault at run-time, we need to check if: i) an order  $o$  is being updated (i.e. task Update Shipment Payment Order is currently being performed for order  $o$ ); and ii) it is likely that this order will be updated again (i.e. task Update Shipment Payment Order will be repeated within the same process instance). The probability that the number of times a task will be repeated within the same instance of the Payment subprocess is computed by dividing the number of instances where the  $MAE^i$  for task Update Shipment Payment Order has been reached, over the number of instances that have executed this task at least as many times as it has been executed by the current instance, and have completed. The tolerance value indicates a threshold above which the risk should be notified to the user. For example, if this threshold is 60% for task  $t$ , a risk should be raised if the probability of  $MAE^i(t)$  is greater than 0.6.

The fourth fault is an *underpayment fraud*. It relates to a situation in which a given task is executed too many times across multiple process instances. Similar to the previous fault, given a task  $t$  we can define a maximum number of allowable executions of  $t$  per process  $MAE^p(t)$  as part of the SLA for  $p$ . In our example, this type of fault occurs when a customer underpays more than three times within the last five days.

To detect the risk of underpayment fraud, we need to check if: i) a debit adjustment is currently being issued to a customer  $c$  (i.e. task Issue Debit Adjustment is currently being performed for customer  $c$ ); and ii) it is likely that the maximum number of debit adjustments will be issued to the same customer in a  $d$ -day time frame. The probability that  $MAE^p$  is reached for task Issue Debit Adjustment of customer  $c$  in  $d$  days is computed by dividing the number of customers for which the  $MAE^p$  for task Issue Debit Adjustment has been reached within  $d$  days, over the number of customers for which this task has been executed at least as many times as it has been executed for  $c$  within  $d$  days. Similar to the previous risk, if this probability is above a tolerance value, the risk should be raised and the user notified. The corresponding fault-tree is shown in Fig. 2.

The faults identified in this example, and the associated risks, can easily be generalized to other domains. For example, a fault due to an approval fraud can occur in any business process that involves an approval procedure (e.g. loan approvals).

### 3 Risk-Aware Business Process Management

As we have seen in the context of the payment example, a fault in a business process is an undesired state of a process instance which may lead to a process failure (e.g. the violation of a policy may lead to a process instance being interrupted). Identifying a fault in a process requires determining the condition upon which the fault occurs. For example, in the payment subprocess, we have an underpayment fraud if a customer underpays more than three times within a five-day time frame.

However, a *fault condition* holds only when the associated fault has occurred, which is typically too late to avoid a process failure. Indeed, we need to be able to estimate the risk of a process fault, i.e. if, and possibly with what likelihood, the fault will occur

in the future. Early risk detection allows process users to promptly react with countermeasures, if any, to prevent the related fault from occurring at all.

We use the notion of *risk condition*, as opposed to fault condition, to describe the set of events that lead to the possibility of a fault to occur in the future. In order to evaluate risk conditions “on-line”, i.e. while a process instance is being executed, we need to consider the current state of the BPMS. This means knowing the state of all running instances of any process (and not only the state of the instance for which we are computing the risk condition), the resources that are busy and those that are available, and the values of the data variables being created and consumed. Moreover, we need to know the historical data, i.e. the execution data of all instances that have been completed. In particular, we can use historical data to estimate the probability of a given fault to occur, i.e. the *risk probability*. For example, for the underpayment fraud, we can estimate the likelihood that another debit adjustment is being issued for a given combination of customer/order (historical data), given that one such debit adjustment has just been issued (current data). To obtain a boolean risk condition, we compare the risk probability that we obtain with a tolerance value, such that the condition holds if the risk probability exceeds the given threshold. For example, we raise the risk of underpayment fraud if the risk probability is greater than 60%.

In other cases, we may avoid to embed a risk probability in the risk condition, if we are able to determine the occurrence of a set of events which directly leads to a high risk. This is the case of the approval fraud, where both the events “Allocation to same resource” and “Other resources are busy” already signal a high risk of approval fraud.

Based on these considerations, we present a novel approach for on-line risk detection in business processes. The focal idea of this approach, shown in Fig. 3 is to embed elements of risk into all four phases of the traditional BPM lifecycle [7].

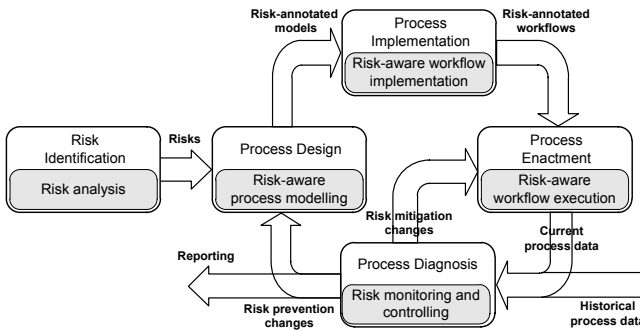


Fig. 3. Risk-aware Business Process Management lifecycle

Input to this “risk-aware” BPM lifecycle is a *Risk Identification* phase, where risk analysis is carried out to identify risks in the process model to be designed. Traditional risk analysis methods such as FTA (as seen in the previous section), Root Cause Analysis [17] or CORAS [25], can be employed in this phase. The output of this phase is a set of risks, each expressed as a risk condition.

Next, in the *Process Design* phase, these high-level risk conditions are mapped down to process model-specific aspects. For example, the condition “debit adjustment being issued to customer *c* for order *o*” is mapped to the occurrence of a specific task, namely “Issue Debit Adjustment” in the Payment process model. The result of this second phase is a risk-annotated process model. In the next phase, *Process Implementation*, these conditions are linked to workflow-specific aspects, such as content of variables, and resource allocation states. For example, “customer *c*” is linked to the *Customer* element of the XML representation of the *Debit Adjustment* document. Process Implementation may be integrated with Process Design if the language used at design-time is executable (e.g. BPMN 2.0 or YAWL).

The risk-annotated workflow model resulting from Process Implementation is then executed by a risk-aware process engine during *Process Enactment*. Historical data stored in process logs, and current execution data coming from process enactment, are filtered, aggregated and analyzed in the *Process diagnosis* phase, in order to evaluate the various risk conditions. When a risk condition evaluates to true, the interested users (e.g. a process administrator) are notified and reports can also be produced during this phase for auditing purposes. Finally, this phase can trigger changes in the current process instance, to mitigate the likelihood of a fault to occur, or in the underlying process model, to prevent a given risk from occurring ever again.

In the next section we describe a sensor-based architecture to operationalize this enhanced BPM lifecycle.

### 4 Sensor-Based Realization

In order to realize our risk-aware BPM lifecycle, we devised an approach based on sensors. In a nutshell, the idea is to capture risk and fault conditions via sensors, and then monitor these sensors during process execution. An overview of this approach is shown in Fig. 4 using the BPMN 2.0 notation [21].

*Sensors* are defined during the *Process Design* and *Process Implementation* phases of our risk-aware BPM lifecycle (see Fig. 3), for each process model for which the presence of risks and/or faults need to be monitored. If the process model is specified via an executable language, then these two phases coincide.

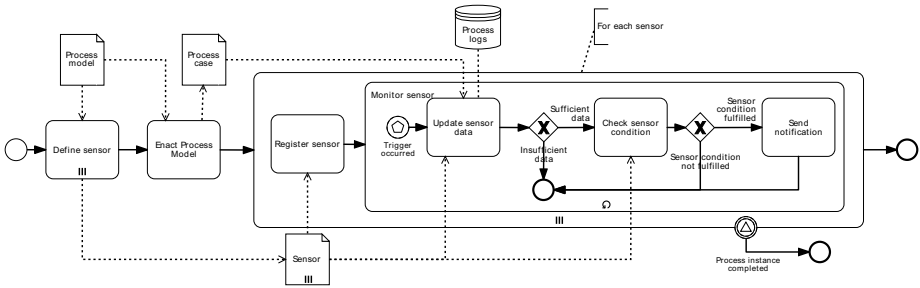


Fig. 4. Realization of risk-aware BPM lifecycle via sensors

$$\begin{aligned}
\text{Sensor} &\triangleq v : \text{Variables}; c : \text{Condition}; \\
&\quad t : \text{Trigger} \\
\text{Variables} &\triangleq \text{Assignment}^+ \\
\text{Condition} &\triangleq \text{riskCond}, \text{faultCond} : \text{boolExpr} \\
\text{Trigger} &\triangleq \text{timer} \mid \text{event} \\
\text{Assignment} &\triangleq \text{CaseExpr} \mid \text{CaseElemExpr} \mid \\
&\quad \text{VarFunc} \mid \text{Definition} \\
\text{CaseExpr} &\triangleq \text{result} : \text{varName}; \\
&\quad e : \text{CaseIDStat}; a : \text{Action} \\
\text{CaseElemExpr} &\triangleq \text{ce} : \text{CaseExpr}; x : \text{TaskOrNet} \\
&\quad \text{VarFunc} \triangleq \text{result}, \text{input} : \text{varName}; \\
&\quad \text{va} : \text{varAction} \\
\text{Definition} &\triangleq \text{result} : \text{varName}; c : \text{constant} \\
\text{CaseIDStat} &\triangleq \text{absoluteExpr} \mid \text{relExpr} \mid \\
&\quad \text{CaseCondSet} \\
\text{CaseCondSet} &\triangleq \text{CaseCondExpr} \mid \text{CaseCond} \mid \\
&\quad \text{CaseParam} \\
\text{CaseCondExpr} &\triangleq \text{pes}_1, \text{pes}_2 : \text{CaseCondSet}; \\
&\quad \text{bo} : \text{booleanOp} \\
\text{CaseCond} &\triangleq x : \text{TaskOrNet}; a : \text{Action}; \\
&\quad c : \text{compOp}; r : \text{rightHandExpr} \\
\text{CaseParam} &\triangleq i : \text{idFunc}; c : \text{compOp}; \\
&\quad r : \text{rightHandExpr} \\
\text{TaskOrNet} &\triangleq \text{taskLabel} \mid \text{netName} \\
\text{Action} &\triangleq \text{predFunc} \mid \text{taskOrNetVar} \mid \\
&\quad \text{SubVarExpr} \mid \text{inputPredFunc}
\end{aligned}$$

(a)

Abstract element	Description
<i>Sensor</i>	is composed by <i>Variables</i> , <i>Condition</i> , <i>Trigger</i>
<i>Variables</i>	identifies a set of <i>Assignment</i>
<i>Condition</i>	identifies the sensor condition composed by a risk condition and by a fault condition
<i>Trigger</i>	specifies the type of trigger desired
<i>Assignment</i>	defines a mapping between a variable and a piece of information
<i>CaseExpr</i>	identifies information belonging to the process instance
<i>CaseElemExpr</i>	identifies information belonging to an element of the process instance
<i>VarFunc</i>	returns the result of a function executed on a variable
<i>Definition</i>	sets the value of a variable to a predefined value
<i>CaseIDStat</i>	identifies a process instance or a set of process instances
<i>CaseCondSet</i>	describes how a process instance can be identified
<i>CaseCondExpr</i>	is a boolean conjunction of <i>CaseCondSet</i>
<i>CaseCond</i>	specifies the condition that the process instance must satisfy
<i>CaseParam</i>	specifies the parameter related to the process instance id
<i>TaskOrNet</i>	identifies an element of the process model using <i>taskLabel</i> or <i>netName</i>
<i>Action</i>	identifies the type of information desired

(b)

**Fig. 5.** Abstract syntax of sensor definition language (a); Description of its elements (b)

A sensor is defined through a boolean *sensor condition*, constructed on a set of process variables, and a *sensor activation trigger*. Process variables are used to retrieve information from the specific instance in which the sensor condition will be evaluated as well as from other instances, either completed or still running. For example, we can use variables to retrieve the resource allocated to a given task, the value of a task variable, or the status of a task. Process instances can either be identified based on the current instance (e.g. the last five instances that have been completed before the current one), or based on the fulfillment of a *case condition* (e.g. “all instances where a given resource has executed a given task”). The sensor condition can represent either a *risk condition* associated with a fault, or a *fault condition*, or both. If both conditions are specified, the fault condition is evaluated only if the risk condition evaluates to true. For example, the sensor will check if an overtime process fault has occurred in a process instance only if first the risk of such fault has first been detected, based on the estimation of the remaining cycle time for this instance. Finally, the sensor activation trigger can be either a timer periodically fired according to a sampling rate (e.g. every 5 minutes), or an event emitted by the process engine (e.g. the completion of a task). Figure 5 shows a simplified version of the sensor definition language by using an abstract syntax [19]; the complete definition of this language is provided in the technical report [5].

During *Process Enactment*, the defined sensors are registered with a *sensor manager*, which activates them. In the *Process Diagnosis* phase, which starts as soon as the process is enacted, the activated sensors receive updates on the variables of their sensor conditions according to their trigger (timer or event). When a sensor receives an update, it checks its sensor condition. If the condition holds, a notification is sent from the sensor to the monitor service of the BPMS.

The sensor manager relies on three interfaces to interact with the BPMS (see Fig. 6(a)):

- *Engine interface*, used to register a sensor with a particular event raised by the BPMS engine. When the event occurs the sensor is notified by the sensor manager.
- *Database interface*, used to query the BPMS database in order to collect current and historical information.
- *Monitor interface*, used to notify the detection of risks and faults to the monitor service of the BPMS.

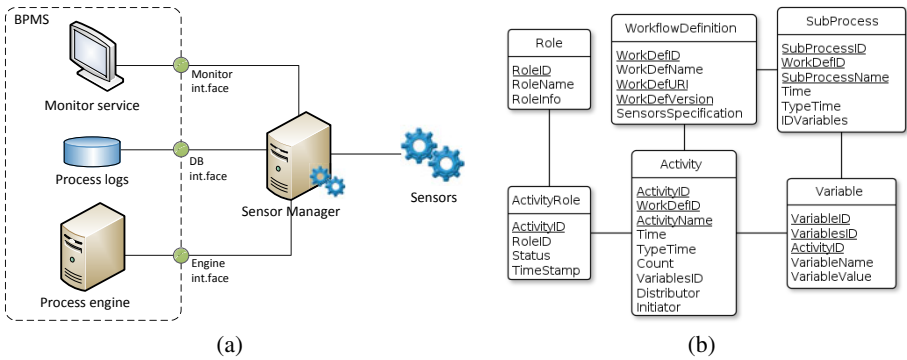


Fig. 6. Sensor-based architecture (a); Database Interface schema model (b)

These interfaces can be implemented by the vendor or user of the BPMS where the sensor manager needs to be installed. In this way, our sensor manager can virtually be interfaced with any BPMS. As an example, the conceptual model of the database interface is showed in Fig. 6(b), where methods have been omitted for space reasons. This conceptual model is inspired by the reference process meta-model of the WfMC [14], in order to cover as many aspects as possible of a workflow model, and meantime, to remain as generic as possible. For example, class *WorkflowDefinition* allows one to retrieve information about the process model where the sensor is defined, such as process identifier and name, while class *SubProcess* allows one to retrieve information about a specific subprocess, and so on. This interface should be implemented according to the characteristics of the specific database used in the BPMS at hand. For an efficient use of the interface, one should also define indexes on the attributes of the BPMS database that map the underlined attributes in Fig. 6(b). These indexes have been determined based on the types of queries that can be defined in our sensor condition language.

An alternative approach to achieve the portability of the sensor manager, would be to read the BPMS logs from a standard serialization format such as OpenXES. However, as we will show in Sect. 5, this solution is rather inefficient.

The advantages of using sensors are twofold. First, their conditions can be monitored while the process model is being executed, i.e. in real-time. Second, according to a distributed architecture, each sensor takes care of checking its own condition after being activated by the sensor manager. In this way, potential execution slowdowns are avoided (e.g., the process engine and the sensor manager could be deployed to two different machines).

We now have all ingredients to show how the risks that we identified for the Payment subprocess can be captured via sensor conditions, using the language defined in Fig. 5. For space reasons we only focus on the approval fraud and underpayment fraud risks. A description of the other sensor conditions is provided in the technical report [5].

We recall that there is an approval fraud whenever a Senior Finance Officer approves two orders for the same customer within five days. Accordingly, the corresponding risk can be detected if given an order  $o$  of customer  $c$  to be approved, either of the following conditions holds: i)  $o$  has been allocated to a Senior Finance Officer who has already approved another order for the same customer in the last five days; or ii) at least one Senior Finance Officer is available who approved an order for customer  $c$  in the last five days and all other Senior Finance Officers who never approved an order for  $c$  during the five days are not available.

This risk condition is triggered by an event, i.e. the spawning of a new instance of task *Approve Shipment Payment Order*. This is checked by using a variable to retrieve the status of this task in the current instance. The risk condition itself is given by the disjunction of the two conditions described above. The first such condition is checked by using a variable ( $r_1$ ) to retrieve which resources were allocated to task *Approve Shipment Payment Order*, and another variable ( $n$ ) to retrieve the number of times this task was completed for customer  $c$ . This latter variable is defined via a case condition over customer  $c$ , the completion time of this task (that must be greater than the allocate time ( $t_1$ ) of the current task *Approve Shipment Payment Order* minus five days ( $d$ ) in milliseconds), and the identifier of the instance (that must be different from the identifier of the current instance).

The second condition is checked by using two variables and invoking two functions. A variable ( $r_2$ ) to retrieve which resources completed task *Approve Shipment Payment Order*, and another variable ( $r_3$ ) to retrieve all resources that can be offered this task (i.e. the current task). The first variable is defined via a case condition over customer  $c$  and the completion time of this task (that must be greater than the offered time ( $t_2$ ) of the current task *Approve Shipment Payment Order* minus five days ( $d$ )). The two invoked functions return the number of tasks started on the resources that completed task *Approve Shipment Payment Order*, and the number of tasks in the execution queue of the resources who have been offered this task, and did not complete it for customer  $c$  in the last five days.

The definition of the above variables in our sensor language is provided below, while the *Action* elements used in these definitions are described in Table II.

```

r1 : ResAllocated = case(current).Approve_Shipment_Payment_Order_593(allocateResource)
c : customer = case(current).Issue_Shipment_Invoice_594.ShipmentInvoice.Company
d : days = 5
t1 : AllocateTime = case(current).Approve_Shipment_Payment_Order_593(AllocateTimeInMillis)
t2 : OfferTime = case(current).Approve_Shipment_Payment_Order_593(OfferTimeInMillis)
n : #TimesApproved = case(Approve_Shipment_Payment_Order_593(completeResource)=ResAllocated ^
Issue_Shipment_Invoice_594.ShipmentInvoice.Company=customer ^
Approve_Shipment_Payment_Order_593(CompleteTimeInMillis)>
(AllocateTime-(days*24*60*60*1000)) ^
(ID)!=[IDCurr]).Approve_Shipment_Payment_Order_593(CountElements)
r2 : ResCompleted = case(Issue_Shipment_Invoice_594.ShipmentInvoice.Company=customer ^
Approve_Shipment_Payment_Order_593(isCompleted)="true" ^
Approve_Shipment_Payment_Order_593(CompleteTimeInMillis)>
(OfferTime-(days*24*60*60*1000)) ^
(ID)!=[IDCurr]).Approve_Shipment_Payment_Order_593(completeResource)
r3 : DefResOffered = case(current).Approve_Shipment_Payment_Order_593(offerDistribution)

```

After the definition of the variables, the risk condition is specified as follows:

$$(\#TimesApproved > 0) \vee ((ResCompleted.startMinNumber = 0) \wedge (DefResOffered.startMinNumberExcept.ResCompleted \geq 1)).$$

We recall that an underpayment fraud occurs whenever a customer underpays more than three times in a five-day time frame. Accordingly, the respective risk can be detected if i) task *Issue Debit Adjustment* is being performed for a given customer and order (this is the trigger for this risk); and ii) the probability that the maximum number of allowable executions for this task will be reached in a five-day time frame, is above the fixed tolerance value for this risk, say 60% (this is the risk condition itself). This condition can be checked by using two variables: one ( $n$ ) to retrieve the number of times the task *Issue Debit Adjustment* has been completed for this customer ( $c$ ) within five days ( $d$ ), the other ( $p$ ) to retrieve the probability that an attempted fraud will take place. For this second variable, we use the *Action* “*FraudProbabilityFunc*” to compute the specific probability (see Table II).

The defined variables are implemented through the sensor language as follows:

```

StartTime = case(current).Issue_Debit_Adjustment_605(StartTimeInMillis)
c : customer = case(current).Issue_Shipment_Invoice_594.ShipmentInvoice.Company
d : days = 5
n : #Completions = case(Issue_Shipment_Invoice_594.ShipmentInvoice.Company=customer ^
Issue_Debit_Adjustment_605(Count)>0 ^
Issue_Debit_Adjustment_605(CompleteTimeInMillis)>(StartTime-days*24*60*60*1000)
Issue_Debit_Adjustment_605(CountElements)
GroupingElem = Issue_Shipment_Invoice_594.ShipmentInvoice.Company
WindowElem = Issue_Debit_Adjustment_605(CompleteTimeInMillis)
Threshold = 0.6
p : Probability = case(Issue_Debit_Adjustment_605(Count)>0 ^ (ID)!=[IDcurr]).Issue_Debit_Adjustment_605
(FraudProbabilityFunc, #Completions, 3, GroupingElem, WindowElem, (days*24*60*60*1000))

```

These variables are used to compose the following risk condition:  $Probability > 0.6$ .



**Table 1.** Description of the *Action* elements used in the example sensor conditions

Action	Description
(ID)	returns the ID of the generic instance that is being analyzed
[IDCurr]	returns the ID of the instance that the sensor is monitoring
Count	returns the number of times a task has been completed
allocateResource	returns the resources to which the task has been allocated
completeResource	returns the resource that completed the task
isStarted	returns "true" if the task has been started
isCompleted	returns "true" if the task has been completed
OfferTimeInMillis	returns the time (in millisecond) when the task has been offered
StartTimeInMillis	returns the time (in millisecond) when the task has been started
CompleteTimeInMillis	returns the time (in millisecond) when the task has been completed
ShipmentInvoice.Company	returns the value of the subvariable <i>Company</i> belonging to the variable <i>ShipmentInvoice</i>
offerDistribution	returns list of resources to which the task is offered by default
CountElements	returns the number of instances that satisfy the parameters required
FraudProbabilityFunc	returns the probability of a fraud using as parameters: the current number of executions, the maximum number of executions allowed, the parameter used to group the instances, the parameter used to identify a temporal window, the dimension of the temporal window

## 5 Evaluation

In this section we discuss the implementation of the sensor-based architecture in the YAWL system and then evaluate its performance.

### 5.1 Implementation

In order to prove the feasibility of our approach, we implemented the sensor-based architecture in the YAWL system<sup>2</sup>. We decided to extend the YAWL system for the following reasons. First, this system is based on a service-oriented architecture, which facilitates the seamless addition of new services. Second, the system is open-source, which facilitates its distribution among academics and practitioners, and widely used in practice (the system has been downloaded over 100,000 times since its first inception in the open-source community). Finally, the underlying YAWL language is very expressive as it provides wide support for the workflow patterns [13].

As part of this implementation, we extended the YAWL Editor version 2.2beta with a new component, namely the Sensor Editor, for the specification of sensors within YAWL process models. Such graphical component, shown in Fig. 7, fully supports the specification of sensor conditions as defined in Sect. 4.

Moreover, we implemented the Sensor Manager as a generic component which exposes three interfaces (engine, database and monitor) as described in Sect. 4. We then wrapped this component into a Web service which implements the three interfaces for the YAWL system, allowing the component to interact with the YAWL Engine, the Monitor service and the YAWL database. While there is a straightforward mapping between the YAWL Engine and our engine interface, and between the YAWL Monitor service and our monitor interface, we had to join several YAWL tables to implement our database interface. This is because in the YAWL system, event logs are scattered across different database tables. For example, to retrieve all identifiers of the process

<sup>2</sup> Available at [www.yawl.foundation.org](http://www.yawl.foundation.org)

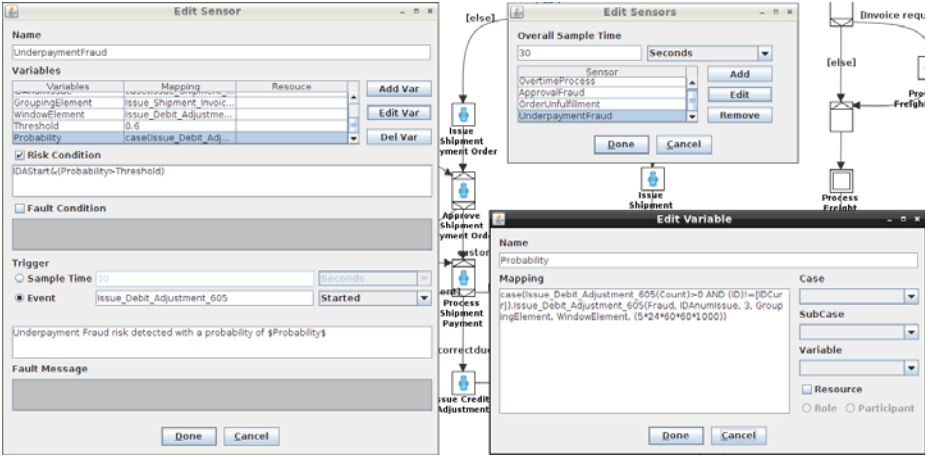


Fig. 7. The Sensor Editor within the YAWL Editor

instances for a specific process model, given the model identifier, we need to perform a join among the following YAWL tables: `logspecification`, `lognetinstance`, `lognet` and `logevent`.

The complete mapping is illustrated in Tab. 2. As an example, this table also shows the mapping between our database interface and the relational schema used by Oracle BPEL 10g to store BPEL process logs. Also in this case, the database can be fully mapped by joining several tables.

Finally, we implemented a separate service to estimate the remaining cycle time  $T_e$  for a process or task instance. This service uses ProM’s prediction miner [29] to compute the estimations, and provides the results to the Sensor Manager on demand. While the estimation of  $T_e$  could be done on-line, i.e. while evaluating a particular sensor condition at run-time, parsing the full logset each time would be inefficient. Rather, we compute this estimation off-line, whenever a new process model is deployed to the YAWL Engine, by using the logset available at that time. Periodically, we update the logset with the new instances being executed meantime, and invoke this service to refresh the estimations for each process model currently deployed.

Table 2. Database interface mapping for YAWL 2.2beta and Oracle BPEL 10g

Database table	Tables that need to be joined	
	YAWL	Oracle BPEL 10g
WorkflowDefinition	logspecification, lognet, lognetinstance, logevent	cube_instance and cube_scope
SubProcess	logspecification, lognet, lognetinstance, logevent	cube_instance and cube_scope
Activity	lognetinstance, logtask, logtaskinstance, lognet, logevent, logspecification, rs_eventlog	wftask and work_item
Variables	logtask, lognet, lognetinstance, logtaskinstance, logevent, logdataitem, logspecification	audit_trail, audit_detail and xml_document
Role	rs_participant	wftask
ActivityRole	rs_eventlog, logtaskinstance	wftask

## 5.2 Performance Analysis

We used our implementation to evaluate the scalability of the approach. First, we measured the time needed to evaluate the basic functions (e.g. counting the number of instances of a task or retrieving the resource allocated to a task). Next, we measured the time needed to evaluate the sensor conditions for the risks defined in the Payment subprocess. The tests were run on an Intel Core I5 M560 2.67GHz processor with 4GB RAM running Linux Ubuntu 11.4. The YAWL logs were stored on the PostGres 9.0 DBMS. These logs contained 318 completed process instances from 36 difference process models, accounting for a total of 9,399 process events (e.g. task instance started and completed, variable's value change). Specifically, there were 100 instances from the Payment subprocess yielding a total of 5,904 process events. The results were averaged over 10 runs.

**Table 3.** Performance of basic functions

Basic function	Description	OpenXES time [ms]	Database time [ms]	Reduction rate [%]
net status	functions checking if a net status has been reached (isStarted, isCompleted)	6,535	18.9	99.71
net time	functions returning the time when a net status has been reached (startTime, completeTime, startTimeInMillis, completeTimeInMillis)	6,781	18.8	99.72
net variable	returns the value of a net variable	6,489	432.6	93.33
task count	number of times a task has been completed	803	19.8	97.53
task resource	functions that return the resources associated with a task (offerResource, allocateResource, startResource, completeResource)	850	20.9	97.54
task status	functions checking if a task status has been reached (isOffered, isAllocated, isStarted, isCompleted)	792	30.5	96.14
task time	functions returning the time when a task status has been reached (offerTime, allocateTime, startTime, completeTime, offerTimeInMillis, allocateTimeInMillis, startTimeInMillis, completeTimeInMillis)	824	22.3	97.29
task variable	returns the value of a task variable	787	96.7	87.71
task distribution	functions returning the resources associated with a task by default (offerDistribution, allocateDistribution, startDistribution, completeDistribution)	243		-
task initiator	functions returning the allocation strategy for a resource association (offerInitiator, allocateInitiator, startInitiator, completeInitiator)	249.6		-

Table 3 shows the results of the evaluation of the basic functions provided by our language. In particular, in this table we compare the evaluation times obtained by accessing the YAWL logs via our database interface, with those obtained by accessing a serialization of the logs, e.g. in the OpenXES format. While OpenXES provides a simple and unique representation of a generic set of process logs, accessing an OpenXES file in real-time, i.e. during the execution of a process instance, is not feasible, due to the long access times (e.g. 6.5 sec. on average for evaluating a net variable). On the other hand, accessing the logs via our database interface, despite it requires the creation of a specific implementation for each BPMS database, provides considerably faster times than accessing OpenXES files (at least 87% gain w.r.t. OpenXES access). In fact, as we can see from Tab. 3, the evaluation times for all the basic functions are below 30 ms, apart from function `task variable`, which takes 100 ms and function `net variable`, which takes 430 ms.

The last two basic functions reported in Tab. 3, namely `task distribution` and `task initiator`, are evaluated in less than 250 milliseconds. These functions

are not computed by accessing the logs, but rather by accessing information that is contained directly in an executable process model, e.g. the resources that are associated with a specific task. However, in our implementation we still use the database interface to access this information, in order to provide the developer with a single access point to all process-related data.

Table 4 reports the results of the evaluation of the sensor conditions defined for our running example. While the sensor conditions for the overtime process and order unfulfillment faults are very low (below 150 ms), longer times are obtained for evaluating the conditions for the two faults related to fraud. This is because both these conditions require to evaluate “complex queries”, i.e. queries over the entire process logs: In the approval fraud, we need to retrieve all resources that approved an order for a specific customer, while in the underpayment fraud we need to retrieve all process instances where a debit adjustment was issued and aggregate these instances per customer. These queries are different than those needed to evaluate the basic functions, as the latter are performed on the events in the logs that are relative to a single known process instance, e.g. the instance for which the sensor condition is being evaluated.

The worst-case complexity of evaluating one such a complex query is still linear on the number of parameters that need be evaluated in the query (corresponding to the language element *Cond-ExprSet* in Sect. 4) multiplied by the total number of instances present in the logs (corresponding to the size of table *WorkflowDefinition* addressed by our database interface).

In conclusion, the performances of evaluating sensor conditions should always be considered w.r.t. the specific process for which the risks are defined, and the type of trigger used. For example, let us assume an average duration of 24 hours for the Payment subprocess, with a new task being executed every 30 minutes. This means we have up to 30 minutes to detect an overtime process risk before a new task is executed, and we need to compute this sensor condition again. If we choose a rate of 5 minutes to sample this condition, we are well below the 6 minute-threshold, so we can check this sensor’s condition up to 6 times during the execution of a task. Since we do this in less than 150 ms, this time is acceptable. For an event-driven risk we also need to consider the frequency of the specific event used as trigger. For example, the approval fraud risk is triggered every time an instance of task Approve Shipment Payment Order is offered to a Senior Financial Officer for execution. Since we take up to 7 seconds to compute this sensor condition, we are able to cope with a system where there is a request for approval every 7 seconds. So also for this sensor, the performance is quite acceptable.

## 6 Related Work

Risk measurement and mitigation techniques have been widely explored in various fields. At the strategic level risk management, standards prescribe generic procedures for identifying, analyzing, evaluating and treating risks (see e.g. [26]). Although helpful, such general guidelines are inevitably vague and fail to provide any specific

**Table 4.** Performance of sensors

Sensor	Min [ms]	Max [ms]	Average [ms]	St.Dev. [ms]
Overtime process	121	137	131.8	4.66
Approval fraud	6,483	7,036	6,766.4	183.06
Order unfulfillment	69	91	77.4	7.18
Underpayment fraud	3,385	3,678	3,523	89.98

guidance for operationalizing risk management strategies in business processes. At the other extreme, there are many techniques for identifying risks in specific areas such as employee fraud [1], conflict of interest [18] and in the engineering field more generally [123]. Other approaches, such as fault-tree analysis [4], are general enough to be applied to multiple domains. However, none of these approaches provides insights on how to define and operationalize the detection of process-related risks.

Previous process-based research recognizes the importance of explicitly linking elements of risk to business process models. zur Muehlen et al. [23,32] propose a taxonomy of process-related risks and describe its application in the analysis and documentation of business processes. This taxonomy includes five process-related risk types (goals, structure, information technology, data and organization) which can be captured by four interrelated model types: i) risk structure model describing the relationships between risks; ii) risk/goal matrix; iii) risk state model describing the dynamic aspects of a risk; and iv) an extension to the EPC notation to assign risks to individual process steps. An extension of the work in [23] is proposed in [20], where the authors describe a four-step approach to integrate risks in business processes at the operational and strategic levels via *value-focused process engineering*.

A different perspective is offered by the ROPE (Risk-Oriented Process Evaluation) methodology [10,28]. ROPE is based on the observation that process activities require resources to be adequately executed. If faults occur (here called “threats”), they impact the functionality of resources until one or more affected resources are no longer available. In the worst case a resource represents a single point of failure and consequently hinders the execution of the related process activity. If a threat is detected, an appropriate countermeasure process is invoked to counteract the threat. However, if this cannot be done, a recovery process can be invoked to re-establish the functionality of the affected resources until they are available again for the respective business process activity. The aim of the ROPE methodology is to incorporate all these aspects in a single model that can be simulated to determine a company’s critical business processes and single points of failure. Finally, on the basis of the ROPE methodology, a reference model for risk-aware BPM is proposed in [15,16].

With respect to the risk-aware BPM lifecycle shown in Fig. 3, all the above proposals only cover the phases of risk analysis and risk-aware process modeling. None of them specifies how risk conditions can be concretely linked to run-time aspects of process models such as resource allocation, data variables and control-flow conditions, for the sake of detecting risks during process execution. Thus, none of these approaches operationalizes risk detection into workflow management systems. Moreover, they neglect historical process data for risk estimation. As such, these approaches are complementary to our work, i.e. they can be used at a conceptual level for the identification of process-related risks, which can then be implemented via our sensor-based technology.

Our sensor-based architecture is also related to real-time monitoring of business process execution. Similarly to our approach, Oracle Business Activity Monitoring (BAM) [22] relies on sensors to monitor the execution of BPEL processes. Three types of sensors can be defined: *activity sensors*, to grab timings and variable contents of a specific activity; *variable sensors*, to grab the content of the variables defined for whole BPEL process (e.g. the inputs to the process); and *fault sensors*, to monitor BPEL faults. These

sensors can be triggered by a predefined set of events (e.g. task activation, task completion). For each sensor, one can specify the endpoints where the sensor will publish its data at run-time (e.g. a database or a JMSQueue). We allow the specification of more sophisticated sensor (and fault) conditions, where different process-related aspects can be incorporated such as data, resource allocation strategies, order dependencies, as well as historical data and information from other running process instances. Moreover, our sensors can be triggered by process events or sampled at a given rate. Nonetheless, our sensor-based architecture is exposed as a service and as such it could be integrated with other process monitoring systems, such as Oracle BAM.

Real-time monitoring of process models can also be achieved via Complex Event Processing (CEP) systems. In this context, CEP systems have been integrated into commercial BPMSs, e.g. webMethods Business Events<sup>3</sup>, ARIS Process Event Monitor [6] and SAP Sybase [27], as well as explored in academia [9,11]. A CEP system allows the analysis of aggregated events from different sources (e.g. databases, email accounts as well as process engines). Using predefined rules, generally defined with a specific SQL-like language [31], a CEP system can verify the presence of a specific pattern among a stream of simple events processed in a given time window. Our approach differs from CEP systems in the following aspects: i) strong business process orientation vs general purpose system; ii) ability to aggregate complex XML-based events (e.g. process variables) and analyze them (e.g. for the sake of computing a risk probability) vs processing simple sequences of events; iii) time-driven and event-driven triggers vs event-driven trigger only. For the same reasons, our sensors differ from (simple) event receptors that are generally available in BPMSs. Moreover, CEP systems typically suffer from performance overheads [11,31] which limit their applicability to real-time risk detection [31].

## 7 Conclusion

The contribution of this paper is twofold. First, it provides a concrete mechanism for identifying risks in executable business process models and for detecting them during process execution. This is achieved by embedding elements of risk within each phase of the BPM lifecycle: from process design, where high-level risks are mapped down to specific process model elements, to process diagnosis, where risk conditions are monitored in real-time. The second contribution is an operationalization of the proposed risk-awareness approach in the context of BPMSs. This is achieved via a distributed, sensor-based architecture that is interfaced with a BPMS via a set of interfaces. Each risk is associated with a sensor condition. Conditions can relate to any process aspect, such as control-flow dependencies, resource allocations, the content of data elements, both from the current process instance and from instances of any process that have already been completed. At design-time, these conditions are expressed via a Java-like query language within a process model. At run-time, each sensor independently alerts a sensor manager when the associated risk condition evaluates to true during the execution of a specific process instance. When this occurs, the sensor manager notifies a process administrator about the given risk by interfacing with the monitoring service of the BPMS. This allows early risk detection which in turn enables proper remedial actions to be taken in order to avoid potentially costly process faults.

<sup>3</sup> <http://www.softwareag.com/au/products/wm/events/overview>

The sensor-based architecture was implemented in the YAWL system and its performance evaluated in practice. The tests show that the sensor conditions can be computed efficiently and that no performance overhead is induced to the BPMS engine. To the best of our knowledge, this is the first attempt to embed risks into executable business processes and enable their automatic detection at run-time. And while we restricted our focus to the realm of risks, our sensor-based architecture can also be applied to other domains where there is a requirement to measure and analyze process data in real-time, e.g. process compliance or cost-monitoring.

This work suffers from several limitations, which provide opportunities for future work. First, it does not support the actual risk mitigation but only risk detection. We plan to devise a mechanism for automatically generating remedial actions that can be applied once a risk has been detected at run-time. The idea is to use genetic algorithms such as simulated annealing [24] to create perturbations on the current process instance in order to rectify its execution and thus avoid a fault from eventually occurring. Our previous application of simulated annealing to the problem of automatically correcting business process models [8] showed that such perturbations can be obtained very efficiently. The challenge stands in properly defining the objective functions so as to create meaningful perturbations. Second, in this paper we only evaluated the performance of the prototype implementation. Clearly this is not enough to state that the approach can be applied in practice. In this regard, we plan to interview a pool of risk analysts drawn from our business contacts in Australia in order to assess the approach's perceived usefulness and ease of use. Finally, we plan to equip the risk modeling component with a set of predefined risks, categorized by type (e.g. approval fraud) and domain (e.g. finance, logistics), which can be used as templates to generate skeletons of risk conditions.

**Acknowledgments.** We thank Colin Fidge for his valuable comments on an early version of this paper, and Michael Adams for his help with the YAWL implementation. This research is partly funded by the ARC Discovery Project "Risk-aware Business Process Management" (DP110100091).

## References

1. Albrecht, W.S., Albrecht, C.C., Albrecht, C.O.: *Fraud Examination*, 3rd edn. South-Western Publishing (2008)
2. Basel Committee on Bankin Supervision. *Basel II - International Convergence of Capital Measurement and Capital Standards* (2006)
3. Bhushan, N., Rai, K.: *Strategic Decision Making: Applying the Analytic Hierarchy Process*, 3rd edn. Springer, Heidelberg (2004)
4. International Electrotechnical Commission. *IEC 61025 Fault Tree Analysis, FTA* (1990)
5. Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M.: *History-aware, real-time risk detection in business processes (extended version)*. QUT ePrints 42222, Queensland University of Technology (2011), <http://eprints.qut.edu.au/42222>
6. Davis, R.B., Brabander, E.: *ARIS Design Platform: Getting Started with BPM*. Springer, Heidelberg (2007)
7. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons (2005)
8. Gambini, M., La Rosa, M., Migliorini, S., ter Hofstede, A.H.M.: *Automated Error Correction of Business Process Models*. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) *BPM 2011*. LNCS, vol. 6896, pp. 148–165. Springer, Heidelberg (2011)

9. Gay, P., Pla, A., López, B., Meléndez, J., Meunier, R.: Service workflow monitoring through complex event processing. In: ETFA. IEEE (2010)
10. Goluch, G., Tjoa, S., Jakoubi, S., Quirchmayr, G.: Deriving resource requirements applying risk-aware business process modeling and simulation. In: ECIS. AISeL (2008)
11. Hermosillo, G., Seinturier, L., Duchien, L.: Using Complex Event Processing for Dynamic Business Process Adaptation. In: SCC. IEEE (2010)
12. Hespos, R., Strassmann, P.: Stochastic Decision Trees for the Analysis of Investment Decisions. *Management Science* 11(10) (1965)
13. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer, Heidelberg (2010)
14. Hollingsworth, D.: The Workflow Reference Model. Workflow Management Coalition (1995)
15. Jakoubi, S., Tjoa, S.: A reference model for risk-aware business process management. In: CRiSIS. IEEE (2009)
16. Jakoubi, S., Tjoa, S., Goluch, S., Kitzler, G.: Risk-Aware Business Process Management: Establishing the Link Between Business and Security. In: Xhafa, F., et al. (eds.) *Complex Intelligent Systems and Their Applications. Optimization and its Applications*, vol. 41, pp. 109–135. Springer Science+Business Media, LLC (2010)
17. Johnson, W.G.: MORT - The Management Oversight and Risk Tree. U.S. Atomic Energy Commission (1973)
18. Little, A., Best, P.: A framework for separation of duties in an sap r/3 environment. *Managerial Auditing Journal* 18(5), 419–430 (2003)
19. Meyer, B.: Introduction to the theory of programming languages. Prentice-Hall (1990)
20. Neiger, D., Churilov, L., zur Muehlen, M., Rosemann, M.: Integrating risks in business process models with value focused process engineering. In: ECIS, AISeL (2006)
21. OMG. Business Process Model and Notation (BPMN) ver. 2.0 (January 2011), <http://www.omg.org/spec/BPMN/2.0>
22. Oracle. BPEL Process Manager Developer's Guide, [http://download.oracle.com/docs/cd/E15523\\_01/integration.1111/e10224/bp\\_sensors.htm](http://download.oracle.com/docs/cd/E15523_01/integration.1111/e10224/bp_sensors.htm) (accessed June 2011)
23. Rosemann, M., zur Muehlen, M.: Integrating risks in business process models. In: ACIS. AISeL (2005)
24. Smith, K.I., Everson, R.M., Fieldsend, J.E., Murphy, C., Misra, R.: Dominance-based multi-objective simulated annealing. *IEEE Trans. on Evolutionary Computation* 12(3) (2008)
25. Soldal Lund, M., Solhaug, B., Stolen, K.: Model-Driven Risk Analysis. Springer, Heidelberg (2011)
26. Standards Australia and Standards New Zealand. Standard AS/NZS ISO 31000 (2009)
27. Sybase. Sybase CEP Implementation Methodology for Continuous Intelligence, [http://www.sybase.com.au/files/WhitePapers/Sybase\\_CEP\\_Implementation\\_Methodology\\_wp.pdf](http://www.sybase.com.au/files/WhitePapers/Sybase_CEP_Implementation_Methodology_wp.pdf) (accessed June 2011)
28. Tjoa, S., Jakoubi, S., Quirchmayr, G.: Enhancing business impact analysis and risk assessment applying a risk-aware business process modeling and simulation methodology. In: ARES, pp. 179–186. IEEE Computer Society (2008)
29. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle Time Prediction: When Will This Case Finally Be Finished? In: Chung, S. (ed.) OTM 2008, Part I. LNCS, vol. 5331, pp. 319–336. Springer, Heidelberg (2008)
30. Voluntary Interindustry Commerce Solutions Association. Voluntary Inter-industry Commerce Standard (VICS), <http://www.vics.org> (accessed June 2011)
31. Wang, D., Rundensteiner, E.A., Ellison, R.T., Wang, H.: Active complex event processing infrastructure: Monitoring and reacting to event streams. In: ICDEW. IEEE (2011)
32. zur Mühlen, M., Ho, D.T.-Y.: Risk Management in the BPM Lifecycle. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 454–466. Springer, Heidelberg (2006)



# Transactional Process Views

Rik Eshuis, Jochem Vonk, and Paul Grefen

Eindhoven University of Technology  
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands  
{h.eshuis,j.vonk,p.w.p.j.grefen}@tue.nl

**Abstract.** To enable effective interorganisational collaborations, process providers have to disclose relevant parts of their local business processes in public process views. A public process view has to be consistent with the underlying private process. Local business processes are typically supported by transactions, which ensure a robust and reliable execution. Process views currently do not support the specification of transactional properties. This paper introduces transactional process views and studies how they can be constructed from an internal business process that is annotated with a transactional specification. This way, we provide a well-structured approach to obtain robust and reliable process behaviour at the public external level, thus facilitating trustworthy, fine-grained collaboration between organisations. We consider various transactional models. The feasibility of the approach is shown by means of a case study.

## 1 Introduction

Due to complex markets, organisations more and more collaborate in dynamic business networks to deliver a requested service or product [18]. To enable an effective collaboration, partners in such a business network have to interconnect their local business processes, such that an inter-organisational business process emerges that is specific to the business network. Since business networks are highly dynamic and change frequently [9], partners are often not willing to fully disclose their local business processes. Moreover, not all details of the local business process are relevant for other partners in the network. Yet an efficient collaboration requires that relevant parts are disclosed.

Public process views have been proposed as means to coordinate and monitor the execution of local, private business processes that are part of a global business network process [3,5,16]. A public process view can hide and omit private or irrelevant details of an internal business process and this way acts as a filter between the internal, private business process and the global business network. Several approaches have been proposed to construct a public process view from a private business process, e.g. [3,5,16,27,35].

Local business process typically use transactions to ensure that they are executed in a robust and reliable way [15,31]. We call such processes transactional processes or transactional workflows [10]. Clearly, the use of transactions

in an internal level process impacts process views generated for that process: transactional features realised by the internal process can be offered in a public process view to provide increased levels of reliability to business partners. For instance, an organisation can offer the same process view with different levels of transactional support at different prices. Conversely, the process provider has to ensure that transactional properties specified for a process view are indeed realisable by the underlying internal transactional process. Existing approaches for constructing process views from internal processes, e.g. [3,5,16,27,35], do not consider transactions and therefore ignore consistency between a public process view and an internal business process from a transactional point of view.

In this paper, we propose the notion of a *transactional process view*, which specifies not only the ordering of public activities but also the offered transactional semantics of different parts of the process. We outline an approach to construct a transactional process view from an internal process model and a transaction specification. The approach extends an existing approach for constructing non-transactional process views from internal block-structured process models [5]. However, the underlying principles can be applied to any of the alternative approaches for constructing non-transactional process views that we discuss in Section 7.

This paper is organised as follows. Section 2 introduces transactional process views by means of an example that is used throughout the paper. Section 3 introduces the various transaction models we consider. Section 4 defines transactional process models, which are process models annotated with transactional properties. We use transactional process models for specifying both process views and private processes. Section 5 discusses how aggregation and customisation can be used to construct a transactional process view for an internal transactional process. We focus especially on how transactional properties of the underlying internal process propagate to the public process view. Section 6 presents a case study of an inter-organisational transactional process from the domain of healthcare. Section 7 discusses related work while Section 8 wraps up with conclusions and further work.

## 2 Overview

By means of an extended example we present an overview of transactional process views. Figure 1 shows the internal business process of a travel agency, in which a client books a trip. The notation is explained in detail in Section 4. In the sales subprocess, the client first selects in parallel a hotel, transport (e.g. flight), and car. Next, the costs are calculated. After the sales subprocess, the client can either cancel the booking or continue by finalising the booking. In parallel, the travel agency prepares the documents and the client receives an invoice and pays. These last two steps are repeated if the client does not pay within a certain time frame. Next, the travel agency sends the documents.

To ensure a reliable execution, a transaction can be used to realise an activity. In Figure 1, activities are annotated with their transactional semantics. The

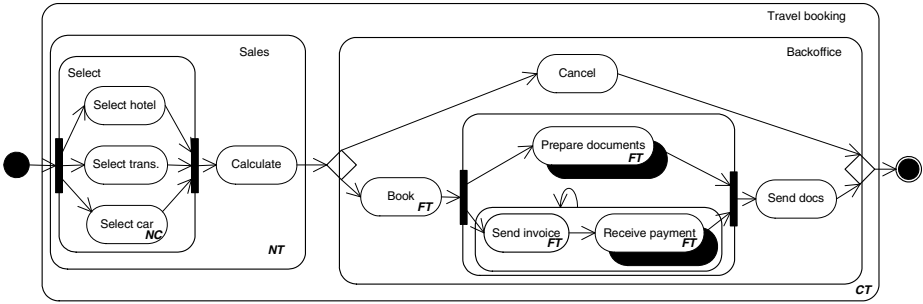


Fig. 1. Travel agency process

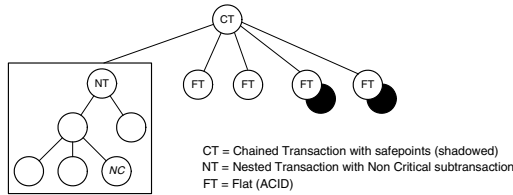


Fig. 2. Layering of transactions for Fig. 1

figure implicitly specifies a layering of transactions that is visualised as a tree in Figure 2. The overall process is a chained transaction (CT) which contains two safe-point activities (indicated with shadows): Prepare documents and Receive payment. A rollback of a chained transaction stops just after the last reached safe-point activities (see Section 3), so in case the chained transaction is rolled back after Send docs has been performed, only Send docs needs to be performed again. Note that the chained transaction also includes the subprocess Backoffice, but this has no separate transactional semantics, so chained transactions can cross the boundaries of subactivities. The Sales subprocess is executed as a nested transaction (NT) in which activity Select car is non-critical (NC): if it fails, the Sales transaction can still complete successfully. The other activities in Sales are critical. There are four flat (FT) transactions, which satisfy the wellknown ACID properties [7]. More explanation on the different transaction models is presented in Section 3.

Figure 3 presents a possible process view for the process in Fig. 1. The process view has been constructed by aggregating two independent parts of the original process. The aggregated parts are shown in the lower half of the figure; each part is linked to a new activity in the process view that hides the activities being aggregated. The remaining activities in the process view like Calculate are realised by private activities that are not shown in the figure. A process view is constructed in a customised way based on input of the process provider [5]. The process provider indicates which activities in the private process must be aggregated, such that their details will be hidden in the process view (dotted activities in Fig. 1). As a consequence of the aggregation requirements, other

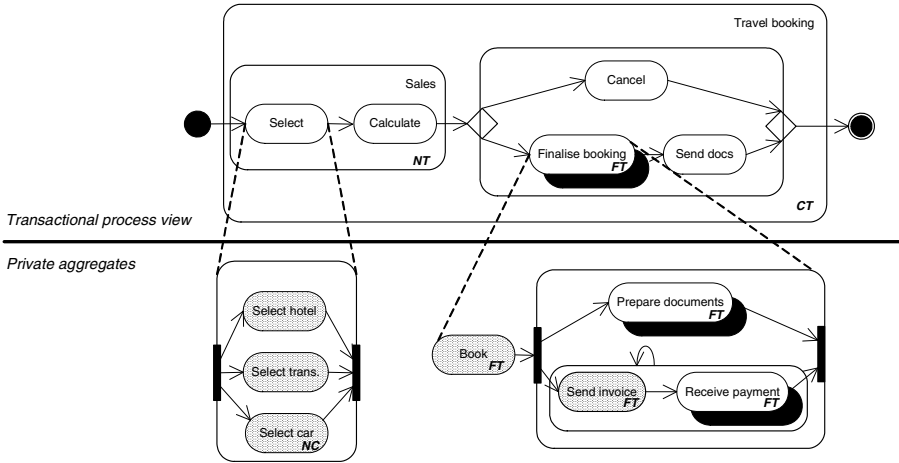


Fig. 3. Example aggregated process view for Fig. 1

activities have to be included in the aggregate to comply with the consistency rules between process view and internal process [5]; in Fig. 1 these are the non-dotted activities contained inside the private aggregates.

In Section 5 we explain in more detail different aspects of the construction of the transactional process view in Fig. 3.

### 3 Transactions

We first discuss the three transactional models we consider in this paper. Next, we explain how they can be combined by layering them.

#### 3.1 Transaction Models

In the paper we consider three mainstream transaction models. An elaborate introduction to transactional models can be found elsewhere [7,31].

A *flat transaction* satisfies the ACID properties. This means that the transaction is guaranteed to execute in its entirety or not at all and that the outcome of operations performed within a flat transaction is the same as if these operations would be performed in a sequence.

A *nested transaction* is composed of sub-transactions in a hierarchical manner. A sub-transaction can be divided into further sub-transactions if necessary, but only the leaf-level sub-transactions really perform database operations while others function as coordinators. A complete nested transaction satisfies the ACID properties. Since sub-transactions within nested transactions in general share data, isolation is relaxed within a nested transaction.

A nested transaction can contain non-critical sub-transactions. A *non-critical* sub-transaction does not need to complete successfully in order for the nested transaction to successfully complete.

**Table 1.** Considered layerings of transaction models (mark ‘x’ means allowed)

	<i>Ancestor</i>		
	Flat	Nested	Chained
Flat	x		x
<i>Descendant</i> Nested		x	x
Chained			x

A *chained transaction* is a long running transaction, for instance a saga [6], that is decomposed into small sub-transactions whose effects can be undone by executing compensating actions. A chained transaction can specify arbitrarily complex behaviour, including sequence, choice, parallelism, and loops (cf. the chained transaction *Travel booking* in Fig. 1 which contains the choice behaviour of subprocess *Backoffice*).

Certain sub-transactions inside a chained transaction can be marked as *safe-points* [11]. Executing a safe-point sub-transaction produces a stable, consistent state of the chained transaction. When a chained transaction is rolled back by executing compensating actions, the rollback stops at the last reached stable state, so the state resulting after executing the last safe-points [11]. To simplify the exposition, we do not consider the specification of compensation actions in the sequel but they can be included without any problem.

### 3.2 Layered Transactions

Different types of transactions can be combined by layering them. In that case, the transaction at a top layer coordinates the transactions at the lower layers. For instance, in the WIDE project [12] a transactional workflow model was developed in which an activity that specifies a chained transaction can contain a subactivity that specifies a nested transaction. In that case, the chained transaction coordinates the nested transaction.

However, not every possible layering is meaningful. A flat transaction cannot coordinate any other type of transaction. A nested transaction can only coordinate nested or flat transactions. A chained transaction can coordinate a flat, chained or nested transaction. Table 1 lists allowed layerings [31]. Other combinations can still be meaningful, but typically require new forms of transaction management that to the best of our knowledge have not been addressed in the literature. Therefore we do not consider these here.

## 4 Transactional Process Models

We introduce transactional process models, which are process models in which activities are annotated with transactional properties. A transactional annotation specifies the transactional semantics of the activity, based on the transactional models introduced in the previous section. Not every annotation is meaningful, however, so we also present constraints that rule out inappropriate annotations.

## 4.1 Definition

A transactional process model [10] specifies how a given set  $A$  of activities are ordered. Activities can be compound, so contain other activities. The used ordering constructs are sequence, choice, parallelism, and loops. Compound activities resemble sub-processes. Non-compound activities are called atomic. In Fig. 1, for instance *Sales* is a compound activity while *Select hotel* is atomic.

Activities can have transactional properties. For this paper, each activity can be annotated with one transaction model, provided the activity type matches the transaction characteristics. For instance, a flat transaction only matches atomic activities, while a chained transaction only matches compound activities. After presenting the formal definition of transactional process models, we formalise the consistency constraints that rule out inappropriate annotations.

Let  $\mathcal{P}$  denote the set of all structured process models and let  $\mathcal{T}$  the set of transaction models. A structured transactional process model  $P \in \mathcal{P}$  is a tuple  $(A, child, type, succ, transType, noncritical, safepoint)$  where:

- $A$  is a set of activities,
- $child : A \times A$  is a predicate that defines the hierarchy relation between activities. We have  $child(a, a')$  if and only if  $a$  is a child (subactivity) of  $a'$ .
- $type : A \rightarrow \{SEQ, PAR, XOR, LOOP, BASIC\}$  is a function that assigns to each activity its type. Type *SEQ* indicates that all children of the activity execute in sequence, *PAR* that they execute in parallel, *XOR* that one of them is executed at a time, and *LOOP* that the children execute zero or more times. We require that each *SEQ*, *XOR*, and *PAR* node has more than one child and that each *LOOP* node has only a single child, which is no *LOOP* node. An activity has type *BASIC* if and only if it is a leaf in the tree, i.e. it has no children.
- $succ : A \rightarrow A$  is partial function that specifies for each child node of a *SEQ* node its successor. We require that if  $(x, y) \in succ$ , then they share the same parent  $z$ , so  $child(x, z)$  and  $child(y, z)$ .
- $transType : A \rightarrow \mathcal{T}$  a function that assigns an activity with its transaction model semantics. For this paper, we let  $\mathcal{T} = \{FT, CT, NT\}$ , where transaction type *FT* stands for the flat (ACID) transaction model, *CT* for the chained transaction model, and *NT* for nested transaction model.
- $noncritical \subseteq A$  is a set of activities that are not critical, i.e. they can fail without jeopardising the successful completion of other activities. Non-critical activities are only used within nested transactions.
- $safepoint \subseteq A$  is a set of activities that are safepoints, i.e., the state of the process is saved and can be recovered. Safepoints are only used within chained transactions.

We use an auxiliary function  $children : A \rightarrow \mathbb{P}(A)$  that defines for each activity its set of child activities. For a leaf activity, this set is empty. The definition of  $children$  makes use of predicate  $child$ :

$$children(a) = \{ a' \in A \mid child(a', a) \}.$$

If  $c \in \text{children}(n)$ , activity  $n$  is parent of  $c$ , written  $\text{parent}(c)$ . By  $\text{children}^+$  and  $\text{children}^*$  we denote the irreflexive-transitive closure and reflexive-transitive closure of  $\text{children}$ , respectively. So  $\text{children}^*(n) = \text{children}^+(n) \cup \{n\}$ . If  $n \in \text{children}^*(n')$ , we say that  $n$  is a descendant of  $n'$  and that  $n'$  is an ancestor of  $n$ . Note that each activity is ancestor and descendant of itself.

To ensure that the *child* predicate indeed arranges activities in a hierarchy represented by a tree structure, we require that each activity has one parent, except one activity  $r$ , which has no parent. Next, we require that  $r$  is ancestor of every activity in  $A$ . These constraints ensure that activity are structured in a tree with root  $r$ . Leaves of the tree are the *BASIC* activities. Internal activities have type *SEQ*, *PAR*, *XOR*, or *LOOP*.

*Notation.* We show structured process models graphically, using a variant of the UML activity diagram notation [28]. We explain the notation using the diagram in Fig. 1. In the diagram, containment indicates hierarchy; for instance, activity *Select car* is child of activity *Select*. Sequential activities have an incoming and outgoing arrow crossing their border, whereas choice and parallel activities have a diamond and bar, respectively, on their border. Within a sequence activity, the ordering relation is specified by means of arrows. Loop activities have no dedicated symbol, but are indicated by drawing a self-edge for the unique child of the loop activity. Safepoints are indicate with shadows. The bold annotations are transactional properties (types), introduced in the previous subsection.

*Auxiliary functions.* To define the construction of process views in Section 5, we will make use of some auxiliary functions on the syntax of structured process models [5]. For a set  $X$  of activities, the least common ancestor (lca) of  $X$ , denoted  $\text{lca}(X)$  is the activity  $x$  such that  $x$  is ancestor of each activity in  $X$ , and every other activity  $y$  that is ancestor of each activity in  $X$ , is ancestor of  $x$ :

- $X \subseteq \text{children}^*(x)$ , and
- For every  $y \in A$  such that  $X \subseteq \text{children}^*(y)$ , we have that  $x \in \text{children}^*(y)$ .

Since activities are arranged in a tree, every set of activities has a unique least common ancestor. For example, in Fig. 1 the lca of *Select trans.* and *Select car* is *Select*, whereas the lca of *Cancel* and *Book* is *Backoffice*. Note that the lca of a single activity is the activity itself, i.e.  $\text{lca}(\{x\}) = x$ .

The before relation  $<$  denotes temporal ordering. Given two activities  $a, a' \in A$ , we have  $a$  before  $a'$ , written  $a < a'$ , if and only if

- activity  $l = \text{lca}(\{a, a'\})$  has type *SEQ*, and
- for the children  $c_a, c_{a'}$  of  $l$  such that  $a$  is descendant of  $c_a$  and  $a'$  is descendant of  $c_{a'}$ , we have that  $c_{a'}$  is a successor of  $c_a$  so  $c_a \text{succ}^* c_{a'}$ .

For example, in Fig. 1 we have *Book*  $<$  *Prepare documents*.

## 4.2 Transaction Type Constraints

A process model  $P$  uses a function  $transType$  that labels each activity with a transaction type. However, not every labelling is meaningful, since a transaction type has to match the kind of activity. We specify next some consistency constraints that rule out inappropriate labellings.

First we introduce an auxiliary definition. The *scope* of an activity  $a$ , denoted  $scope(a)$ , is the most nested activity that is strict ancestor of  $a$ , so  $scope(a) \neq a$ , and that has a transactional type. For instance, in Fig. 1 the scope of **Select car** is **Select** whereas the scope of **Book** is **Backoffice**.

Let  $a \in A$  be an activity from  $P$ . The following constraints are valid for  $a$ :

1. a flat transaction has no children:  $transType(a) = FT$  implies  $children(a) = \emptyset$ .
2. a nested or chained transaction has children:  $transType(a) = CT$  or  $transType(a) = NT$  implies  $children(a) \neq \emptyset$ .
3. a non-critical activity is only used within the scope of a nested transaction:  $a \in noncritical$  implies  $transType(scope(a)) = NT$ .
4. a safe point is only used within the scope of a chained transaction:  $a \in safe point$  implies  $transType(scope(a)) = CT$ .
5. a chained transaction is not used within a nested transaction:  $transType(a) = CT$  implies there is no ancestor  $a'$  of  $a$  such that  $transType(a') = NT$ .

Most constraints are self explanatory. The last constraint formalises the layering constraint stated in Section 3.2.

## 5 Aggregation and Customisation

First, we discuss how transactional aggregates are constructed for transactional process views, by extending an earlier defined aggregation procedure for non-transactional process views [5]. Next, we focus on deriving the right transactional properties in the process views for the constructed transactional aggregates. This way, we can define the proper derivation of a transactional process view from a transactional process. Finally, we outline how aggregated transactional process view can be customised by hiding activities from the process view [5].

### 5.1 Constructing Transactional Aggregates

An aggregate  $agg$  is a set of activities from the process model that is represented in the process view by a single activity  $a_{agg}$ , i.e. activity  $a_{agg}$  hides the activities contained in the aggregate  $agg$ . Activity  $a_{agg}$  is atomic and has no child activities in the process view. Therefore, its transactional type can neither be chained ( $CT$ ) nor nested ( $NT$ ).

In the approach of [5] for constructing non-transactional process views, the user must specify which set of activities have to be aggregated. However, the aggregate might need to contain some additional activities as well, in order to get a process view that is consistent with the underlying process model. The view



and the process model are consistent if the orderings of the underlying process model are respected by the view and no additional orderings are introduced in the view. We have defined declarative rules and an equivalent operational algorithm for constructing a minimal aggregate set for an input set of activities, such that the resulting process view is consistent with the underlying process [5].

In a transactional process view, activity  $a_{agg}$  may hide transactional activities. In that case, the transaction type of  $a_{agg}$  becomes flat ( $FT$ ). There are two specific cases. The first case is that the aggregate  $agg$  equals a chained or nested transaction  $x$  with all its descendants, so  $agg = children^*(x)$  and either  $transType(x) = CT$  or  $transType = NT$ . Predicate  $completeCompound$  captures formally when  $agg$  hides a complete chained or nested transaction:

$$completeCompound(agg) = \exists x \in agg : children^*(x) = agg \\ \wedge (transType(x) = CT \vee transType(x) = NT)$$

In that case  $a_{agg}$  hides the complete chained or nested transaction and  $a_{agg}$  becomes a flat transaction.

In the second case, aggregate  $agg$  contains only flat transactions, so predicate  $completeFT(agg)$  is true, where

$$completeFT(agg) = \forall x \in agg : transType(x) = FT.$$

Since  $agg$  only contains flat transactions,  $a_{agg}$  becomes a flat transaction as well.

In all other cases, the transaction type of  $a_{agg}$  is undefined. For instance, activity **Select** in Fig. 3 has an undefined transaction type, since the corresponding aggregate contains only part of the nested transaction **Sales**, while **Finalise booking** has transaction type  $FT$  since the corresponding aggregate contains only flat transactions.

We formalise these rules to determine the transaction type of a new activity in the process view, by defining partial function  $getTT(a, agg)$ , which expects an activity  $a$  that represents an aggregate  $agg$  (set of activities) and returns the transaction type of  $a$ , if defined:

$$getTT(a, agg) = \begin{cases} FT & , \text{if } completeCompound(agg) \text{ or } completeFT(agg) \\ undefined & , \text{otherwise} \end{cases}$$

## 5.2 Properties of Transactional Aggregates

Let  $a_{agg}$  be a new activity in the process view that hides an aggregate set  $agg$  of activities of the underlying internal process. If  $a_{agg}$  is in the scope of a nested or chained transaction, then  $a_{agg}$  can have special transactional properties; for instance  $a_{agg}$  can become a safe point if its scope is a chained transaction. In this subsection, we study how these transactional properties for  $a_{agg}$  are derived from the underlying aggregate.

First, we lift the notion of scope to sets of activities. The scope of a set of activities  $X$  is the most nested activity  $a$  that contains all activities in  $X$  and that has a transactional type.

**Nested Transactions: Dealing with Non-critical Activities.** Consider an aggregate  $agg$  whose scope is a nested transaction. If  $agg$  contains non-critical activities, then  $a_{agg}$  is non-critical if and only if *all* aggregated activities are non-critical. In that case, if all aggregated activities fail at the private level, the aggregate can fail at the public level and the nested transaction can still complete successfully. In all other cases, an aggregated activity in  $agg$  can fail that is critical. Then the abstract activity  $a_{agg}$  and the nested transaction have to fail as well. In the process view in Fig. 3 activity **Select** is critical since the corresponding aggregate contains critical activities **Select hotel** and **Select trans**.

**Chained Transactions: Dealing with Safepoints.** Consider an aggregate  $agg$  whose scope is a chained transaction, so  $a_{agg}$  is in the scope of the chained transaction in the process view. Some activities in the aggregate can be safepoints. Now the question is whether  $a_{agg}$  should become a safepoint in the process view. If safepoints in  $agg$  are not present in the process view, an inconsistency between process view and private process can arise if a rollback is performed for  $a_{agg}$ . Then  $a_{agg}$  in the process view is completely rolled back, but in the private process the rollback stops already at the last reached safepoints. Clearly, then there is an inconsistency between the execution state of the process view and the state of internal underlying process.

To avoid these inconsistencies, we require that an aggregate  $agg$  is only a safepoint if and only if all “last” activities it contains are safepoints, where “last” means that  $a$  is not followed by other activities in the aggregate, and in addition all safepoints it contains are last, so the aggregate does not contain intermediate safepoints that are followed by other safepoints. If the aggregate would contain intermediate safepoints, an inconsistency could arise between the state of the transactional process view and the state of the underlying transactional process, as explained above.

Formally, an activity  $a$  in an aggregate  $agg$  is last if and only if it has no successor activity in the set of aggregated activities:

$$last(a, agg) \Leftrightarrow \text{for all } x \in agg : a \neq x \Rightarrow a \not\prec x.$$

Predicate  $lastSP$  is true if and only if all last activities in the aggregate  $agg$  are safepoints *and* all safepoints are last activities.

$$lastSP(agg) \Leftrightarrow \text{for all } a \in agg : last(a, agg) \Leftrightarrow a \in safepoint$$

The aggregate in Fig. 3 that corresponds to **Finalise booking** satisfies  $lastSP$ .

It may well be that the aggregation procedure described in 5 constructs an aggregate that does contain intermediate safepoints, which is undesirable. To address such cases, the aggregation procedure can be changed by returning multiple subaggregates and letting safepoints determine the boundary of each subaggregate. The union of the returned subaggregates then comprises the complete aggregate. In each subaggregate, safepoints are the last activities. This is only feasible if the subaggregates are ordered sequentially. For instance, consider a process that consists of a sequence of activities  $A, B, C, D, E$ , and let  $C$

and  $E$  be safe-points. If  $B$  and  $D$  have to be aggregated, the aggregation procedure of [5] constructs an aggregate containing  $B$ ,  $C$  and  $D$ . The changed aggregation procedure could return one subaggregate for  $B$  and  $C$ , and one for  $D$  and  $E$ , and the activities representing both subaggregates in the process view become safe-points. Due to space limitations, we do not further elaborate such an extension of the aggregation algorithm.

### 5.3 Generating Transactional Process Views with Aggregates

Above, we have outlined how transactional properties of aggregated activities at the internal level propagate to the activities in the process view that represent the aggregated activities. Now we define a function  $gen : (\mathcal{P} \times A) \rightarrow \mathcal{P}$  that generates from a given structured process model and an aggregate the resulting transactional process view, which is again a structured process model. If there are multiple aggregates, the function can be repeatedly applied. The function extends an earlier defined function for generating non-transactional process views [5] with transactional elements.

If  $agg$  is the constructed aggregate for process model  $P$  with activity set  $A$ , so  $agg \subseteq A$ , then the process model  $P' = gen(P, agg)$  is constructed by replacing  $agg$  with a new activity  $a_{agg} \notin A$  that does not get any children in the process view  $P'$ .

Now the problem is that the new activity  $a_{agg}$  needs to be attached as child to some activity  $A \setminus agg$ , i.e., some activity  $l \in A \setminus agg$  has to act as parent of  $a_{agg}$  in the process view  $P'$ . Let  $l$  be the lowest activity in  $A \setminus agg$  that is ancestor (in  $P$ ) of all activities in  $agg$ . The construction procedure in [5] ensures that activity  $l$  exists and is unique. Therefore,  $l$  can be the unique parent of  $a_{agg}$  in  $P'$ .

Formally,  $P' = (A', child', type', succ', transType', noncritical', safepoint')$  where

- $A' = A \setminus agg \cup \{a_{agg}\}$
- $child' = (child \cap (A' \times A')) \cup \{(a_{agg}, l)\}$
- $type' = type \cap (A' \times \{SEQ, PAR, XOR, BASIC\}) \cup \{(a_{agg}, BASIC)\}$
- $succ' = (succ \cap (A' \times A')) \cup \{(a_{agg}, y) \mid x \in agg \wedge y \notin agg \wedge (x, y) \in succ\} \cup \{(x, a_{agg}) \mid x \notin agg \wedge y \in agg \wedge (x, y) \in succ\}$
- $transType' = transType \cap (A' \times \{FT, CT, NT, ET\}) \cup \{a_{agg}, getTT(a_{agg})\}$
- $noncritical' = \begin{cases} (noncritical \setminus agg) \cup \{a_{agg}\} & , \text{if } a_{agg} \subseteq noncritical \\ noncritical \setminus agg & , \text{otherwise} \end{cases}$
- $safepoint' = \begin{cases} (safepoint \setminus agg) \cup \{a_{agg}\} & , \text{if } lastSP(agg) \\ safepoint \setminus agg & , \text{otherwise} \end{cases}$

The definition of  $transType'$  makes use of function  $getTT$  that identifies the transaction type of  $a_{agg}$  based on the transactional properties of  $agg$ ; see Section 5.1.

## 5.4 Customisation

The consumer for which the process provider will execute its process can create a customised process view that only contains the “relevant” activities, where a process consumer can determine himself which activities are relevant [5]. All the other activities are “noise” that should be filtered.

To create a customised process view, the process consumer first selects the relevant activities that have to be in the view. Next, all unselected activities are replaced with internal activities, which are executed at the private level but invisible at the public process view level. Each internal activity gets label  $\tau$ , so internal activities cannot be distinguished from each other from the consumer point of view. Therefore, they can be grouped as much as possible. For instance, a sequence of internal activities is grouped into a single internal activity [5].

Let  $P$  be an (aggregated) process view and let  $I$  be a set of activities from  $P$  that the consumer wishes to monitor in the customised process view  $P' = (A', child', type', succ', transType', noncritical', safepoint')$ . The non-transactional elements of the tuple, i.e.  $A'$ ,  $child'$ ,  $type'$  and  $succ'$ , have been defined in our previous work [5]. We define the transactional elements of the tuple by keeping only the transactional properties related to activities in  $I$ :

- $transType' = transType \cap (I \times \{FT, CT, NT, ET\})$
- $noncritical' = noncritical \cap I$
- $safepoint' = safepoint \cap I$

Since a  $\tau$  action is hidden, we assume its transactional properties are hidden too. For instance, if in the customisation process safepoint activities are relabelled  $\tau$ , they are no longer safepoints in the customised view: the safepoint information is simply invisible at the public level, though the safepoint still exists at the private level. By similar reasoning, if non-critical activities in the scope of a nested transaction become non-critical, the non-critical activities are lost in the customised view.

## 6 Case Study

As a case study, we have applied the approach to an inter-organisational business process from the healthcare domain. Processes in the healthcare domain are often very complex. They contain numerous subprocesses and activities covering multiple departments in a hospital and/or hospitals and/or other organisations. As healthcare processes deal with patients, reliability of such processes is even more important than it is in many other complex business processes. Using transactional process views, quality aspects of process execution such as reliability aspects can be captured and monitored.

The case study is based on a teleradiology process for the acquisition and interpretation of medical scans of patients (see Fig. 4). The process has been designed in close collaboration with an industrial partner [30] that can offer technology support for certain parts of this process. The complete process is

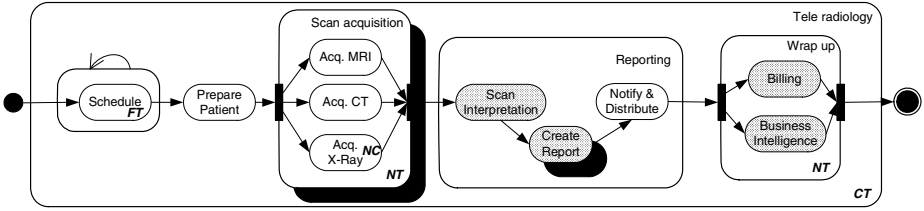


Fig. 4. Teleradiology process

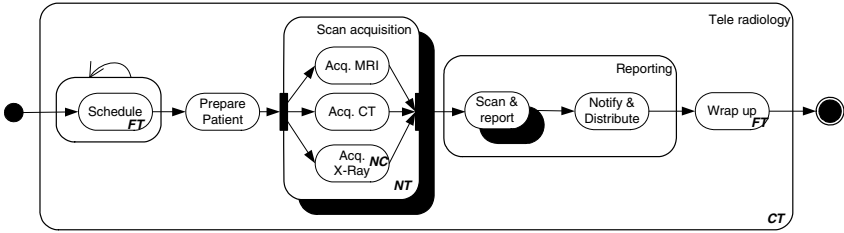


Fig. 5. Process view for teleradiology process in Fig. 4

offered by for instance the radiology department of a hospital or by a specialised radiology clinic.

A process designer at the process provider side (e.g. a specialised radiology clinic) has designed the internal process specification shown in Fig. 4. The process starts by scheduling the patient. A patient can reschedule the appointments if necessary. At the scheduled time, the required scans are acquired (Scan acquisition), after which an interpretation report is created and distributed to the client (Reporting). The process ends after billing and business intelligence have been performed (Wrap up). The process results in a report that a medical specialist, who ordered the scan, can use to base his diagnosis and treatment on. An extensive description of the process is provided elsewhere [30].

The provider can be a specialised radiology clinic that wishes to hide certain parts of the internal process for business reasons. First, the provider wishes to aggregate the first part of the Reporting subprocess into a new activity Scan & report in Fig. 5. Note that Reporting and its subactivities are part of the top-level chained transaction. The generation of a report is safepoint at the internal level that is propagated to the process view. If for instance the compound transaction is rolled back immediately after executing Notify & distribute, the internal process rolls back to the state reached after Create Report which is consistent with the state in the process view reached after completing Scan & report, so the resulting state in the internal process is consistent with the resulting state in the transactional process view.

Since billing and business intelligence are private parts of the underlying process, the provider also aggregates these two activities into activity Wrap up in the process view. Activity Wrap up in Fig. 5 is a flat transaction in the process

view, since it has no children and the underlying aggregate contains the complete nested transaction for compound activity *Wrap up* in Fig. 4.

Note that many other process views can be constructed for the process in Fig. 4. For instance, if the consumer is only interested in reporting and billing, a customised process view can be created in which all activities related to patient scheduling, preparation and scanning are hidden by having them grouped inside a single internal activity. Due to space limitations, this process view is not shown.

## 7 Related Work

Process views have originally been proposed for use in an inter-organisational context. One of the earliest works is by Liu and Shen [16], who focus on deriving a non-transactional process view from a given structured process definition in the context of collaborative processes that operate within virtual enterprises. Such processes span the boundaries of multiple organisations. Process views are used to align the collaborative process with the local private business processes.

Chiu et al. [4] use non-transactional process views to support interoperability of multiple workflows across organisations. They present a meta model and an interoperation model for workflow views, consisting of communication scenarios between these views, and a set of interoperation parameters. They show how the approach can be realised using web services and XML technology, but they do not consider the use of transactions.

Next, there are approaches that use views for enabling inter-organisational workflow cooperation [3,26,27,34,36]. The approach of Chebbi et al. [3] consists of three main steps: workflow advertisement, workflow interconnection, and workflow cooperation. The main focus of the paper is on the second step. They present reduction rules to derive from an internal process model an abstract process model which only contains tasks that cooperate with partner workflows outside the organisation. On this public process, partner-specific views can be defined that are linked with an access contract. Related to this work, Tahamtan and Eder [26,27] focus on the construction of process views in the context of federated choreographies. Zhao et al. [34,36] use visibility constraints on internal process models to derive partner-specific workflow views. Each partner can combine the workflow views of its partner with its internal process into what Zhao et al. call a relative workflow model. Next, they discuss how an organisation can use a relative workflow model to track the progress of its indirect partners, e.g. how a consumer can track the progress of the process of the provider of the provider. None of these works combines process views with transactions.

Ye et al. [33] study the analysis of atomicity properties for a set of interacting public process views that use atomicity spheres [13,24]. Part of the approach considers the use of axioms from process algebra to construct a public process view from a private process while preserving atomicity. However, the constructed process view is fixed: its construction cannot be influenced by a user, whereas the transactional process views generated in this paper are constructed based on user input, specifying which activities are to be aggregated or omitted. Atomicity spheres can be easily incorporated in the formalisation of transactional

process models. However, some atomicity spheres are incorrect, since they do not terminate properly. The transactional process models defined in this paper are by construction correct. Incorporating atomicity spheres means we have to introduce the concept of incorrect process view, which considerably complicates the approach.

Schulz and Orlowska [25] focus on architectural support for workflow (process) views. They look into possible interactions that can occur between workflow views and between workflow views and private workflows. Next, they analyze how such interactions can be supported by describing different ways of coupling workflow views and private workflows. Finally, they define a cross-organisational workflow architecture that supports the execution of workflow views. However, they do not consider the use of transactions.

Some existing industrial standards, notably WS-BPEL [1] and BPMN [32], distinguish between an abstract (public) and a concrete (private) process. The abstract process is a nondeterministic protocol describing possible interactions, whereas a concrete process is actually executable by a process engine. This distinction between abstract and concrete process is similar to the one made in this paper between a process view and its underlying internal process. Unlike our approach, these standards do not define any consistency constraints between these different process levels, nor do they address customisation.

Some researchers have studied process views in the context of WS-BPEL [1]. König et al. [14] propose a new WS-BPEL profile for easy checking of compatibility between an abstract BPEL process and an executable BPEL process. The profile uses consistency rules that enforce compatibility. Zhao et al. [35] discuss the construction and implementation of process views in WS-BPEL. They develop a framework to support different abstraction and concretisation operators for WS-BPEL processes. The process view transformation is under supervision of a set of rules on structural consistency and validity.

Though WS-BPEL does not provide transactional mechanisms, it is complemented by standards like WS-Coordination [21] and WS-Transaction [20] that do provide transactional mechanisms. For an elaborate overview of these different web service transaction standards we refer to Wang et al. [31]. We are unaware of any research in which these web service transaction standards are used in combination with WS-BPEL abstract processes.

Process views are also used in intra-organisational business process management to generate user-specific process views. Liu and Shen [17] discuss the construction of personalised process views, that are specific to a organisational user (actor) of the process, based on relationships among tasks and roles that are specified in role-based access control systems. Bobrik et al. [2] also study the creation of such personalised views. They identify parameterisable operations that can be flexibly composed in order to reduce or aggregate process information in the desired way, based on the specific needs of the respective target application. However, small inconsistencies are tolerated in favour of a more adequate visualisation. Motahari [19] et al. focus on the use of process views to

abstract and visualise process executions that are captured in process logs. Also in all these approaches, transactions are not considered.

Several researchers have proposed combinations of workflow management or business process management and transaction technology [8,10,24]. The focus of these works is on the interaction of process management technology and transaction management technology. However, these works do not consider the use and construction of transactional process views. For instance, in the CrossFlow project [8,29], cross-organisational processes with transactional semantics are defined on the external level, which resembles with process views. But the relation between external level and internal level is not explicitly defined: external level models are created ad-hoc and there is no consistency relation between an internal level and an external level process in CrossFlow. This paper complements the work done in CrossFlow by defining a structured and systematic approach for constructing transactional process views that are consistent with the underlying transactional processes.

## 8 Conclusion

We have introduced a well-structured approach that allows a process provider to construct a transactional process view from a transactional process model. In defining transactional process models, we have considered the mainstream transactional models, including nested transactions and chained transactions, but the approach can be easily extended to other transaction models with a comparable complexity. Using the approach, process views can be constructed in a flexible way, since the actual process view that is generated is driven by input of the process consumer and provider about the activities that need to be hidden or omitted. Using this construction approach, it is possible to specify robust and reliable process behaviour at the public external level.

In today's business, dynamic cooperations between autonomous organisations becomes increasingly important. In the past, organisations cooperated with each other in rather static networks with a long life span. To comply with current market settings, however, organisations have to shift their priority to flexibility and ability to change if they want to survive [22]. As a consequence, dynamic cooperation between organisations is often required to meet market demands [9]. Collaborations have a short lifespan and are increasingly with a lot of different business partners with different levels of trust. By using transaction management technology in combination with process management technology, organisations can collaborate with each other in a trustworthy yet fine-grained way. Our approach can aid in establishing such dynamic yet trustworthy collaborations in an efficient way.

There are several directions for further work. First, the approach can be extended to deal with process models that are not block-structured. However, we expect this complicates the integration with transactions, since process models that are not block-structured can easily contain modelling errors that lead to deadlocks at run time. Next, we plan to investigate technology to support the



run-time execution of transactional process views. Various proposals for combining transaction and process management have been made [23,24] that can be used as starting point.

## References

1. Anders, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. In: Standards proposal by BEA Systems. International Business Machines Corporation, Microsoft Corporation. SAP AG, Siebel Systems (2002)
2. 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)
3. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.* 56(2), 139–173 (2006)
4. Chiu, D., Cheung, S., Till, S., Karlapalem, K., Li, Q., Kafeza, E.: Workflow view driven cross-organizational interoperability in a web service environment. *Inf. Tech. and Management* 5(3–4), 221–250 (2004)
5. Eshuis, R., Grefen, P.: Constructing customized process views. *Data and Knowledge Engineering* 64(2), 419–438 (2008)
6. Garcia-Molina, H., Salem, K.: Sagas. In: Dayal, U., Traiger, I.L. (eds.) SIGMOD Conference, pp. 249–259. ACM Press (1987)
7. Gray, J., Reuter, A.: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann (1993)
8. Grefen, P., Aberer, K., Hoffner, Y., Ludwig, H.: CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science and Engineering* 15(5), 277–290 (2000)
9. Grefen, P., Mehandjiev, N., Kouvas, G., Weichhart, G., Eshuis, R.: Dynamic business network process management in instant virtual enterprises. *Computers in Industry* 60(2), 86–103 (2009)
10. Grefen, P., Vonk, J.: A taxonomy of transactional workflow support. *Int. J. Cooperative Inf. Syst.* 15(1), 87–118 (2006)
11. Grefen, P., Vonk, J., Apers, P.: Global transaction support for workflow management systems: from formal specification to practical implementation. *The VLDB Journal* 10(4), 316–333 (2001)
12. Grefen, P., Vonk, J., Boertjes, E., Apers, P.M.G.: Two-layer Transaction Management for Workflow Management Applications. In: Hameurlain, A., Tjoa, A.M. (eds.) DEXA 1997. LNCS, vol. 1308, pp. 430–439. Springer, Heidelberg (1997)
13. Hagen, C., Alonso, G.: Exception handling in workflow management systems. *IEEE Trans. Software Eng.* 26(10), 943–958 (2000)
14. König, D., Lohmann, N., Moser, S., Stahl, C., Wolf, K.: Extending the compatibility notion for abstract WS-BPEL processes. In: Huai, J., Chen, R., Hon, H.-W., Liu, Y., Ma, W.-Y., Tomkins, A., Zhang, X. (eds.) WWW 2008, pp. 785–794. ACM (2008)
15. Leymann, F., Roller, D.: *Production Workflow – Concepts and Techniques*. Prentice-Hall (2000)
16. Liu, D.-R., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. *Inf. Syst.* 28(6), 505–532 (2003)

17. Liu, D.-R., Shen, M.: Business-to-business workflow interoperation based on process-views. *Decision Support Systems* 38(3), 399–419 (2004)
18. Mehandjiev, N., Grefen, P.: *Dynamic Business Process Formation for Instant Virtual Enterprises*. Springer, Heidelberg (2010)
19. Motahari Nezhad, H.R., Benatallah, B., Casati, F., Saint-Paul, R.: From business processes to process spaces. *IEEE Internet Computing* 15, 22–30 (2011)
20. OASIS. Web services atomic transaction 1.1 (2007)
21. OASIS. Web services coordination 1.1 (2007)
22. Pieper, R., Kouwenhoven, V., Hamminga, S.: *Beyond the hype: E-business strategy in leading European companies*. Van Haren Publishing (2002)
23. Schäfer, M., Dolog, P., Nejdil, W.: An environment for flexible advanced compensations of web service transactions. *TWEB* 2(2) (2008)
24. Schuldt, H., Alonso, G., Beeri, C., Schek, H.-J.: Atomicity and isolation for transactional processes. *ACM Trans. Database Syst.* 27(1), 63–116 (2002)
25. Schulz, K., Orłowska, M.: Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.* 51(1), 109–147 (2004)
26. Tahamtan, A., Eder, J.: Privacy preservation through process views. In: *IEEE AINA Workshops 2010*, pp. 1100–1107. IEEE Computer Society Press (2010)
27. Tahamtan, A., Eder, J.: View driven interorganizational workflows. *International Journal of Intelligent Information and Database Systems* (to appear, 2011)
28. UML Revision Taskforce. *UML 2.3 Superstructure Specification*. Object Management Group, 2010. OMG Document Number formal (May 5, 2010)
29. Vonk, J., Grefen, P.: Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases* 14(2), 137–172 (2003)
30. Vonk, J., Wang, T., Grefen, P., Swennenhuis, M.: An analysis of contractual and transactional aspects of a teleradiology process. *Beta Working Paper Series, WP 263*, Eindhoven University of Technology (2011)
31. Wang, T., Vonk, J., Kratz, B., Grefen, P.: A survey on the history of transaction management: from flat to grid transactions. *Distributed and Parallel Databases* 23(3), 235–270 (2008)
32. White, S., et al.: *Business Process Modeling Notation (BPMN) Specification, Version 1.1*. Object Management Group (2008), <http://www.bpmn.org>
33. Ye, C., Cheung, S.-C., Chan, W.K., Xu, C.: Atomicity analysis of service composition across organizations. *IEEE Trans. Software Eng.* 35(1), 2–28 (2009)
34. Zhao, X., Liu, C.: Tracking Over Collaborative Business Processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) *BPM 2006*. LNCS, vol. 4102, pp. 33–48. Springer, Heidelberg (2006)
35. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M., Yongchareon, S.: Implementing process views in the web service environment. *World Wide Web* 14(1), 27–52 (2011)
36. Zhao, X., Liu, C., Yang, Y.: An Organisational Perspective on Collaborative Business Processes. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) *BPM 2005*. LNCS, vol. 3649, pp. 17–31. Springer, Heidelberg (2005)

# Edit Distance-Based Pattern Support Assessment of Orchestration Languages

Jörg Lenhard, Andreas Schönberger, and Guido Wirtz

Distributed and Mobile Systems Group, University of Bamberg, Germany  
{joerg.lenhard, andreas.schoenberger, guido.wirtz}@uni-bamberg.de

**Abstract.** Orchestration languages are of paramount importance when implementing business processes based on services. Several languages for specifying Web Services-based orchestrations are available today. Examples are the Web Services Business Process Execution Language or Windows Workflow. Patterns for process-aware information systems have frequently been used to assess such languages. Various studies discuss the degree of support such languages provide for certain sets of patterns. However, the traditional trivalent support measure is limited in terms of granularity and selectivity. This paper proposes an edit distance complexity measure that allows to overcome these issues. The applicability of this measure is demonstrated by an analysis of several orchestration languages using four different pattern catalogs.

**Keywords:** SOA, Pattern, Edit Distance, Orchestration, BPEL, WF.

## 1 Introduction

Today, *service-oriented architectures* (SOAs) form the primary means to create flexible, interoperable and cooperative information systems. In SOAs, business processes are implemented as *composite services* [15]. Such composite services combine calls to existing services by defining control- and data-flow dependencies between the different service invocations. *Orchestrations* have been introduced in [16] as executable process definitions that use Web Services as primitives for interactions.

Several languages for implementing Web Services-based orchestrations are available, such as the *Web Services Business Process Execution Language* (BPEL) [13] or *Windows Workflow* (WF) [3]. Traditional comparisons of such languages predominantly use patterns to assess their expressiveness. *Workflow control-flow patterns* [23] and *service interaction patterns* [2] are of outstanding importance. The more patterns a language supports and the higher the degree of support it provides is, the more expressive it is.

The traditional pattern support measure offers three possible values, i.e., *direct* (+), *partial* (+/-) or *no direct support* (-) for a pattern [23, p. 50]. These values are calculated based on the number of language constructs (e.g. decision or loop constructs) needed for implementing a pattern. The constructs needed therefore are used to represent how directly a pattern is supported. If a pattern can be

implemented using a single construct, a solution provides direct support. If a combination of two constructs needs to be used, the result is partial support. In all other cases, there is no direct support. This support measure has been used in various studies [1, 5, 6, 8, 17–20, 23–25, 27]. Unfortunately, it comes with a number of problems, so that the calculation of the degree of support provided even is left out in some studies (cf. [14]):

1. Although discussed by some authors, the support measure in its present form does not reveal whether a pattern can be implemented in a language *at all*. Solutions using more than two constructs may exist. However, these are rated as offering *no direct support*. This low level of granularity results in a limited degree of selectivity provided by the support measure. The notion of *no direct support* can easily be misinterpreted to believe that a pattern cannot be implemented in a language at all.
2. Usually, the degree of support is determined by a *conceptual analysis of the set of core constructs* of a language specification. Due to this, the complexity of pattern solutions and the effort required by the implementer of a pattern is not truly captured. An executable implementation of a pattern might require the use of more constructs and other aspects that do not belong to this set, such as variable or correlation set definitions, which an analysis of a specification does not reveal.
3. Only the *use*, but not the *configuration* of constructs is considered in the traditional support measure. This can lead to an inaccurate classification. Assume the default configuration of a construct provides a complete solution to a pattern in one language. In another language, also a single construct is sufficient, but this construct must be configured in several non-trivial steps. Clearly, the first language supports the pattern more directly and the solution is less costly. Nevertheless, both languages achieve the rating of *direct support*.

The contribution of this paper is the definition of a more accurate support measure that offers higher granularity and selectivity. It hence contributes to alleviating the above deficiencies. This measure is based on the *edit distance* concept [11] and operationalized for pattern-based analysis of orchestration languages. The basic idea is to count the number of steps taken by the user of a language when implementing a pattern, as this is more suitable to capture the effort required by the user. The implementation steps we consider are semantically meaningful units of change, specific for orchestration languages. The proposed measure allows to determine the cost associated with the implementation of a pattern in a certain language and unveils whether or not a pattern can be implemented at all.

To verify the applicability of this support measure, we furthermore provide an extensive analysis of the orchestration languages BPEL 2.0 [13], its implementation in the OpenESB BPEL Service Engine (Sun BPEL), and WF in revision 4 [3]. We analyze these languages for their support of four pattern catalogs, the *workflow control-flow patterns* [19, 23], the *service interaction patterns* [2], the *time patterns* [8] and the *patterns for changes in predefined regions* [24]. These catalogs have been used in various studies, e.g. [4, 6, 12, 14, 25, 27], and describe aspects that are of paramount importance for realistic processes. In total,

they consist of 70 different patterns. We calculate the degree of support with our proposed support measure, the traditional trivalent measure, and compare the results. Although partly based on preceding analyses [2, 19, 25, 27], the assessment of the support provided by the languages in the specified revisions for these pattern catalogs is new.

This paper builds upon previous work on assessing pattern support. In [10], we have introduced a uniform method for checking whether a pattern implementation is valid and we have put forward the idea of using an edit distance-based support measure. The approach for assessing the validity of candidate pattern implementations has been reused for this study and is not further discussed. Concerning the edit distance-based support measure, we provide an operationalization for orchestration languages that enables comparability across orchestration languages and we provide an extensive evaluation of our measure in terms of selectivity and applicability by assessing the above four pattern catalogs.

The paper is structured as follows: In the following section we review related work and discuss relevant pattern catalogs and pattern-based analyses. In Sect. 3 we define and demonstrate the use of the edit distance support measure. Thereafter, we present the results of the analysis and discuss their implications. Finally, Sect. 4 concludes the paper with remarks on future work.

## 2 Related Work

Research on applying patterns for the design and implementation of SOAs and process-aware information systems is very popular, e.g. [12, 28]. However, the work here focuses on applying patterns to measure the effort for creating process models, in particular orchestration models. Therefore, we first identify relevant pattern catalogs and orchestration languages. Then, we discuss existing studies that evaluate the expressiveness of process languages using patterns and finally focus on metrics for measuring pattern support.

There are numerous approaches that present pattern catalogs for assessing process-aware information systems. The *workflow patterns initiative* started this movement with the *control-flow patterns* [19, 23]. They also cover *data patterns* [18], *resource patterns* [17] and *exception handling patterns* [20]. The first pattern catalog that addresses specific requirements of service-oriented processes with a number of important interaction scenarios is the *service interaction patterns* catalog [2]. Similarly, [22] characterizes business functions that are frequently needed in processes as *activity patterns*. [1] defines correlation mechanisms that are used in service interactions in the form of *correlation patterns*. The means for creating process instances are covered by *process instantiation patterns* [5]. Constructs that allow for flexibility of processes in the face of changes to their structure are captured as *change patterns* in [24]. Common time constraint mechanisms are proposed in [8] in the form of *time patterns*.

In the area of orchestration languages, BPEL [13], currently available in revision 2.0, is a widely accepted standard. It is a mainly block-structured language specification that has been implemented by several vendors. As is common practice in pattern-based analyses [17-19, 23], an implementation of this language

needs to be treated as a separate language. One such implementation is the OpenESB BPEL Service Engine<sup>1</sup> WF [3] is a proprietary orchestration language maintained by Microsoft as a part of the .NET framework. Since April 2010, it is available in revision 4. Compared to previous versions, it has undergone significant changes. WF contains block-structured and graph-oriented elements.

Almost all of the studies that introduce pattern catalogs also analyze the degree of support that varying languages and systems provide for the patterns in focus [1, 2, 5, 8, 17–20, 23, 24]. The support provided by BPEL 1.1 and in some cases also by one of its implementations is discussed for the control-flow, data, resource, service interaction, and correlation patterns [2, 17–20, 23]. The process instantiation patterns and correlation patterns [1, 5] are evaluated for BPEL 2.0. Further pattern-based analyses of BPEL 1.1 and comparisons to other Web Services composition languages can be found in [6, 25]. For WF, only a single study that assesses its support for control-flow patterns in its revision 3.5 can be found so far [27]. All these studies use the traditional support measure or do not qualify the degree of support at all.

In this paper, we evaluate the complexity of a process model through the computation of a distance metric. Other attempts for measuring this complexity focus on the structure of the execution sequences produced by a process [4] or count the number of *and*, *or*, and *xor* nodes in its graph [21]. The *edit distance*, or *Levenshtein distance* [11], originally measures the distance between strings by counting the minimal amount of insertions, substitutions or deletions of characters that are needed to transform one string into the other. Here, we address process models instead of strings. The *graph edit distance* [7] has been put forward for this use case. It computes the distance between two process models using the amount of insertions, deletions and substitutions of nodes and control-flow edges in the process graph. Such edit distances have various applications in the area of service-oriented systems, for example in *service discovery* of composite services [26]. There, they are used to match a set of processes to a given query. Here however, our aim is to accurately classify the cost of changing process models. We argue that the graph edit distance is too abstract to be used for the problem at hand. It does not consider the configuration of nodes or edges and does not consider crucial characteristics that are typical for orchestration languages, such as variables or correlation sets. Consequently, its application would bear the same issues as the traditional support measure. Therefore, we specialize the set of underlying edit operations to capture the specifics of orchestration languages. A specialized set of edit operations allows for a more accurate cost estimation, than a mere comparison of nodes and edges would.

### 3 Edit Distance Support Measure

Our support measure is based on the idea to measure the degree of support provided by a language through the computation of the distance between an executable *process stub* without specific functionality and a process implementing

<sup>1</sup> Please refer to <http://wiki.open-esb.java.net/Wiki.jsp?page=BPELSE>

a given pattern [10]. A process stub is a minimal process definition, a process model that forms the typical starting point of any realistic process. This process stub can be extended with the language constructs that are needed to implement exactly a single pattern. The distance between these two process models, the process stub and the pattern implementation, computed in a metrical scaling, provides a notion of complexity for the implementation of a pattern in a language. It is of importance that the process stub and the computation of the distance metric are similar for different languages. Also, the languages have to reside on a similar level of abstraction, which is the case for the languages in focus here. Omitting this requirement would not allow for meaningful results when directly comparing different languages.

Similarity between the process stubs of different languages can not be based on syntactical similarity because the languages have a different syntax. However, the process stubs can be implemented according to the same abstract scheme which will be presented in the next section, along with code samples<sup>2</sup> for the process stubs in BPEL 2.0, Sun BPEL, and WF 4. In the same fashion, the computation of the distance between the process stub and a pattern implementation using an edit distance, must be based on the same set of edit operations, even for different languages. Therefore, we go on with defining an abstract set of edit operations for orchestration languages and describe the mapping of these edit operations to the specifics of the languages in focus.

### 3.1 Common Schema for Process Stubs

To obtain distance values that are comparable for multiple languages, it is necessary to have a common basis, i.e., to use a semantically equivalent process stub. In this study, we examine multiple orchestration languages. Therefore, also the process stub ought to be applicable to orchestration languages in general. As a minimal feature of the executability of an orchestration model, it should provide the ability to create new process instances. In an orchestration, this is typically done using a *single event trigger* [5] such as an incoming message. Activities that process incoming messages, often called *receive* activities, are most convenient for this purpose. The main aim of the process stub is to extend it with the implementation of pattern. This can be achieved with a *sequence* activity. For the reason of extensibility, it is beneficial to be able to direct some input to a process instance. The initial message that triggers the creation of a new process instance is sufficient for this. Normally, this requires the definition of a variable in the process model, and a mapping from the input message to this variable. In BPEL<sup>3</sup> there are `receive` and `sequence` activities for these purposes. To have input data available for a process instance, a `variable` definition and its

<sup>2</sup> All code samples in this paper are limited to the most crucial aspects that are needed for understanding. Certain features, such as XML namespaces, are omitted completely.

<sup>3</sup> For the sake of brevity, we will not discuss the specifics of Sun BPEL separately from BPEL. Naturally, the languages are similar to a large extent. In cases where differences exist, we will make this explicitly clear.

reference in the `receive` activity are needed. Finally, also the WSDL interface for the BPEL process needs to be imported and used in a `myRole` `partnerLink`. List. 1 (taken from [10]) outlines such a process stub in BPEL 2.0.

**Listing 1.** Process stub for BPEL 2.0 and Sun BPEL

```
<process>
  <import location="ProcessInterface.wsdl" />
  <partnerLinks>
    <partnerLink name="MyPartnerLink" myRole="patternRole" />
  </partnerLinks>
  <variables>
    <variable name="StartProcessInput" messageType="int" />
  </variables>
  <sequence>
    <receive createInstance="yes" variable="StartProcessInput"
      partnerLink="MyPartnerLink" operation="StartProcess"/>
    <!-- Pattern Implementation -->
  </sequence>
</process>
```

In WF, the process stub looks very similar. An orchestration can be implemented as a *workflow service* and `Receive` and `Sequence` activities with the same semantics as in BPEL are available. `Variables` are scoped differently in WF and the definition of the service interface is not needed, but is inferred by the workflow engine from the messaging activities in the workflow. List. 2 outlines the process stub for WF.

**Listing 2.** Process stub for WF 4

```
<WorkflowService>
  <Sequence>
    <Sequence.Variables>
      <Variable Name="InstanceID" TypeArguments="Int32" />
    </Sequence.Variables>
    <Receive CanCreateInstance="True" OperationName="StartProcess">
      <ReceiveParametersContent>
        <OutArgument Key="InputData"> [InstanceID] </OutArgument>
      </ReceiveParametersContent>
    </Receive>
    <!-- Pattern Implementation -->
  </Sequence>
</WorkflowService>
```

These process stubs are now enriched with the implementation of a particular pattern. The distance between a process stub and the pattern implementation is used to determine the degree of support provided for a pattern by the languages. It reflects the effort needed by the implementer of a pattern and the complexity of the implementation.

### 3.2 Edit Operations

The idea for calculating the edit distance here is to count all meaningful *implementation steps* that are needed by a user to achieve the functionality required



by a pattern definition. Therefore, all relevant changes in a process model should be covered and not just changes to nodes and edges in the process graph. In the following, we present a set of edit operations for orchestration languages and describe how they are represented in BPEL 2.0 and WF 4. The operations represent self-contained semantical units that stand for atomic implementation steps from a programmer perspective. As an example, adding a variable to a process model and setting its name and type, is considered as a single implementation step. These steps are independent of the serialization format or graphical representation of a language. This means that, although they might require several changes to the code of a process model or several actions in a graphical editor, they capture a single semantic change of a feature of the process model. Changes to process models can be facilitated by using a sophisticated development environment when implementing a pattern. But in taking into account features such as auto-completion or the macros of an editor, an analysis would no longer evaluate a language, but rather an environment for the language. As this is not our intention, we abstract from any tooling available when calculating the distance values.

The set of edit operations has been identified as follows: Starting with a list of candidate operations, the relevance of an operation was determined by the fact that it was needed to implement the behaviour required by a pattern. During the implementation of certain patterns it became obvious that further edit operations had to be defined, especially for messaging activities, and the list was extended. This procedure is similar to the method used by several authors for extrapolating a set of patterns from a set of real-world process models or the capabilities of several workflow systems [2, 8, 17, 18, 22–24]. No other edit operations apart from the operations described were necessary and it was possible, using this set of operations, to calculate selective and meaningful support values (cf. Sect. 4). More than 150 process models<sup>4</sup> were developed and these models serve as empirical evidence for the relevance of the edit operations from a programmer perspective. Each of the following operations adds one point to the edit distance of a solution to a pattern.

**Insert Activity:** *The insertion or substitution of an activity and the setting of the activity name.*

Any BPEL 2.0 activity and any WF 4 activity is covered by this operation. Further configuration of an inserted activity is not included. In a block-structured process model, activities are necessarily nested. There, inserting an activity also includes the modification of a composite activity (e.g. inserting a child activity). Therefore, the modification of the composite activity is also included in the insertion. For example, inserting an activity into an `onMessage` activity in BPEL 2.0 also modifies it.

**Insert Edge:** *The insertion of a control-flow edge and the setting of its name.*

This operation is generally only available in a graph-oriented model. In BPEL

<sup>4</sup> As discussed, we considered a total of 70 patterns. Because we analyzed three languages, there are up to three process models per pattern. Moreover, several patterns, especially the service interaction patterns [2], require more than one process model for a valid implementation.

2.0, edges are represented by `links` in a `flow` activity. In WF 4, edges are represented by `FlowSteps` in a `Flowchart` activity. An insertion of an edge in a graph also includes the setting of its target and source. All further configuration, such as the setting of a condition for the activation of an edge is not included in its insertion. For all other consideration, edges can be treated just as activities.

**Insert Auxiliary Construct:** *The insertion of a process element, apart from nodes and edges.*

Apart from activities and edges, languages may use a variety of auxiliary constructs that can be defined in a process model and be used by the activities of a process. Such elements are, for example, variables, correlations, or references to partners involved in the process. The insertion of such an element involves its initial configuration, such as the setting of its name and type. In BPEL 2.0, such constructs are `variables`, `correlationSets`, and `partnerLinks`. For a `variable`, the name and type must be fixed. For a `correlationSet`, the name and properties must be specified and for a `partnerLink`, the `name`, `partnerLinkType`, and `role` must be declared. In WF 4, the only additional process elements are `Variables`. Correlations can be defined using variables of a specific type, `CorrelationHandle`, and references to partners are not defined explicitly, but are contained in the configuration of messaging activities. In WF 4, the insertion of a `Variable` involves the setting of its type, name, and optionally a default value.

**Configure Messaging Properties:** *The setting of the messaging properties in a messaging activity.*

Messaging activities require the configuration of several properties, all related to the interface of the service to which they correspond. Typically, this is the setting of a service name and an operation name. The operation name marks an operation provided by the service which is identified by the service name. As they are logically related, these configurations are captured in a single edit operation. In BPEL 2.0, the configuration of the messaging properties of an activity involves the setting of the `partnerLink`, `portType` and `operation`. In WF 4, it corresponds to the setting of the `ServiceContractName` and the `OperationName`.

**Configure Addresses:** *The setting of an endpoint address.*

For an outbound messaging activity, apart from the messaging properties that relate to the interface of the service, also the address of a concrete service instance needs to be set. Otherwise, a messaging activity would not be able to direct a message to it. In BPEL 2.0, this is not necessarily needed, as the address of a service may be inferred from the `partnerLink` used in the activity which needs to be set as part of the messaging properties. This is not the case for WF 4, as it does not make use of an explicit predefined specification of the partners a process interacts with. Here, the relevant addresses need to be set separately for each messaging activity. Setting addresses can be done in several ways, such as via an `Endpoint` element and the setting of its `AddressUri` and the `Binding` to be used.

**Configure Correlations:** *The configuration of message correlation.*

In general, the configuration of message correlation requires a mapping from the parameters available to an activity to a predefined correlation variable. In BPEL 2.0, this operation involves the definition of a `correlations` and a `correlation` element, the setting of its name, and potentially whether the correlation set should be initiated. In WF 4, a `CorrelationHandle` needs to be referenced and a query that determines the elements of the parameters of the activity that identify the correlation needs to be specified.

**Configure Parameters:** *The setting of the input or output parameters of messaging activities.*

In BPEL 2.0, this implies referencing a predefined `variable`. If the parameters capture an outbound message, concrete data has to be assigned to this variable in a separate activity in advance to its use as a messaging parameter. The insertion of the activity that performs this assignment and its configuration is *not* covered by this edit operation. In WF 4 it implies the definition of a parameter and the mapping of the parameter to a predefined `Variable` using an expression.

**Configure Expression:** *The setting of an attribute value of an activity, edge, or auxiliary construct to an expression.*

Several attributes of process elements may require expressions as values. Examples are logical expressions used in the termination condition of a looping activity. The construction of such an expression may require several steps and may involve the use of several operators in a dedicated expression language (which is XPath 1.0 for BPEL 2.0 and Visual Basic for WF 4). The construction of an expression is rated as a single edit operation.

**Assignment:** *The setting of the content of an assign activity.*

The assignment of a value to a variable involves the specification of the variable name and the expression that produces the value which is assigned. While WF 4 allows for single assignments in an `Assign` activity only, BPEL 2.0 allows for multiple ones using multiple `copy` elements.

**Change Configuration:** *The change of any default value of an activity, edge or auxiliary construct to another value.*

All other changes of the configuration of the elements of a process can be represented by this operation. In BPEL 2.0 or WF 4, this could for example be the change of an attribute value or the setting of a child element of an activity. Several activities capture their configuration in child elements. An example is an `onAlarm` in BPEL 2.0 where the configuration of the wait condition is fixed in a `for` or `until` element.

The goal of the edit operations chosen is to measure the effort of implementing a pattern from a programmer perspective. The granularity of this set of operations essentially is a design choice and it would be an option to decrease or increase it. For instance, rating the insertion of an activity and its complete configuration as a single operation decreases the granularity, while counting each modification of an attribute of a construct increases it. Although we did not perform a complete analysis, we calculated the distance values for a subset of the process models with

these two modifications and observed the effect this had on the selectivity of the measure and its ability to characterize the complexity of the process models. It turned out that in neither case, the ability of the measure to discriminate between the languages (cf. Sect. 4) differed strongly. The problem with the first approach is that it assigns similar distance values to the implementations of control-flow and service interaction patterns, in spite of the fact that the latter describe more complicated scenarios. This is the case, because even complex messaging activities are as costly as more simple activities. As an example, a sequence of two **assign** activities would have the same distance value as a sequence of a **send** and a **receive** activity, although the latter two are more complicated to configure. In the second approach, activities that are rather simple but needed frequently and require extensive configuration tend to hide the complexity of other activities where the setting of attribute values corresponds to important design decisions. Examples are **assign** activities which require the specification of sources, targets, and possibly expressions or type specifications, and easily outweigh the configuration of a looping activity to execute in parallel. To sum up, the set of edit operations presented here provides, in our point of view, an acceptable trade off between the selectivity of the measure and its ability to assess the complexity of a pattern. An empirical study is needed to assess whether the complexity values calculated with the measure really correspond to the impression of human implementers.

From a theoretical point of view, edit distances that do not use the same weights for insertion and deletion operations are *quasi-metrics* [29, p. 12], as they do not satisfy the property of symmetry. Here, we assign higher costs to insertions, as we specify a number of fine-grained insertion operations. Deletions can only be achieved indirectly through substitution. The upper bound of the computation complexity of the distance value of a process model to a process stub is linear to the number of activities and auxiliary constructs used, multiplied with the maximum number of configuration options available for an activity.

### 3.3 Calculation Example

Next, we describe an implementation of the *Racing Incoming Messages* pattern [2] and present a code sample for the executable process in WF 4. This pattern describes a scenario where a party awaits one out of a set of messages. The messages may be of different structure, originate from different parties and be processed in a different manner, depending on their type. This aspect is well understood in all of the languages in focus here, although the solutions differ slightly. The implementation presented is motivated by the solution to the pattern in BPEL 1.1 described in [2].

The implementation in WF 4 builds upon the **Pick** activity. This activity contains multiple **PickBranch** activities, one for each of the messages that can be received. A minimal realization of the pattern contains two alternative messages. Each **PickBranch** activity contains a **Trigger**. Any WF activity can serve as **Trigger** and as soon as the **Trigger** completes, the according body is executed. When a **Trigger** completes, all other **PickBranches** are canceled.

**Listing 3.** Racing Incoming Messages pattern in WF

```

<WorkflowService>
  <Sequence>
    <Pick>
      <PickBranch>
        <PickBranch.Trigger>
          <Receive ServiceContractName="RacingIncomingMessages"
            OperationName="ReceiveMessageA" CanCreateInstance="True" />
        </PickBranch.Trigger>
        <!-- Process MessageA -->
      </PickBranch>
      <PickBranch>
        <PickBranch.Trigger>
          <Receive ServiceContractName="RacingIncomingMessages"
            OperationName="ReceiveMessageB" CanCreateInstance="True" />
        </PickBranch.Trigger>
        <!-- Process MessageB -->
      </PickBranch>
    </Pick>
  </Sequence>
</WorkflowService>

```

This structure is outlined in List. [3](#). The following edit operations add to the edit distance of this process compared to the process stub presented in List. [2](#):

1. **Insert Activity:** Substitute the `Receive` activity from the process stub with the `Pick` activity.
2. **Insert Activity:** Insert the first `PickBranch` activity.
3. **Insert Activity:** Insert the first `Receive` activity into the `Trigger` of the first `PickBranch` activity.
4. **Configure Messaging Properties:** To be able to receive messages, the `OperationName` and `ServiceContractName` of the first `Receive` activity has to be set.
5. **Configure Activity:** The `Receive` activity must be able to create a new process instance (by setting its `CanCreateInstance` attribute to `true`), otherwise the executable process would not be valid.
6. **Insert Activity:** Insert the second `PickBranch` activity.
7. **Insert Activity:** Insert the second `Receive` activity into the `Trigger` of the second `PickBranch` activity.
8. **Configure Messaging Properties:** To be able to receive messages, the `OperationName` and `ServiceContractName` of the second `Receive` activity has to be set.
9. **Configure Activity:** Set the `CanCreateInstance` attribute of the second `Receive` activity to `true`.

In total, this adds up to an edit distance of nine. There are similar process models in BPEL 2.0 and Sun BPEL, based on the `pick` activity. BPEL 2.0 scores an edit distance of eight, and Sun BPEL of ten (cf. Table [2](#)). The edit distance allows to see that there are subtle differences in the degree of support provided by the languages. Things are different, when using the traditional trivalent measure. There, the solution achieves direct support, as there is a single essential

activity (the Pick) implementing the pattern. This valuation could be questioned, as there is undoubtedly more than one activity involved in the solution. Nevertheless, this valuation corresponds to the assumptions made in other evaluations [2, 6]. The same applies to the solutions in BPEL 2.0 and Sun BPEL. They also result in a rating of direct support. So when using the traditional measure, this pattern reveals no difference in the support provided by WF 4, BPEL 2.0, or Sun BPEL.

## 4 Results and Evaluation

In the following sections, we present the results of an assessment of the languages WF 4, BPEL 2.0, and Sun BPEL for the *control-flow patterns* [19, 23], the *service interaction patterns* [2], the *time patterns* [8] and the *patterns for changes in predefined regions* [24]. A detailed discussion of all these patterns and the solutions to them in the respective languages is not possible in the context of this paper. Therefore, the following sections present the overall results of the analysis and discuss their implications. We refer the interested reader to a technical report [9] for a description of every pattern and a discussion of the solutions in each of the languages. All process models that have been developed are available.<sup>5</sup> Our intention when developing the process models was to minimize the edit distance while providing a valid solution to a pattern. More efficient solutions to the patterns in terms of computing complexity may be possible.

### 4.1 Control-Flow Patterns

Table 4.1 outlines the results for the control-flow patterns [19, 23] and compares them to the results of studies which analyzed preceding versions of BPEL and WF. The control-flow patterns describe typical structures of the control-flow perspective of automated processes. Our solutions to the patterns in the newer language revisions were motivated by preceding studies [19, 25, 27], given required language constructs were still in place in the newer language versions. The results of preceding studies only present the trivalent support measure.

Table 4.1 reveals that the edit distance support measure provides a higher degree of selectivity among the languages than the traditional trivalent measure does. The number of solutions in both, BPEL 2.0 and WF 4, can be used to assess the quality of the languages, but also to quantify the degree of selectivity provided by a support measure. This quantification is based on the number of solutions where a support measure discriminates between the languages in relation to the total number of solutions to the same patterns. A value of 1 for this relation states that a support measure completely discriminates in all cases. A value of 0 states that a support measure discriminates in no case. For 30 of the 43 patterns here, solutions could be found in WF 4. In BPEL 2.0, 31 patterns could be implemented. For 29 patterns, solutions could be found in both, WF 4

<sup>5</sup> The process models can be downloaded at <http://www.uni-bamberg.de/pi/orch-pattern>. [9] also contains a description on how to execute them.

**Table 1.** Support of workflow control-flow patterns. If available, the edit distance is displayed first followed by the trivalent measure in parentheses. A value of ‘-’ for the edit distance means that no valid solution could be found. As opposed to this, a value of ‘-’ for the trivalent measure means that either no valid solution could be found or that all possible valid solutions require the use of more than two constructs.

Pattern	WF 3.5 taken from [27]	WF 4	BPEL 1.1 taken from [19]	BPEL 2.0	Sun BPEL
<b>Basic Patterns</b>					
WCP-1. Sequence	+	2 (+)	+	2 (+)	2 (+)
WCP-2. Parallel Split	+	3 (+)	+	3 (+)	3 (+)
WCP-3. Synchronization	+	3 (+)	+	3 (+)	3 (+)
WCP-4. Exclusive Choice	+	4 (+)	+	4 (+)	4 (+)
WCP-5. Simple Merge	+	4 (+)	+	4 (+)	4 (+)
<b>Advanced Branching and Synchronization Patterns</b>					
WCP-6. Multi-Choice	+	7 (+/-)	+	7 (+/-)	7 (+/-)
WCP-7. Structured Synchronizing Merge	+	7 (+/-)	+	7 (+/-)	7 (+/-)
WCP-8. Multi-Merge	-	- (-)	-	- (-)	- (-)
WCP-9. Structured Discriminator	+/-	9 (+/-)	-	10 (-)	10 (-)
WCP-28. Blocking Discriminator	-	- (-)	-	- (-)	- (-)
WCP-29. Cancelling Discriminator	+	9 (+)	-	10 (-)	10 (-)
WCP-30. Structured Partial Join	+/-	12 (+/-)	-	31 (-)	31 (-)
WCP-31. Blocking Partial Join	-	- (-)	-	- (-)	- (-)
WCP-32. Cancelling Partial Join	+	12 (+)	-	31 (-)	31 (-)
WCP-33. Generalized AND-Join	-	- (-)	-	- (-)	- (-)
WCP-37. Acyclic Synchronizing Merge	+/-	- (-)	+	11 (+)	- (-)
WCP-38. General Synchronizing Merge	-	- (-)	-	- (-)	- (-)
WCP-41. Thread Merge	-	- (-)	+/-	- (-)	- (-)
WCP-42. Thread Split	-	- (-)	+/-	- (-)	- (-)
<b>Multiple Instances (MI) Patterns</b>					
WCP-12. MI without Synchronization	+	6 (+)	+	7 (+)	12 (+/-)
WCP-13. MI with a priori Design-Time Knowledge	+	6 (+)	+	7 (+)	19 (+/-)
WCP-14. MI with a priori Run-Time Knowledge	+	6 (+)	-	7 (+)	- (-)
WCP-15. MI without a priori Run-Time Knowledge	-	- (-)	-	- (-)	- (-)
WCP-34. Static Partial Join for MI	+/-	10 (+/-)	-	8 (+/-)	- (-)
WCP-35. Cancelling Partial Join for MI	+	10 (+)	-	8 (+)	- (-)
WCP-36. Dynamic Partial Join for Multiple Instances	-	- (-)	-	- (-)	- (-)
<b>State-based Patterns</b>					
WCP-16. Deferred Choice	+	9 (+/-)	+	8 (+)	10 (+)
WCP-17. Interleaved Parallel Routing	+	- (-)	+/-	16 (+/-)	- (-)
WCP-18. Milestone	+	11 (+/-)	-	11 (+/-)	11 (+/-)
WCP-39. Critical Section	+	9 (+/-)	+	15 (+/-)	40 (-)
WCP-40. Interleaved Routing	+	9 (+/-)	+	15 (+/-)	40 (-)
<b>Cancellation Patterns</b>					
WCP-19. Cancel Activity	+	9 (+/-)	+	8 (+/-)	8 (+/-)
WCP-20. Cancel Case	+	4 (+)	+	3 (+)	3 (+)
WCP-25. Cancel Region	+	9 (+/-)	+/-	8 (+/-)	8 (+/-)
WCP-26. Cancel MI Activity	+	14 (+/-)	-	13 (+/-)	55 (-)
WCP-27. Complete MI Activity	-	10 (+)	-	8 (+)	- (-)
<b>Iteration Patterns</b>					
WCP-10. Arbitrary Cycles	+	17 (-)	-	18 (-)	18 (-)
WCP-21. Structured Loop	+	5 (+)	+	5 (+)	5 (+)
WCP-22. Recursion	-	- (-)	-	- (-)	- (-)
<b>Termination Patterns</b>					
WCP-11. Implicit Termination	+	0 (+)	+	0 (+)	0 (+)
WCP-43. Explicit Termination	+	9 (+/-)	-	6 (+)	6 (+)
<b>Trigger Patterns</b>					
WCP-23. Transient Trigger	+	- (-)	-	- (-)	- (-)
WCP-24. Persistent Trigger	+	0 (+)	+	0 (+)	0 (+)

and BPEL 2.0. Only in six of these cases, the trivalent measure discriminates, so the degree of selectivity amounts to  $6/29 = 0.21$ . The edit distance discriminates in 18 cases, so this number amounts to  $18/29 = 0.62$ . For all 25 patterns to which solutions could be found in Sun BPEL, also solutions in WF 4 could be found. Here, the degree of selectivity of the trivalent measure amounts to  $11/25 = 0.44$ , and for the edit distance it amounts to  $14/25 = 0.56$ .

It is not surprising that the degree of support for several patterns, such as *Parallel Split* or *Exclusive Choice*, is identical in WF 4 and BPEL 2.0 even using the edit distance. These patterns relate to concepts that are very common and consequently the solutions are very similar. For several patterns, such as the *Discriminator* and *Partial Join* patterns in BPEL 2.0, it is interesting to see that there is no support according to the trivalent measure, but the edit distance shows that they are relatively easy to implement.

The degree of pattern support has changed marginally from BPEL 1.1 to BPEL 2.0. There are few differences in the set of activities available and only the new parallel `forEach` activity has an impact on the support for control-flow patterns provided by the language. With the help of this activity, several of the *Multiple Instances* patterns can be supported. More efficient solutions to the *Discriminator* and *Partial Join* patterns in BPEL 2.0 could be implemented by providing a `completionCondition` for the `flow` activity similar to the `forEach` activity. BPEL 2.0 does not support several patterns due to its structuredness and the inability to create cycles using `links`, as well as its threading model. The support provided by Sun BPEL is severely limited by its lack of `links`, isolated `scopes` and parallel `forEach` activities.

When comparing WF 3.5 to WF 4, one thing becomes obvious: While the solutions of the patterns have changed considerably (cf. [9]), there is little change in the overall degree of support provided by the language. All in all however, fewer patterns are supported. There are two main reasons for this. First, the lack of the state machine modeling style that was present in WF 3.5 limits the support. This modeling style was especially suited to provide elegant solutions to state-based patterns and several other patterns for unstructured process models. In April 2011, Microsoft reacted to the demands of the community of WF users and re-introduced the state machine modeling style for WF 4 in its first platform update of .NET 4 [6]. Second, while the new flowchart modeling style provides an excellent means for building unstructured, graph-oriented process models, it is not able to live up to its full potential, due to its inability to describe concurrent branches. A *Parallel Split* or *Multi-Choice* construct is yet missing in this style.

Altogether, the support provided by WF 4 and BPEL 2.0 is still very similar. Things are different when comparing Sun BPEL and WF 4. WF 4 provides support for more patterns and the solutions are often also less complex.

## 4.2 Service Interaction Patterns

Service interaction patterns [2] describe interaction scenarios that are typical in the B2B domain. Due to their distributed nature, nearly all service interaction

---

<sup>6</sup> The documentation is available at <http://support.microsoft.com/kb/2478063>.



patterns must be implemented by more than one process. In most cases there is an initiator process that starts a communication session and one or more responder processes. The edit distance that describes the support for a pattern is the sum of the edit distances of all the processes involved. The results of the analysis for the service interaction patterns are outlined in Table 2. The degree of support

**Table 2.** Support of Service Interaction Patterns

Pattern	WF 4	BPEL 2.0	Sun BPEL
<b>Single-Transmission Bilateral Patterns</b>			
SIP-1 Send	4 (+)	7 (+)	7 (+)
SIP-2 Receive	4 (+)	7 (+)	7 (+)
SIP-3 Send/Receive	9 (+)	15 (+/-)	15 (+/-)
<b>Single-Transmission Multi-lateral Patterns</b>			
SIP-4 Racing Incoming Messages	9 (+)	8 (+)	10 (+)
SIP-5 One-to-Many Send	9 (+)	12 (+)	12 (+)
SIP-6 One-from-Many Receive	37 (+/-)	49 (+/-)	49 (+/-)
SIP-7 One-to-Many Send/Receive	36 (+/-)	- (-)	- (-)
<b>Multi-Transmission Bilateral</b>			
SIP-8 Multi Responses	71 (-)	90 (-)	90 (-)
SIP-9 Contingent Requests	28 (+)	34 (+)	34 (+)
SIP-10 Atomic Multicast Notification	40 (-)	- (-)	- (-)
<b>Routing Patterns</b>			
SIP-11 Request with Referral	21 (+)	28 (+)	- (-)
SIP-12 Relayed Request	31 (+/-)	47 (+/-)	- (-)
SIP-13 Dynamic Routing	- (-)	- (-)	- (-)

for the service interaction patterns provided by WF 4 is considerably better than that of BPEL 2.0. WF 4 provides a wider range of messaging activities and its correlation mechanism is less restrictive. WF 4 supports more patterns and, as the edit distance demonstrates, almost all solutions are less complex. As before, Sun BPEL falls behind the other two languages in both, the number of patterns supported and the complexity of the solutions. Especially its lack of support for dynamic partner binding is critical, resulting from the inability to re-assign endpoint references to `partnerLinks`.

### 4.3 Time Patterns

Time patterns [8] mark typical time-related constraints of the control-flow perspective of processes. The results of the analysis of the support for time patterns are given in Table 3. The support for time patterns relies heavily on the representation for dates and times and the expression languages available. WF 4 uses sophisticated data types and time-based operations from the .NET class library. BPEL 2.0 requires only the support for XPath 1.0 as expression language, which completely lacks time-based operations. Sun BPEL incorporates some time-based functions of XPath 2.0 and thus allows to increase the degree of pattern support. In fact, for this pattern catalog, Sun BPEL excels BPEL 2.0. By consolidating the BPEL standard to also require the support for XPath 2.0 as expression language, BPEL would achieve a similar degree of support as WF 4.

**Table 3.** Support of Time Patterns

Pattern	WF 4	BPEL 2.0	Sun BPEL
<b>Durations and Time Lags</b>			
TP-1. Time Lags between two Activities	8 (+)	- (-)	- (-)
TP-2. Durations	6 (+/-)	7 (+/-)	7 (+/-)
TP-3. Time Lags between Events	8 (+)	- (-)	- (-)
<b>Restrictions of Process Execution Points</b>			
TP-4. Fixed Date Elements	3 (+)	3 (+)	3 (+)
TP-5. Schedule Restricted Elements	3 (+)	- (-)	- (-)
TP-6. Time Based Restrictions	6 (+)	- (-)	- (-)
TP-7. Validity Period	4 (+)	- (-)	4 (+)
<b>Variability</b>			
TP-8. Time Dependent Variability	3 (+)	11 (+/-)	4 (+)
<b>Recurrent Process Elements</b>			
TP-9. Cyclic Elements	12 (+/-)	- (-)	- (-)
TP-10. Periodicity	8 (+/-)	7 (-)	7 (-)

#### 4.4 Patterns for Changes in Predefined Regions

Patterns for changes in predefined regions are a subset of the change patterns [24]. They describe structures that allow for changes in the control-flow perspective of processes at run-time. In most cases, these structures can be captured using certain control-flow patterns. As can be seen in Table 4, WF 4 and BPEL 2.0 are roughly equivalent concerning their support for patterns for changes in predefined regions. On average, the solutions in WF 4 are less complex. In any case WF 4 and BPEL 2.0 excel Sun BPEL.

**Table 4.** Support of Patterns for Changes in Predefined Regions

Patterns for Changes in Predefined Regions	WF 4	WS-BPEL 2.0	Sun BPEL
PP-1. Late Selection of Process Fragments	9 (+/-)	8 (+)	10 (+)
PP-2. Late Modeling of Process Fragments	- (-)	- (-)	- (-)
PP-3. Late Composition of Process Fragments	9 (+/-)	15 (+/-)	40 (-)
PP-4. Multiple Instance Activity	6 (+)	7 (+)	- (-)

## 5 Conclusion and Outlook

This study introduced an edit distance-based measure for assessing pattern support that overcomes granularity and selectivity issues of the traditional measure. Its applicability was assessed by an analysis of the orchestration languages BPEL 2.0, Sun BPEL, and WF 4. The use of this support measure for calculating the degree of support overcomes the problems the traditional trivalent support measure posed on preceding analyses. What is more, it gives a notion for the complexity of a solution to a pattern in a language and the effort required by its implementer. Also, it is directly comparable across the boundaries of languages and pattern catalogs. Future analyses can provide more meaningful and selective results by relying on this edit distance support measure.

Furthermore, the results show that WF 4 excels both, BPEL 2.0 and its implementation Sun BPEL, concerning the degree of pattern support. BPEL 2.0 and WF 4 are largely equivalent concerning their degree of support for control-flow and change patterns. Things are different when looking at the service interaction

and time patterns. WF 4 supports two service interaction patterns that are not supported by BPEL 2.0 and more than twice as many time patterns. Furthermore, for almost all time and service interaction patterns, the solutions are less complex in WF 4. For Sun BPEL, the analysis demonstrates that its degree of pattern support is rather limited.

Future work concentrates on the automation of the calculation of the edit distance. The edit operations presented here can serve as foundation for a unified model of orchestration languages that allows for an automated calculation of distance values. To fully automate the computation, it is necessary to construct a mapping from a concrete orchestration language to this model. Another open issue is the assessment of the efficiency and scalability of the solutions described here. As discussed, we cannot guarantee that we have found the most efficient solutions to all patterns in the languages in focus. A community approach, starting with the results from [9] and involving researchers from other institutions, might help to optimize the process models. Also the analysis of closely related languages, such as BPMN 2.0, is an interesting field of study.

## References

1. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)
2. Barros, A., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
3. Bukovics, B.: Pro WF: Windows Workflow in .NET 4. Apress (June 2010), ISBN-13: 978-1-4302-2721-2
4. Cardoso, J.: Business Process Quality Metrics: Log-Based Complexity of Workflow Patterns. In: Meersman, R., Tari, Z., et al. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 427–434. Springer, Heidelberg (2007)
5. Decker, G., Mendling, J.: Process Instantiation. *Data and Knowledge Engineering* 68, 777–792 (2009)
6. Decker, G., Overdick, H., Zaha, J.: On the Suitability of WS-CDL for Choreography Modeling. In: EMISA, Hamburg, Germany, pp. 21–33 (October 2006)
7. Dijkman, R., 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. Lanz, A., Weber, B., Reichert, M.: Workflow Time Patterns for Process-Aware Information Systems. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 94–107. Springer, Heidelberg (2010)
9. Lenhard, J.: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Technical Report 88, Otto-Friedrich-Universität Bamberg, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik (March 2011)
10. Lenhard, J., Schönberger, A., Wirtz, G.: Streamlining Pattern Support Assessment for Service Composition Languages. In: ZEUS, Karlsruhe, Germany. CEUR Workshop Proceedings, vol. 705, pp. 112–119. CEUR-WS.org (February 2011)
11. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady* 10(8), 707–710 (1966)

12. Norta, A., Hendrix, M., Grefen, P.: A Pattern-Knowledge Base Supported Establishment of Inter-organizational Business Processes. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 834–843. Springer, Heidelberg (2006)
13. OASIS. Web Services Business Process Execution Language, v2.0 (April 2007)
14. O’Hagan, A., Sadiq, S., Sadiq, W.: Evie - A developer toolkit for encoding service interaction patterns. *Information Systems Frontiers* 11(3), 211–225 (2009)
15. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented Computing. *Communications of the ACM* 46(10), 24–28 (2003)
16. Peltz, C.: Web Services Orchestration and Choreography. *IEEE Computer* 36(10), 46–52 (2003)
17. 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)
18. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Data Patterns: Identification, Representation and Tool Support. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 353–368. Springer, Heidelberg (2005)
19. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. Technical report, BPM Center Report (2006)
20. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow Exception Patterns. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
21. Sánchez-González, L., Ruiz, F., García, F., Cardoso, J.: Towards Thresholds of Control Flow Complexity Measures for BPMN models. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 1445–1450. ACM (2011)
22. Thom, L.H., Reichert, M., Iochpe, C.: Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence. *IJBPM* 4(2), 93–110 (2009)
23. 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)
24. Weber, B., Rinderle-Ma, S., Reichert, M.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowledge Engineering* 66(3), 438–466 (2008)
25. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of Web Services Composition Languages: The Case of BPEL4WS. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 200–215. Springer, Heidelberg (2003)
26. Wombacher, A., Li, C.: Alternative Approaches for Workflow Similarity. In: IEEE SCC, Miami, Florida, USA, pp. 337–345 (July 2010)
27. Zapletal, M., van der Aalst, W.M.P., Russell, N., Liegl, P., Werthner, H.: An Analysis of Windows Workflow’s Control-Flow Expressiveness. In: ECOWS, Eindhoven, The Netherlands, pp. 200–209 (November 2009)
28. Zdon, U., Dustdar, S.: Model-driven and pattern-based integration of process-driven soa models. *IJBPM* 2(2), 109–119 (2007)
29. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. *Advances in Database Systems*, vol. 32. Springer, Heidelberg (2006) ISBN 978-0-387-29146-8

# Towards Robust Service Workflows: A Decentralized Approach (Short Paper)

Mario Henrique Cruz Torres and Tom Holvoet

Department of Computer Science  
Katholieke Universiteit Leuven  
Leuven, Belgium

{MarioHenrique.CruzTorres, Tom.Holvoet}@cs.kuleuven.be

**Abstract.** Nowadays service workflows are used to increase the efficiency of process oriented supply chains. Service workflows can encompass hundreds of services around a single process. These services are geographically spread and cross organizational boundaries. This raises the need for coordination, such as assigning tasks, synchronizing production schedules, between companies collaborating through services. We present a fully decentralized coordination mechanism that, using the local knowledge available at each company participating in the supply chain, allows the enactment of robust processes. We evaluate our solution through simulations and show that it can create robust service compositions.

## 1 Introduction

Nowadays the business world faces an increasing pressure to quickly create new products and offer them at lower prices. In order to achieve these goals companies focus on their core specialties and at the same time collaborate with a wide range of suppliers. Several companies organize themselves around processes, creating process oriented supply chains. On the one hand process oriented supply chains help companies to improve their efficiency, be more agile and responsive to market changes. On the other hand, process oriented supply chains demand better communication, even more, better collaboration amongst its participants [1]. Cooperation is a cornerstone to enable process oriented supply [2].

Companies can expose their services on the internet. However there is no simple way to define how multiple services can interact with each other in order to improve the overall efficiency of a system, or of a process. There are studies proposing how services can share resource information in standardized ways [3]. However there is a lack of mechanisms to specify how services interact, which information to share, and how to monitor cross-organizational business processes.

Our main contribution is to present a decentralized coordination mechanism that selects services to participate in a supply chain process. Our mechanism, based on Delegate MAS [4], is a step in the direction of decentralized services in supply chains.

The rest of the paper is organized as follows. Section 2 describes the problem explored. We present our solution in Section 3 and evaluate it on Section 4. Finally, Section 5 shows the related works and Section 6 our conclusions.

## 2 Service Selection in Supply Chains

The Supply Chain Management (SCM) problem we tackle in our research is how to coordinate the allocation, and monitoring of the activities of companies engaged in a supply chain process in order to have robust processes. A supply chain process, is defined by a set of activities that need to be performed in order to achieve a goal, such as producing and delivering a product.

The problem becomes even more complex due to dynamic nature of the events that occur in the environment. Communication networks can fail, trucks can break, schedules can slip, companies can have weak commitments towards the supply chain.

### 2.1 Formalization

We model the supply chain as a graph  $Sc = \langle A, C \rangle$ . Where  $A = \{a_1, a_2, \dots, a_n\}$  is the set of agents representing partner companies, and  $C \subseteq A \times A$  is the set of connections between agents.

The set of all possible service types offered by the companies participating in our supply chain is defined by  $S = \{s_0, s_1, \dots, s_n\}$ , where  $s_i \in \mathbb{N}$ . Agents maintain a schedule indicating when they are busy. There is also a monetary price associated to the use of an agent. An agent  $a_i$  is defined as:

$$a_i = (s_i, J_i, p_i) \text{ where } \begin{cases} s_i \in S & \text{service type} \\ J_i = \{(t_0, e_0), (t_1, e_1), \dots, (t_n, e_n)\} & \text{schedule} \\ p_i \in \mathbb{R}^+ & \text{price} \end{cases} \quad (1)$$

Where  $s_i$  is the type of service offered by the agent. The agents maintain information about the times they will execute an operation and the expected duration time to perform such operation. This information is the agent schedule  $J$ , that is a set of tuples  $\langle t, e \rangle$ , where  $t$  is the time to start executing an operation and  $e$  is the expected duration to execute such operation. Finally,  $p$  is the monetary price to use the agent service.

A process oriented supply chain constitutes of a number of activities that need to be executed in a specific order to create the desired product. For simplicity, we define a process  $P = \{s_i, \dots\}$ , where  $s_i \in S$ , as a set of sequential service types that need to be invoked to complete the process.

When a distributor requests a new batch of products, it starts a process instance. A process instance can be seen as a path containing the expected time to start executing a service and the agent that will execute the service. A path is defined as

$$Path = \{\langle t_j, a_k \rangle\}, \text{ where } t_j \leq t_{j+1}, \forall t \in \mathbb{N} \text{ and } a_k, \forall a_k \in A \quad (2)$$

A given  $Path$  is a valid path for the process  $P$  if and only if for each pair  $\langle t, a \rangle$ , the agent  $a$ , associated to the service type  $s_i$  from  $P$ , provides operations required by services of type  $s_i$ . This means that a path is valid if and only if the agents scheduled to participate in the path offer services of the same type required by the process.

The expected completion time of a  $Path = \{\langle t_1, a_1 \rangle, \langle t_2, a_2 \rangle, \dots, \langle t_n, a_n \rangle\}$  is given by  $t_n + \theta(a_n, t_n)$ , where  $\theta : A \times \mathbb{N} \rightarrow \mathbb{N}$  provides the time it takes to execute a task in a given agent  $a_n$  at a given time  $t_n$ . Agents use the schedule information  $J$  and the time information to define the function  $\theta$ .

The main problem we want to solve is to find a robust  $Path$ , illustrated in Equation 3 while minimizing the costs associated to the selected path. The metric we use to define a robust path is the variation of total completion times.

$$\min \sqrt{\frac{1}{n} \sum_{i=0}^n \left( \theta(a_i, t_i) - \overline{\theta(a, t)} \right)^2} \quad (3)$$

### 3 Decentralized Service Coordination

In our approach, all supply chain companies participate, through their agents, in a service overlay network. This network is created according to trust relations, business contracts, between the companies. We assume that each agent maintains the information about its peers in the network. Hence there is no central entity used to store a map of the network.

Selecting services to participate in a supply chain process is a complex problem. We use Ant Colony Optimization (ACO) techniques in our coordination mechanism to tackle this complexity.

#### 3.1 Ant Colony Optimization

ACO algorithms are inspired by food foraging behavior of real ant colonies. Ants are good at finding the shortest paths between their nest and sources of food. However they have little capacity to find the paths individually, they collaborate with each other to solve their food foraging problem. Ants drop pheromones in the environment leaving pheromone trails where they walk. Pheromones are chemical scents that stay in the air during a certain period. As more ants follow a certain path, they reinforce the pheromone trail on that path, leading to even more ants to follow that particular path [5].

In ACO a set of virtual ants, which are software agents, indirectly cooperate to find solutions to complex optimization problems. The ACO meta-heuristic defines a number of steps to create algorithms that mimic ant behavior. It can be applied to any problem that can be reduced to path-traversal problems.

The virtual ants have well defined behavior in order to contribute to finding solutions. Virtual ants *drop* information, called pheromones, along the paths that lead to good solutions. That way, other virtual ants, can *smell* the pheromone trail and follow it as well, converging to a path that represents a good solution to the problem at hand.

We use the pheromone concept to avoid direct communication between the agents, and also to indicate the quality of a certain path. We also use the concepts of virtual ants in our agents, called ExplorationAnts and IntentionAnts. We also use the probabilistic nature of the ants behavior to design our ExplorationAnts, what is explained next.

### 3.2 Decentralized Coordination and ACO

Companies taking part in the service overlay network can have multiple participation modes: (i) service provider, which offer operations to other companies, (ii) service requester, which request operations from other companies.

A service provider is represented by a **ResourceAgent**. ResourceAgents are also responsible for storing the quality of service (QoS) information of the services they represent. Besides representing service providers, ResourceAgents maintain a reservation list indicating when their services will be used. This reservation list is frequently updated to avoid having stale information, i.e. in the case an agent does not intend to use the resource anymore. ResourceAgents also maintain information about their direct peers.

**OrgAgents** represent the service requesters. OrgAgents are responsible for selecting the services that will participate in a composition and are also responsible for maintaining the QoS of the composite service they represent. The main goal of OrgAgents is to create robust compositions, that is, enact supply chain processes with the best participants. OrgAgents delegate parts of their work to other agents, more specifically to ExplorationAnts and IntentionAnts, as TaskAgents in Delegate MAS.

ExplorationAnts and IntentionAnts are a special type of agent with constrained behavior. An OrgAgent sends out a number of ExplorationAnts to find out what are the best services available to accomplish the process at hand.

Once the ExplorationAnts find a valid path, they return this information to their OrgAgent. The OrgAgent collects the information brought by all ExplorationAnts and decides, based on its goals, to commit to one particular path. When an OrgAgent commits to a particular path, it sends out IntentionAnts that will reinforce the intention of the OrgAgent, making reservations, to use the services offered the ResourceAgents along the selected path.

**ExplorationAnts** main function is to search for ResourceAgents that can participate in a certain process. In order to do that, ExplorationAnts crawl the agent overlay network looking for ResourceAgents that: (i) offer the required type of service, (ii) can perform the service at the required time, and (iii) have a good QoS. The QoS is represented by the vector  $q = (q_1, \dots, q_n)$ , where  $q_i \in \mathbb{R}$  represents the quality  $i$ . The quality can be, for instance, the duration time to execute one operation, price, trustiness, etc. ExplorationAnts try to find a path containing agents capable of performing all the required operations needed by their OrgAgents. ExplorationAnts take QoS parameters into account, asking this information to ResourceAgents. They evaluate the QoS using an heuristic  $\eta$ , defined in Equation 4:

$$\eta : (q_1, \dots, q_n) \rightarrow \mathbb{R}, \text{ such that } \eta((q_1, \dots, q_n)) = \frac{1}{\sum_i^n q_i^*}, q_i^* \text{ is the normalized } q_i \quad (4)$$

where  $\eta$  is evaluated every time an ExplorationAnt checks for the quality of a service represented by a given ResourceAgent. An ExplorationAnt decides on which path to follow according to the probability given by Equation 5:

$$P_{ij}(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{l \in N_i} [\tau_{il}(t)]^\alpha [\eta_{il}(t)]^\beta} \quad (5)$$



where  $\tau$  is the pheromone level,  $\alpha$  indicates how much the ExplorationAnt values the pheromone information,  $\eta$  is the heuristic that takes the QoS into account, and  $\beta$  indicates the weight to the QoS information the ExplorationAnt gives.

ExplorationAnts select a path to follow according to the probabilities defined in Equation 5. There is always room for exploratory behavior, unless the parameter  $\alpha$  is set to high. If  $\alpha$  is set to high, ExplorationAnts will only follow the paths with the highest pheromone concentration. ExplorationAnts jump from ResourceAgent to ResourceAgent, either until their *time to live* expires or they have found a valid path, that is, they have found all the required ResourceAgents needed for the composition.

When an OrgAgent decides to follow a particular path, it sends out **IntentionAnts**. IntentionAnts make reservations to use the services of the selected ResourceAgents. Another function of IntentionAnts is to drop pheromones along the chosen path in the agent overlay network. The pheromone information is used by ExplorationAnts, when they are looking for services.

If a pheromone trail becomes too strong, ExplorationAnts will have a higher probability to follow this trail instead of exploring new solutions. We use a mechanism called **Pheromone Evaporation** that decreases the pheromone level associated to a certain path as time pass by.

## 4 Evaluation

This section shows the experimental results that we had simulating our coordination mechanism in different scenario settings. As explained in Section 3 there are two types of agents operating in the overlay network, ResourceAgents and OrgAgents, and two types of ants, ExplorationAnts and IntentionAnts. All ResourceAgents are bootstrapped with the same parameter configurations. OrgAgents are bootstrapped with the same commitment level.

In our simulation's scenario there are 20 agents of type Manufacture1 (mean response 110 ms  $\pm$ 11 ms), 100 agents of type TransportRegion1 (mean response time 300 ms  $\pm$ 30 ms), 20 agents of type Manufacture2 (mean response time 80 ms  $\pm$ 8 ms), 100 agents of type TransportRegion2 (mean response time 300 ms  $\pm$ 30 ms), and 8 OrgAgents. Due to the probabilistic nature of our coordination mechanism we performed 30 simulations per experiment to obtain statistically significant results with a 95% confidence interval.

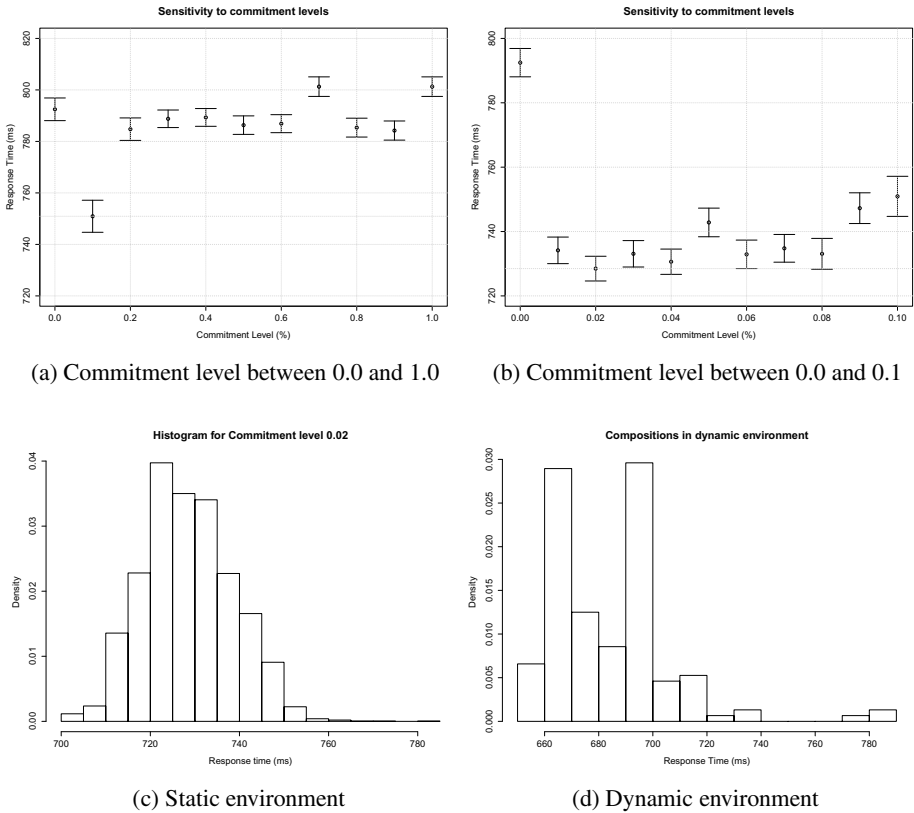
### 4.1 Static Environment

In this experiment, services never fail and have an execution time given by the Poisson probability distribution as describe above. OrgAgents are configured with a commitment level of 0.0. The compositions created by our coordination mechanism had a average response time of 790 ms  $\pm$ 80 ms, basically resembling the nature of the component services. With this values at hand we decided to investigate how different OrgAgent's commitment strategies could influence the created compositions. What is explained in the OrgAgent sensitivity analysis.

**Sensitivity Analysis of OrgAgent’s Commitment Levels** influence the quality of created compositions. A commitment level of 1.0 indicates that the OrgAgent performs the component service selection only once and never changes this selection, it is called a blinded committed agent. A blind committed OrgAgent is unable cope with unexpected changes in the environment, since it never changes its component services.

A cautious agent has a commitment level of 0.0. It revises its commitments every cycle, what can incur in instability in the system, since the agent can keep alternating between component services and never engage with any one.

Figure 1a indicates the average response time of the created compositions by instantiating OrgAgents with different commitment levels, in a static environment.



**Fig. 1.** Sensitivity analysis of OrgAgent’s commitment levels and simulation results

Given the results obtained we found that a commitment level of 0.02 gives the best compositions for this scenario, depicted in Figure 1b. A commitment level of 0.02 indicates that an agent will change its path if the increment in the solution is greater than 2 %.

The quality of the compositions created by the OrgAgents with a commitment level of 0.02 increased by a factor of more than 10%, what is depicted in Figure 1c. It is

interesting to note that the standard variations were also minimized when the OrgAgent had their commitment levels changed to 0.02.

## 4.2 Dynamic Environment

The reality has many sources of dynamic events, such as network failures, failures to keep commitments, even failures to properly predict schedules. To approximate our simulations to the reality we performed simulations with our coordination mechanism in a dynamic environment. We randomly, using the Uniform probability distribution, fail services that are participating in the agent overlay network at a rate of 1 service failure per 10 iterations. The OrgAgents were configured with commitment level of 0.02. The results are depicted in Figure 14.

It can be seen from Figure 14 that the coordination mechanism can still create responsive service compositions, in the presence of component service failures. However we should note that the objective to minimize the standard deviation is hard to achieve, since a number of services are failing and the remaining services can have greater differences in their QoS values, in this case response time.

## 5 Related Work

There is plenty of research about selecting component services to participate in service compositions. These works focus on creating algorithms capable of selecting the best available component services, in terms of QoS. Our work shares the idea that it is possible to improve properties from composite services by selection and bidding to component services at runtime. However, in our research we study the problem of not only selecting the best available services, but also selecting component services that will lead to robust compositions.

The work presented in [6] discusses and evaluate different techniques for component service selection. The simulation results show that the most efficient approach is the proxy-based, followed by the collaborative approach. In the proxy-based approach, all the service invocations go through a proxy that can then, load balance and select the best available services. In the collaborative approach different composite services collaborate, sharing QoS information about component services, to allow a better component service selection.

Our solution shares characteristics with both the proxy-based and collaborative approaches. We focus, however, on the creation of robust compositions instead of focusing only on minimizing a certain QoS metric, such as response time or price.

The work on [7] explicitly focus on creating robust service compositions. The work uses decision theory for dealing with the uncertainty associated with component service providers. It proposes a mechanism for component service selection that explicitly takes the reliability of the created composition into account. The service selection algorithm takes the most critical tasks into account and use service redundancy for these tasks. The algorithm also uses planning techniques to create contingency plans, in the case of component services failures. Another interesting characteristic of the algorithms presented in [7] is the use of service reservation for parts of the composite service.

Our work also takes the robustness of the composition into account when selecting component services. However our approaches differ in how to create robust compositions. Our approach relies on the aggregated information available in the agent overlay network and in advanced reservations of services that will participate in the composition.

## 6 Conclusions and Future Work

The main contribution of this work is to present a decentralized coordination mechanism capable of creating robust service compositions. We have strong indications that our mechanism can be used to create collaborative systems that need to support companies interacting in a supply chain.

We should note that supply chains are quite irregular in reality what was not explored in our current work, but should, definitely, be explored in our future works.

A future research is to evaluate how adding autonomic behaviour to our OrgAgents, for instance, changing the commitment levels at runtime, can influence the quality of the created compositions. Finally we intend to evaluate our coordination mechanism in a real network environment.

## References

1. Sandberg, E., Abrahamsson, M.: The role of top management in supply chain management practices. *International Journal of Retail & Distribution Management* 38(1), 57–69 (2010)
2. Glenn Richey, R., Tokman, M., Dalela, V.: Examining collaborative supply chain service technologies: a study of intensity, relationships, and resources. *Journal of the Academy of Marketing Science* 38(1), 71–89 (2010)
3. Ludwig, H., Nakata, T., Wäldrich, O., Wieder, P., Ziegler, W.: Reliable Orchestration of Resources Using WS-Agreement. In: Gerndt, M., Kranzlmüller, D. (eds.) *HPCC 2006*. LNCS, vol. 4208, pp. 753–762. Springer, Heidelberg (2006)
4. Holvoet, T., Weyns, D., Valckenaers, P.: Patterns of delegate mas. In: *International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 1–9 (2009)
5. Dorigo, M., Caro, G.D., Gambardella, L.M.: Ant algorithms for discrete optimization. *Artificial Life* 5, 137–172 (1999)
6. Ghezzi, C., Motta, A., Panzica La Manna, V., Tamburrelli, G.: QoS Driven Dynamic Binding in-the-Many. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) *QoSA 2010*. LNCS, vol. 6093, pp. 68–83. Springer, Heidelberg (2010), [http://dx.doi.org/10.1007/978-3-642-13821-8\\_7](http://dx.doi.org/10.1007/978-3-642-13821-8_7)
7. Stein, S., Payne, T., Jennings, N.: Robust execution of service workflows using redundancy and advance reservations. *IEEE Transactions on Services Computing* 4(2), 125–139 (2011)

# Pricing Information Goods in Distributed Agent-Based Information Filtering<sup>\*</sup>

Christos Tryfonopoulos<sup>1</sup> and Laura Maria Andreescu<sup>2</sup>

<sup>1</sup> University of Peloponnese, Tripoli, Greece  
trifon@uop.gr

<sup>2</sup> APYDOS, Luxembourg  
landreescu@apydos.com

**Abstract.** Most approaches to information filtering taken so far have the underlying hypothesis of potentially delivering notifications from every information producer to subscribers; this exact information filtering model creates efficiency and scalability bottlenecks and incurs a cognitive overload to the user. In this work we put forward a distributed agent-based information filtering approach that avoids information overload and scalability bottlenecks by relying on approximate information filtering. In approximate information filtering, the user subscribes to and monitors only carefully selected data sources, to receive interesting events from these sources only. In this way, system scalability is enhanced by trading recall for lower message traffic, information overload is avoided, and information producers are free to specialise, build their subscriber base and charge for the delivered content. We define the specifics of such an agent-based architecture for approximate information filtering, and introduce a novel agent selection mechanism based on the combination of resource selection, predicted publishing behaviour, and information cost to improve publisher selection. To the best of our knowledge, this is the first approach to model the cost of information in a filtering setting, and study its effect on retrieval efficiency and effectiveness.

## 1 Introduction

Much information of interest to humans is available today on the Web, making it extremely difficult to stay informed without sifting through enormous amounts of information. In such a dynamic setting, *information filtering (IF)*, also referred to as *publish/subscribe*, *continuous querying*, or *information push*, is equally important to one-time querying, since users are able to subscribe to information sources and be notified when documents of interest are published. This need for *content-based* push technologies is also stressed by the deployment of new tools such as Google Alert. In an IF scenario, a user posts a *subscription* (or *continuous query*) to the system to receive *notifications* whenever certain events of interest take place (e.g., when a document on Special Olympics becomes available). Since in an IF scenario the data is originally highly distributed residing on millions of sites (e.g., with people contributing to blogs,

---

<sup>\*</sup> Part of the work was done while the authors were with Max-Planck Institute for Informatics. The authors would like to thank David Midgley for his comments and suggestions on the economic aspects of this work.

news portals, social networking feeds), a distributed approach seems an ideal candidate for such a setting.

In this work we put forward ABIS (*Agent-Based Information filtering System*), a novel agent-based architecture that supports content-based *approximate information filtering*. While most *exact information filtering* approaches [32][15][14][34][127][8] taken so far have the underlying hypothesis of potentially delivering notifications from every information producer, ABIS relaxes this assumption by monitoring only selected sources that are likely to publish documents relevant to the user's interests in the future. In ABIS, a user subscribes with a continuous query and monitors only the most interesting sources in the system, to receive published documents from these sources only. The system is responsible for managing the user query, discovering new potential sources and moving queries to better or more promising publishers. Approximate IF improves the scalability issues of exact IF by trading recall for lower message traffic, avoids information overload to the user, by allowing him to receive selected notifications from selected publishers, and proves an interesting business model for pricing information goods delivered by information producers. In approximate IF, each information producer might have its own customer base of interested subscribers, and may charge the delivered content by subscription or per item. Notice that this is not possible in the case of exact IF, since information consumers receive *all* matching notifications, from all producers, while to facilitate the distribution of the service, no notion of ownership control and publisher quality is employed. Finally, notice that system throughput and notification latency in exact IF depend heavily on publication size (which is usually large for textual IF). On the other hand, approximate IF is not affected by publication size (as there is no notion of information dissemination at publication time) and offers one-hop latency, since each publisher maintains its own database of subscribers. The interested reader is referred to [36] for an insightful comparison of exact and approximate IF.

As possible application scenarios for ABIS consider the case of news filtering (but with the emphasis on information quality rather than timeliness of delivery) or blog filtering where users subscribe to new posts. Not only do these settings pose scalability challenges, but they would also incur an information avalanche and thus cognitive overload to the subscribed users, if the users were alerted for each and every new document published at any source whenever this matched a submitted continuous query. Our approximate IF approach ranks sources, and delivers matches only from the best ones, by utilising novel publisher selection strategies. These strategies take into consideration the quality of the information publisher, based on per-publisher statistics, and the price of information as this is set by the publisher. Despite the utilisation of a *Distributed Hash Table* (DHT) [31] to maintain publisher statistics, notice that our architecture can also be realised in other settings, like a single coordinator agent monitoring a number of distributed sources, or a cloud-based multi-agent system providing an alerting service.

To the best of our knowledge, this is the first approach that aims at connecting system efficiency and effectiveness with the cost component, and puts economic modelling in the picture of distributed IF. In the light of the above, the contributions presented in this work are threefold:

- We define an agent-based architecture and its related protocols to support *approximate* IF functionality in a distributed multi-agent environment. This is the first approach to model approximate IF in an agent-based setting.
- We show that traditional resource selection strategies are not sufficient in approximate IF, and devise a *novel* method to rank publishers according to their expertise, their predicted publishing behavior (based on time-series analysis of IR metrics) and the price of the information goods they publish. This method allows us to achieve high recall, while monitoring only a small number of publishers.
- We study the effect of introducing a price component in an IF setting and experimentally demonstrate that price of information is a key element, that may have an significant effect on recall observed by the subscribers. Our modelling utilises concepts such as correlation between the quality/expertise of the publisher and the price it charges for information goods, computation of this price depending on the demand, and charging agents for utilisation of resources such as local agent and network utilisation.

In previous work, we have compared exact and approximate information filtering in [36,44], applied approximate IR and IF to the digital library domain [46], and investigated different time series analysis methods [45]. The current paper extends the core ideas behind approximate IF in a multi-agent architecture, and emphasises the price component and its effect on system effectiveness and efficiency.

The rest of the paper is organised as follows. Related work is discussed in Section 2. Section 3 presents the ABIS architecture, implemented services and agent protocols, while Section 4 introduces our agent selection method. Experimental results are presented in Section 5, and Section 6 concludes this paper.

## 2 Related Work

In this section we discuss related work in the context of pricing information goods in agent-based systems, and IF in distributed (e.g., multi-agent, P2P) environments.

### 2.1 Pricing of Information in Agent-Based Models

Information has the property of non-rivalrous consumption, contrary to other goods such as cars and apples that need to be produced individually in order to be consumed individually, and once purchased are removed from the market for subsequent buyers.

One distinct feature of information goods is that they have large fixed costs of production, and small variable costs of reproduction, which makes value-based more appropriate than cost-based pricing [40]. Different consumers may have radically different values for the same information good, so techniques for differential pricing become very important. The best known form of differential pricing is called quality discrimination or versioning [39]. Using versioning, the producer will divide the consumers into different groups according to their willingness to pay, or choose the price of the versions and their compelling features to induce the consumers to “self select” into appropriate categories [40].

Different pricing strategies are relevant for different disciplines and applications. It is important to be aware of the fact that not all services will necessarily be provided to all users. In [19] a thorough analysis on database pricing strategies is carried out, and different strategies (such as connect-time pricing, per-record charge, and value-based pricing) are identified and compared. In [21] complex adaptive systems are used to analyse pricing decisions in an industry with products that can be pirated, while [7] looks into pricing models for information from Bloomberg, Reuters and Bridge, and presents the advantages of subscription, two-tier pricing schemes (flat fee for subscription and then a small charge for every item ordered) and n-tier pricing schemes.

Available research also offers a variety of models that can be used to study the cost of transactions between agents in a market model. [18] models the cost of a product on two dimensions: a price for the product itself and a transaction cost. The transaction cost also has two components: a component that is correlated to the amount of data that needs to be transported through the network and a component that is based upon changes in quantities ordered. Along the same line, [38,41] discuss incentives for different pricing schemes for information goods, distinguishing between pure competitive markets (several producers of an identical commodity) and markets where producers have market power. In [13] the problem of using services provided from other agents is considered, while [11] presents a case-study on file-transfer and focuses on the utilisation factor of the links between agents.

## 2.2 Distributed Information Filtering

Research on distributed processing of continuous queries has its origins in SIENA [4], and extensions on the core ideas of SIENA, such as DIAS [22] and P2P-DIET [23,17].

With the advent of DHTs such as CAN, Chord and Pastry, a new wave of publish/subscribe systems has appeared. Scribe [30], Hermes [27], HYPER [42], Meghdoot [15], PeerCQ [14], and many others [36,14,8,34] utilised a DHT to build a content-based system for processing continuous queries.

Many systems also employed an IR-based query language to support information filtering on top of structured overlay networks have been deployed. DHtrie [34], Ferry [43], and [2], extended the Chord protocol [31] to achieve exact information filtering functionality and applied document-granularity dissemination to achieve the recall of a centralised system. In the same spirit, LibraRing [33] presented a framework to provide information retrieval and filtering services in two-tier digital library environments. Similarly, pFilter [32] used a hierarchical extension of the CAN DHT [29] to store user queries and relied on multi-cast trees to notify subscribers. In [1], the authors show how to implement a DHT-agnostic solution to support prefix and suffix operations over string attributes in a publish/subscribe environment.

Information filtering and retrieval have also been explored in the context of multi-agent systems. In [28] the design of a distributed multi-agent information filtering system called D-SIFTER is presented, and the effect of inter-agent collaboration on filtering performance is examined. In [24], a peer-to-peer architecture for agent-based information filtering and dissemination, along with the associated data models and languages for appropriately expressing documents and user profiles is presented. Finally,



the MAWS system [16] utilises mobile agents to reduce the volume of irrelevant links returned by typical search engines.

Query placement, as implemented in exact information filtering approaches such as [132,134], is deterministic, and depends upon the terms contained in the query and the hash function provided by the DHT. These query placement protocols lead to filtering effectiveness of a centralised system. Compared to a centralised approach, [132,134] exhibit scalability, fault-tolerance, and load balancing at the expense of high message traffic at publication time. In ABIS however, only the most promising agents store a user query and are thus monitored. Publications are only matched against its local query database, since, for scalability reasons, no publication forwarding is used. Thus, in the case of approximate filtering, the recall achieved is lower than that of exact filtering, but document-granularity dissemination to the network is avoided.

### 3 Services and Protocols in ABIS

In this section we present the services implemented in ABIS and the respective protocols that regulate agent interactions.

#### 3.1 Types of Services

Within the multi-agent system we can distinguish between three types of services: a directory service, a publication service and subscription service. All agents in ABIS implement the directory service, and one or both of the publication and subscription service, depending whether they want to act as information producers, consumers or both.

**Directory Service.** The directory service manages aggregated statistical meta-information about the documents that are offered by publishers (i.e., aggregated statistical information for terms, prices per document). The role of this service is to serve as a global meta-data index about the documents and the prices available on the market. This index is partitioned among all agents in ABIS and is utilised by the subscribers to determine which publishers are promising candidates to satisfy a given continuous query in the future. There are different alternatives to implementing this type of directory, ranging from centralised solutions that emphasise accuracy in statistics and rely on server farms, to two-tier architectures. In our approach, we utilise a distributed directory of agents organised under a Chord DHT [31] to form a conceptually global, but physically distributed directory. The directory manages the statistics provided by the publishers in a scalable manner with good properties regarding system dynamics (e.g., churn). The DHT is used to partition the term space, such that every agent is responsible for the statistics of a randomised subset of terms within the directory. Hence, there is a well defined directory agent responsible for each term (through the DHT hash function).

**Publication Service.** The publication service is implemented by information producers (e.g., digital libraries or agents with local crawlers that perform focused crawling at portals of their interest) or users that are interested in selling their content. The publishers do not a priori know how much a subscriber is willing to pay for information

from his domain. In an ideal model the publishers will adjust the price according to the demand from the market: when a publisher is overloaded with requests, he would increase the price for the information he is offering. An agent implementing only the publication service creates meta-data for the resources it stores and uses the directory service to offer them to the rest of the network.

Each publisher exposes its content to the directory in the form of per-term statistics about its local index. These posts contain contact information about publishers, together with statistics to calculate quality measures and prices for a given term (e.g., frequency of occurrence). Typically, such statistics include quality measures to support the publisher ranking procedure carried out by subscribers, and are updated after a certain number of publications occurs. Finally, publishers are responsible for locally storing continuous queries submitted by subscribers and matching them against new documents they publish.

**Subscription Service.** The agents implementing the subscription service are information consumers, which subscribe to publications and receive notifications about resources that match their interests. The goal of the subscribers is to satisfy their long-term information needs by subscribing to publishers that will publish interesting documents in the future. A subscriber has access to all prices set by publishers for certain resource types through the directory service, and the subscribers are free to choose the best offer from the market that suits their needs and budget. To do so, subscribers utilise directory statistics to score and rank publishers, based on appropriate publisher selection and behaviour prediction strategies, as well as on the actual price of the requested item and the budget of the agent as we will discuss in following sections. To follow the changes in the publishing behaviour of information producers, subscribers periodically re-rank publishers by obtaining updated statistics from the directory, and use the new publisher ranking to reposition their continuous queries.

### 3.2 The ABIS Protocols

All agents implementing the aforementioned services follow a specific protocol to facilitate message exchange in a scalable way. Below we describe the protocols that facilitate agent interaction, for each one of the described services.

**The Directory Protocol.** The directory service manages aggregated information about each agent's local knowledge in a compact form. Every agent is responsible for the statistics for a randomised subset of terms within the directory. To keep the statistics up-to-date, each agent distributes per-term summaries of its local index along with its contact information. For efficiency reasons, these messages are piggy-backed to DHT maintenance messages and batching is used.

To facilitate message sending between agents we will use the function  $\text{SEND}(msg, I)$  to send message  $msg$  to the agent responsible for identifier  $I$ . Function  $\text{SEND}()$  is similar to the Chord function  $\text{LOOKUP}(I)$  [31], and costs  $O(\log N)$  overlay hops for a network of  $N$  agents. In ABIS, every publisher uses  $\text{POST}$  messages to distribute per-term statistics. This information is periodically updated (e.g., every  $k$  time units or every  $k$  publications) by the publisher agent, in order to keep the directory information as up-to-date

as possible. Let us now examine how a publisher agent  $P$  updates the global directory. Let  $T = \{t_1, t_2, \dots, t_k\}$  denote the set of all terms contained in all document publications of  $P$  occurring after the last directory update. For each term  $t_i$ , where  $1 \leq i \leq k$ ,  $P$  computes the maximum frequency of occurrence of term  $t_i$  within the documents contained in  $P$ 's collection ( $t f_i^{max}$ ), the number of documents in the document collection of  $P$  that  $t_i$  is contained in ( $d f_i$ ), and the size of the document collection  $cs$ . Having collected the statistics for term  $t_i$ ,  $P$  creates message  $\text{POST}(id(P), ip(P), t f_i^{max}, d f_i, cs, t_i)$ , where  $id(P)$  is the identifier of agent  $P$  and  $ip(P)$  is the IP address of  $P$ .  $P$  then uses function  $\text{SEND}()$  to forward the message to the agent responsible for identifier  $H(t_i)$  (i.e., the agent responsible for maintaining statistics for term  $t_i$ ). Once an agent  $D$  receives a  $\text{POST}$  message, it stores the statistics for  $P$  in its local post database to keep them available on request for any agent.

Finally, notice that the directory service does not have to use Chord, or any other DHT; our architecture allows for the usage of any network structure given that the necessary information (i.e., the per-agent IR statistics) is made available through appropriate protocols to the rest of the services.

**The Subscription Protocol.** The subscription service is implemented by agents that want to monitor specific information producers. This service is critical since it is responsible for selecting the publishers that will index a query. This procedure uses the directory service to discover and retrieve the publishers that have information on a given topic. Then a ranking of the potential sources is performed and the query is sent to top- $k$  ranked publishers. Only these publishers will be monitored for new publications.

Let us assume that a subscriber agent  $S$  wants to subscribe with a *multi-term query*  $q$  of the form  $t_1 t_2 \dots t_k$  with  $k$  distinct terms. To do so,  $S$  needs to determine which publishers in the network are promising candidates to satisfy the continuous query with appropriate documents published in the future. This publisher ranking can be decided once appropriate statistics about data sources are collected from the directory, and a ranking of the publishers is calculated based on the agent selection strategy described in Section 4.

To collect statistics about the data publishers,  $S$  needs to contact all directory agents responsible for the query terms. Thus, for each query term  $t_i$ ,  $S$  computes  $H(t_i)$ , which is the identifier of the agent responsible for storing statistics about other publishers that publish documents containing the term  $t_i$ . Subsequently,  $S$  creates message  $\text{COLLECTSTATS}(id(S), ip(S), t_i)$ , and uses the function  $\text{SEND}()$  to forward the message in  $O(\log N)$  hops to the agent responsible for identifier  $H(t_i)$ . Notice that the message contains  $ip(S)$ , so its recipient can directly contact  $S$ .

When an agent  $D$  receives a  $\text{COLLECTSTATS}$  message asking for the statistics of term  $t_i$ , it searches its local post store to retrieve the agent list  $L_i$  of all posts of the term. Subsequently, a message  $\text{RETSTATS}(L_i, t_i)$  is created by  $D$  and sent to  $S$  using its IP found in the  $\text{COLLECTSTATS}$  message. Once  $S$  has collected all the agent lists  $L_i$  for the terms contained in  $q$ , it utilises an appropriate scoring function  $score(n, q)$  to compute a agent score with respect to  $q$ , for each one of the agents  $n$  contained in  $L_i$ . Based on the score calculated for each publisher, a ranking of publishers is determined and the highest ranked agents are candidates for storing  $q$ .

Subsequently,  $S$  selects the highest ranked publishers that will index  $q$ . Thus, only publications occurring *at those publishers* will be matched against  $q$  and create appropriate notifications. Agents publishing documents relevant to  $q$ , but not indexing  $q$ , will not produce any notification for it, since they are not aware of  $q$ . Since only selected agents are monitored for publications, the publisher ranking function becomes a critical component, which will determine the final recall achieved. This ranking function is discussed in detail in the next section.

Once the agents that will store  $q$  have been determined,  $S$  constructs message INDEXQ( $id(S), ip(S), q$ ) and uses the IP addresses associated with the agent to forward the message to the agents that will store  $q$ . When a publisher  $P$  receives a message INDEXQ containing  $q$ , it stores  $q$  using a local query indexing mechanism such as [35].

Filtering and agent selection are dynamic processes, therefore a periodic query repositioning, based on user-set preferences, is necessary to adapt to changes in publisher's behaviour. To reposition an already indexed query  $q$ , a subscriber would re-execute the subscription protocol, to acquire new publisher statistics, compute a new ranking, and appropriately modify the set of agents indexing  $q$ .

**Publication and Notification Protocol.** The publication service is employed by users that want to expose their content to the network. A publisher  $P$  utilises the directory to update statistics about the terms contained in the documents it publishes. All queries that match a published document produce appropriate notifications to interested subscribers.

According to the above, the procedure followed by  $P$  at publication time is as follows. When a document  $d$  is published by  $P$ , it is matched against  $P$ 's local query database to determine which subscribers should be notified. Then, for each subscriber  $S$ ,  $P$  constructs a notification message NOTIFY( $id(P), ip(P), d$ ) and sends it to  $S$  using the IP address associated with the stored query. If  $S$  is not on-line at notification arrival, then  $P$  utilises function SEND() to send the message through the DHT, by using the  $id(S)$  also associated with  $q$ . In this way,  $S$  will receive the message from its successor upon reconnection. Notice that agents publishing documents relevant to a query  $q$ , but not storing it, will produce no notification.

## 4 Publisher Ranking Strategy

To select which publishers will be monitored, the subscription protocol of Section 3.2 uses a scoring function to rank publisher agents according to quality and price. In this section we quantify these concepts, and give the rationale between our choices.

### 4.1 Quality vs Price

The publisher ranking strategy is a critical component, since it decides which publishers will store a continuous query. Contrary to exact information filtering, where the system would deliver all events matching a subscription, in approximate information filtering a subscriber registering with a continuous query  $q$  has to decide which publishers are the most promising candidates for satisfying  $q$ , as he will receive events that match  $q$  only from those publishers.

To make an informed selection on the publishers, a subscriber agent ranks them based on a combination of publisher quality and price quoted for the specific type of information. This combination describes the benefit/cost ratio and allows the subscriber to assign a score to every publisher. Empirical studies have shown that price and quality are the two key determinants of the consumer's choice to buy or not a product [9]. The score for each publisher is computed as follows:

$$\text{score}(P, q) = (1 - \alpha) \cdot \text{quality}(P, q) - \alpha \cdot \text{price}(P, q) \quad (1)$$

In Equation 1,  $\text{quality}(P, q)$  denotes how relevant  $P$  is to the continuous query  $q$ , while  $\alpha$  is a tunable parameter that affects the balance between the importance of price over quality. The  $\text{price}(P, q)$  component in Equation 1 refers to the price a publisher is quoting for published documents matching a continuous query  $q$ . The publishers are computing the price on demand according to their popularity and the popularity of their documents. The price has the same domain as quality for allowing their use within the same formula, and is recomputed whenever the popularity of the publisher changes (i.e., a new continuous query is stored at the publisher). In the experimental section we study the price in different scenarios, and show the effect on recall when the price choice is (i) random, (ii) strongly correlated with quality, and (iii) partially correlated with quality.

## 4.2 Calculating Publisher Quality

To assess the quality of the information producer, as required in Equation 1, the subscriber uses a combination of *resource selection* and *behaviour prediction* as shown below:

$$\text{quality}(P, q) = \gamma \cdot \text{sel}(P, q) + (1 - \gamma) \cdot \text{pred}(P, q) \quad (2)$$

The functions  $\text{sel}(P, q)$  and  $\text{pred}(P, q)$  are scoring functions based on resource selection and publication prediction methods respectively that assign a score to a publisher  $P$  with respect to a query  $q$ . The tunable parameter  $\gamma$  affects the balance between authorities (high  $\text{sel}(P, q)$  scores) and agents with potential to publish matching documents in the future (high  $\text{pred}(P, q)$  scores). Based on these scores, a score representing the quality of a publisher is determined.

To show why an approach that scores publishers based only on resource selection is not sufficient, and to give the intuition behind publisher behaviour prediction, consider the following example. Assume an agent  $A_1$  that is specialised and has become an authority in sports, but publishes no relevant documents any more. Another agent  $A_2$  is not specialised in sports, but is currently crawling a sports portal, and publishing documents from it. Imagine a user who wants to stay informed about the 2011 Special Olympics, and subscribes with the continuous query *2011 Special Olympics*. If the ranking function solely relies on resource selection, agent  $A_1$  would always be chosen to index the user's query (since it was a sports authority in the past), despite the fact that it no longer publishes sports-related documents. On the other hand, to be assigned a high score by the ranking function, agent  $A_2$  would have to specialise in sports – a long procedure that is inapplicable in a filtering setting which is by definition dynamic. The fact that resource selection alone is not sufficient is even more evident in the case of news items. News items have a short shelf-life, making them the worst candidate for slow-paced resource selection algorithms.

**Behaviour Prediction.** To predict the publishing behaviour of an agent, we model IR statistics maintained in the distributed directory as time-series data and use statistical analysis tools [5] to model publisher behaviour. Time-series techniques predict future values based on past observations and differ in (i) their assumptions about the internal structure of the time series (e.g., whether trends and seasonality are observed) and (ii) their flexibility to put more emphasis on recent observations. Since the IR statistics we utilise exhibit trends, for instance, when agents successively crawl sites that belong to the same/different topics, or, gradually change their thematic focus, the employed time series prediction technique must be able to deal with trends. Furthermore, in our scenario we would like to put more emphasis on an agent’s recent behaviour and thus assign higher weight to recent observations when making predictions about future behaviour. For the above reasons we chose *double exponential smoothing* (DES) as our prediction technique, since it supports decreasing weights on observed values and allows for trend in the series of data.

The function  $pred(P, q)$  returns a score for a publisher  $P$  that represents the likelihood of publishing documents relevant to query  $q$  in the future. Using the DES technique described above, two values are predicted. Firstly, for all terms  $t$  in query  $q$ , we predict the value for  $df_{P,t}$  (denoted as  $df_{P,t}^*$ ), and use the difference (denoted as  $\delta(df_{P,t}^*)$ ) between the predicted and the last value obtained from the directory to calculate the score for  $P$  (function  $\delta()$  stands for difference). Value  $\delta(df_{P,t}^*)$  reflects the number of relevant documents that  $P$  will publish in the next period. Secondly, we predict  $\delta(cs_P^*)$  as the difference in the collection size of agent  $P$  reflecting the agent’s overall expected future publishing activity. We thus model two aspects of the publisher’s behaviour: (i) its potential to publish relevant documents in the future (reflected by  $\delta(df_{P,t}^*)$ ), and (ii) its overall expected future publishing activity (reflected by  $\delta(cs_P^*)$ ). The time series of IR statistics that are needed as an input to our prediction mechanism are obtained using the distributed directory. The predicted behaviour for agent  $P$  is quantified as follows:

$$pred(P, q) = \sum_{t \in q} \log(\delta(df_{P,t}^*) + \log(\delta(cs_P^*) + 1) + 1) \quad (3)$$

In the above formula, the publishing of relevant documents is more accented than the dampened publishing rate. If an agent publishes no documents at all, or, to be exact, the prediction of  $\delta(df_{P,t}^*)$  or  $\delta(cs_P^*)$  is 0 then the  $pred(P, q)$  value is also 0. The addition of 1 in the log formulas yields positive predictions and avoids  $\log(0)$ .

**Resource Selection.** The function  $sel(P, q)$  returns a score for a publisher  $P$  and a query  $q$ , and is calculated using standard resource selection algorithms from the IR literature, such as tf-idf based methods, CORI or language models (see [26] for an overview). Using  $sel(P, q)$  we identify authorities specialised in a topic, which, as argued above, is not sufficient for our IF setting. In our implementation we use an approach based on document frequency ( $df$ ), and maximum term frequency ( $t f^{max}$ ). The values of  $sel(P, q)$  for all query terms  $t$  are aggregated as follows:

$$sel(P, q) = \sum_{t \in q} \beta \cdot \log(df_{P,t}) + (1 - \beta) \cdot \log(t f_{P,t}^{max}) \quad (4)$$

The value of the parameter  $\beta$  can be chosen between 0 and 1 and is used to emphasise the importance of  $df$  versus  $tf^{max}$ . Experiments with resource selection have shown that  $\beta$  should be set around 0.5.

### 4.3 Economic Modelling of ABIS

In this section we analyse the economic modeling of ABIS and review the basic assumptions and expectations from such a modelling.

Usefulness of the information goods received by a subscriber, is a qualitative criterion, that is difficult to model. In ABIS we model usefulness by matching interest, i.e., by assuming that all received documents relevant to the requested topic are useful to the subscriber, and do not discuss issues such as novelty and coverage of information, or user effort. In our modelling, after a subscriber acquires a history of transactions with certain publishers it develops an affect for some of the publishers. Affect can be modelled in various ways, depending on the task at hand, and can be either positive or negative (as in e.g., [10] where affect causes a “preference shock” to the consumers that buy only from a certain manufacturer). In ABIS, an information consumer does not know the quality of the information goods, but he uses the affect developed from previous transactions to approximate it. Subsequently, he compares the values of information quality to the expected values and update its affect [25].

The costs in ABIS are results of agent actions [18], such as transactions (e.g., un-subscribing from a publisher and subscribing to another, changing a submitted query), network communication, and use of common infrastructure (e.g., the directory service). Since each agent may play a dual role both as a publisher and a subscriber, it will naturally try to maximise his revenue, and the utility of the received resources, while minimising expenses that occur due to publication or subscription actions. In general, the information market in the ABIS system is not a pure competitive market [38] since the subscribers do not know in advance the exact quality of the information they are buying. The ABIS system resembles the modelling of a team of sales people [37]. In this model agents would try to collaborate with others in order to get their expertise for a (cross/up) sale. After deciding which agents to collaborate with, it will be possible to model the gap between the initial expectations and the actual actions of the agent. In [12] it is shown that this gap is smaller in a competitive relationship compared to that of a cooperative relationship. As in many cooperative environments each agent usually retains its connections with the other agents, while also being free to explore new mutually beneficial connections.

The main goal of this agent-based modelling is to study the influence of the cost component on the quality of received resources, study the interactions between agents that are trying to maximise the benefits of information flow, and gain insights about the activity and the behaviour of the publishers and subscribers.

## 5 Experimental Evaluation

In this section we present our findings regarding the introduction of cost in information goods, and how it affects the effectiveness of an information filtering system. We study

the behaviour of the ABIS system using different publishing scenarios, while varying the correlation between price, quality and customer demand.

## 5.1 Experimental Setup

To conduct each experiment described in the next sections the following steps are executed. Initially the network is set up and the underlying Distributed Hash Table (DHT) is created. Then, subscribers use the ranking function based on resource selection, predicted publishing behaviour and cost of information to decide which are the best publishers subscribe to. Subsequently, they utilise the subscription protocol described in detail in Section 3.2 to subscribe to the selected publishers. Once the queries are stored, the documents are published to the network and at certain intervals (called *rounds*) queries are repositioned, and new documents are published.

**Evaluation Metrics.** To measure the effect of cost in information filtering, and compare between cases of IF with and without monetary flow in ABIS, we utilise the following metrics:

- **Messages.** We measure the number of directory, subscription and notification messages in the system to perform the filtering task at hand.
- **Recall.** We measure recall by computing the *ratio* of the total number of notifications received by subscribers to the total number of published documents matching subscriptions. In experiments we consider the *average recall* computed over *all* rounds (i.e., for the complete experiment).
- **Ranking.** We use an extension of Spearman’s footrule distance to compare rankings of publishers calculated by subscribers. This metric allows us to compare two different publisher rankings by calculating the distance between the elements in two ranking lists. In our extension of Spearman’s metric, if an element from list A is not present in list B, it is considered as being in the last available position in B.

**System Parameters.** There is a number of system parameters that regulate agent behavior and had to be determined and set. Due to space considerations the procedure and experimentation of finding the optimal values for these parameters is omitted and the interested reader is referred to [44]. One such key parameter is the percentage of the available publishers that a subscriber can follow. When all publishers are monitored, then recall is 100%, and our approach degenerates to exact filtering. Exact information filtering will always give the best result with regard to recall, but also incur high message traffic to the system and cost to the subscriber. On the other hand, when using a random selection of publishers, monitoring  $k\%$  of publishers will typically result in recall around  $k\%$ . To achieve higher than  $k\%$ , the publisher ranking strategy presented in Section 4 is employed. Additionally, parameter  $\gamma$  in Equation 2 controls the balance between resource selection and behavior prediction; a value of  $\gamma$  close to 0 emphasises behavior prediction, while values close to 1 emphasise resource selection. In previous work [44], we determined that monitoring around 10% of publishers, and setting the value of  $\gamma$  to 0.5 represents a good trade-off between recall and message overhead. Additionally, both coefficients used in the double exponential smoothing were set to 0.5, as in [45].



Finally, for deciding the budget per agent, we relied to studies on budget distribution and spending for a variety of cases, ranging from family budgets to consumer budgets [20,6]. The main conclusions drawn from these studies are that (i) budget distribution follows a power law, with a small percentage of families/consumers having a high yearly budget, and a large percentage of the families being in the (long) tail of the distribution, with a low budget, and (ii) the percentage of the income spend on (information) goods does not vary with the budget. According to [3] and the above remarks, we divided the agents into three classes: low budget agents, average budget agents, and high budget agents. 60% of the agents are part of the low budget class, 30% of the agents have an average budget, and 10% belong to the high budget agent class. Subsequently, we experimentally computed a budget that would allow the information consumers to subscribe to all top-k publishers, and allowed the low budget agents to have 60%, the medium budget agents to have 80% and the high budget agents to have 120% of this ideal budget.

**Documents and Queries.** The document collection contains over 2 million documents from a focused Web crawl categorised in one of ten categories: *Music, Finance, Arts, Sports, Natural Science, Health, Movies, Travel, Politics, and Nature*. The overall number of corpus documents is 2,052,712. The smallest category consists of 67,374 documents, the largest category of 325,377 documents. The number of distinct terms after stemming adds up to 593,876.

In all experiments, the network consists of 1,000 agents containing 300 documents each in their initial local collection. Each agent is initialised with 15% random, 10% non-categorised, and 75% single category documents, resulting in 100 specialised agents for each category. Using the document collection, we construct continuous queries containing two, three or four query terms. Each of the query terms selected is a strong representative of a document category (i.e., a frequent term in documents of one category and infrequent in documents of the other categories). Example queries are *music instrument, museum modern art, or space model research study*.

## 5.2 Varying the Price-Quality Correlation

In this experiment we aimed at observing the behaviour of the system when we varied the correlation between the price and quality of publisher. In this experiment, 100% correlation between price and publisher quality, means that the better the quality of the publisher, the higher the price it charges for publications. In this case quality can be easily forecasted and consumers will know that the information goods will be expensive but useful to them. The other extreme in this correlation is when prices have no (0%) correlation with quality, and are chosen randomly. In the case of 75% (respectively 50% and 25%) correlation between price and quality, we modelled the correlation, as the likelihood that a publisher sells 25% (respectively 50% and 75%) of the information goods underpriced up to 20% of the initial value, and 75% (respectively 50% and 25%) overpriced up to 10% of the initial value.

In Figure 1(a) the achieved recall of the system for different values of  $\alpha$  and different price-quality correlations is shown. The first observation emanating from this graph is that the introduction of a price component reduces the observed recall of the system (notice that recall has the highest values for  $a = 0$ , i.e., not pricing involved in the

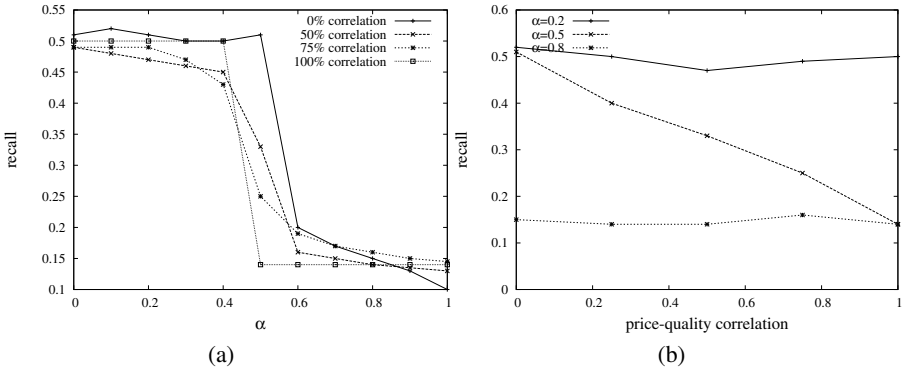


Fig. 1. Recall against  $\alpha$  and price-quality correlations

ranking of publishers). This is an important result, showing that when information publishers charge for information, consumers trade quality subscriptions for cheaper ones. Notice also that in all cases of price-quality correlation, recall is retained high, as long as it plays the most important role in the ranking ( $\alpha < 50\%$ ). This was also expected, since when price is of importance, consumers will choose cheaper publishers, leading to a reduction in the observed recall. Additionally, when the price is the only ranking criterion for information consumers ( $\alpha = 1$ ), recall is close to that of a random choice of publishers (remember that agents monitor only 10% of publishers in the system).

Another observation is that the correlation between the price set by the publisher and its actual quality, plays (as expected) an important role only when price and quality are equally important. When one of the two components becomes dominant in the ranking function, it outweighs the effect of the other. This is also in accordance with our expectations, since when price comes into the picture, quality is sacrificed to reduce costs, or increase the received publications. These observations are best shown in Figure 1(b) where recall for varying the value of price-quality correlation and three different values of  $\alpha$  is presented. Finally, notice that the small variation in the observed recall and agent behavior between different price-quality correlations, is also partly due to the modelling of ABIS as a closed system, where monetary flow is limited through the budgets of the agents, since no new wealth is produced.

Figure 2(a) shows the difference in publisher rankings when varying  $\alpha$  and for different price-quality correlations. The difference in the ranking of publishers is measured using an extension of Spearman’s footrule metric. To produce a point in the graph we compare the list of publishers ranked by a subscriber when no cost is introduced, and the same list when we introduce cost with the given value for  $\alpha$ . This is performed for all the subscribers in the system, and the average metric is calculated. The first observation emanating from the graph is that for  $\alpha = 0$ , all price-quality correlations are naturally zero, since no cost is associated with the information goods, and thus the lists compared are identical. Another observation, is that when  $\alpha$  is increasing, i.e., price becomes more important in the ranking process, Spearman’s metric increases too, as publishers with high quality get lower positions in the ranking, while publishers with lower quality (but cheaper) are ranked high. Additionally, notice that the difference in

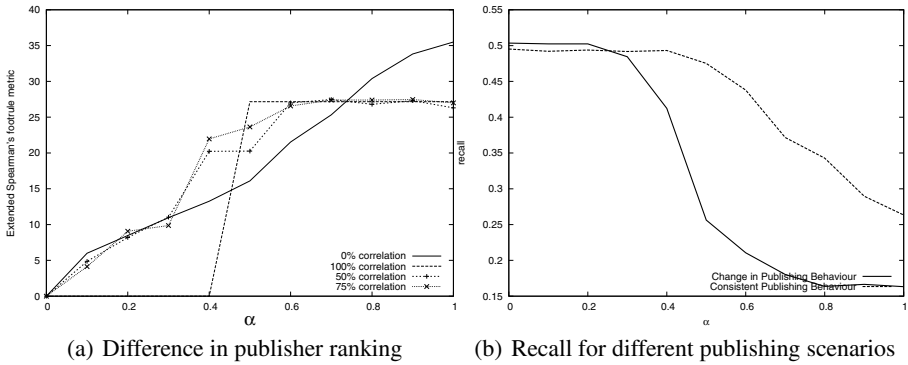


Fig. 2. Publisher rankings and behaviors against  $\alpha$

recall observed in Figure 1 is also partially depicted here as difference in the ranking of the information producers. Finally, when the price of a publisher has no association with the quality of its documents (random price setting), the difference in the ranking of publishers is about 40% higher, than the case of price and quality being correlated (leftmost points in the graph).

### 5.3 Varying the Publishing Behaviour

In this section we look into recall and how this is affected by different publishing behaviours, when varying the importance of information cost in the system. The publishing behavior of agents is modelled using two different scenarios: *consistent publishing* and *category change*, that represent the two extremes of publishing behaviours.

**Consistent publishing.** In the consistent publishing scenario, the publishers maintain their specialisation, by disregarding market conditions, even if this results in very low revenue.

**Category change.** In the category change scenario, publishers change their topic of specialisation over time based on changes in consumer behaviour, revenue and market conditions. In this scenario, a publishing agent initially publishes documents from one category, and switches to a different category after a number of rounds, to simulate changes in portfolio contents or business strategies.

As we can observe in Figure 2(b), in both scenarios, the system reaches the highest recall when no price component is added (leftmost point in the graph), while as cost of information gains importance, the observed recall drops, since agents seek for cheaper publishers. A second observation, is that the consistent publishing scenario is less affected by the introduction of the price component, and achieves significantly higher recall as  $\alpha$  increases. This happens because in this scenario, publishers have build up an expertise, and since this expertise is not changed, the quality component increases, leading to the ranking of these publishers high in the list. Contrary, when publishers change their area of focus, the observed recall of the system falls, since subscribers are

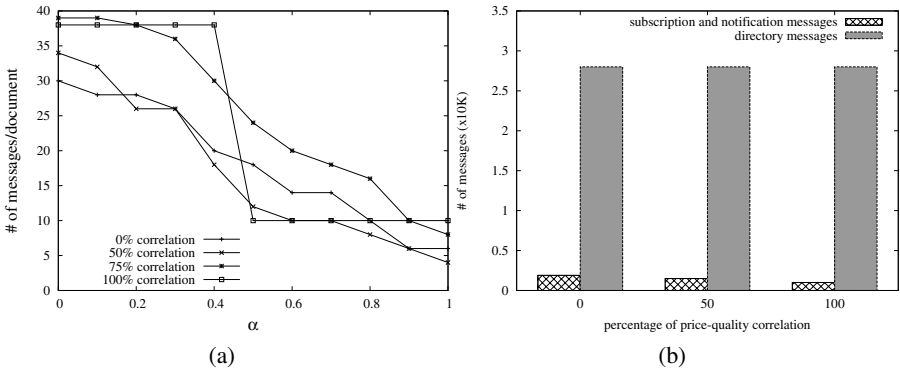


Fig. 3. Message traffic against  $\alpha$  and price-quality correlations

not able to correlate price and quality for the publishers (see also Section 5.2). This observation resembles the case of companies that have to allocate a marketing budget to convince consumers about a new product. Here, the publishers change their publishing behaviour to sell a new product (i.e., a new topic) and the old customers walk away, resulting in recall reduction.

### 5.4 System Performance

In this series of experiments we targeted the system performance in terms of message traffic. In Figure 3(a) we present the message traffic per agent (subscription and notification messages) incurred in the system when varying  $\alpha$ . In this graph we see that the number of messages per agent is reduced, as the price component is emphasised. This can be explained as follows. As subscribers utilise the price component to rank publishers, they choose publishers of lower quality and price. This, as we also observed in the previous sections, results in a reduction in the observed recall, since subscribers receive less notifications, as they subscribe to non-expert publishers. On the other hand expert publishers have a smaller customer base, and are thus forced to notify fewer subscribers. The interested reader is also referred to [44], where we demonstrated that recall and message traffic are interconnected.

Figure 3(b) demonstrates the total amount traffic observed in the system, and how this traffic is split in the various message categories, as the price-quality correlation is varied. As expected the directory traffic dominates the messaging load of the system, as necessary messages with agent statistics and prices are disseminated. Notice that directory traffic is not affected by the correlation between price and quality, since the publishers are responsible for updating their publication statistics and prices, regardless of the size of their customer base. Finally, notice that the number of subscription and notification messages is slightly affected from the price-quality correlation, as quality publishers widen their customer base with more subscribers.

### 5.5 Summary of Results

The experiments presented in this section, show the behaviour of a IF system when a price component is introduced in the selection process of publishers. To the best of

our knowledge these are the first results that connect recall and message traffic with the cost component, and put economic modelling in the picture of distributed IF. Our findings show that when introduced, the price component affects the average recall of the system, since it outweighs quality in the ranking of publishers. Our experiments showed that the price component should participate in publisher ranking with no more than 10-20% of the total score, to avoid loss in observed recall. Additionally, we showed that adding a price component in such a system, reduces message traffic, as (i) this is directly connected to recall, and (ii) agents avoid costly actions like frequent document publications, and query repositioning. Thus, pricing information goods in distributed settings should be carried out carefully, to avoid user dissatisfaction due to reduced flow of relevant documents.

## 6 Conclusions and Outlook

In this work we have defined an architecture and the associated protocols to achieve distributed agent-based approximate IF, and introduced a novel publisher selection mechanism that ranks monitored information producers according to their expertise, their predicted publishing behavior (based on time-series analysis of IR metrics) and the price of the information goods they publish. We have showed that approximate IF is an efficient and effective alternative to the exact IF paradigm, as it manages to trade recall for low message overhead, while providing an interesting business model. We are currently porting our implementation to PlanetLab to conduct more extensive experimentation, and adding new features such as monitoring of monetary flow.

## References

1. Aekaterinidis, I., Triantafillou, P.: PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. In: ICDCS (2006)
2. Bender, M., Michel, S., Parkitny, S., Weikum, G.: A Comparative Study of Pub/Sub Methods in Structured P2P Networks. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 385–396. Springer, Heidelberg (2007)
3. Breker, L.P.: A survey of network pricing schemes. In: Theoretical Computer Science (1996)
4. Carzaniga, A., Rosenblum, D.-S., Wolf, A.: Design and Evaluation of a Wide-Area Event Notification Service. In: ACM TOCS (2001)
5. Chatfield, C.: The Analysis of Time Series - An Introduction. CRC Press (2004)
6. DeLong, J.B.: Six Families Budget Their Money. In: Lecture notes for American Economic History, University of California at Berkeley (2008)
7. Demetriades, I., Lee, T.Y., Moukas, A., Zacharia, G.: Models for pricing the distribution of information and the use of services over the Internet: A focus on the capital data market industry (1998), <http://web.mit.edu/ecom/www/Project98/G12/>
8. Drosou, M., Stefanidis, K., Pitoura, E.: Preference-aware publish/subscribe delivery with diversity. In: DEBS (2009)
9. Dumrogsiri, A., Fan, M., Jain, A., Moinzadeh, K.: A supply chain model with direct and retail channels. European Journal of Operational Research (2008)
10. Faig, M., Jerez, B.: Inflation, Prices, and Information and Competitive Search. Journal of Macroeconomics (2006)

11. Feldman, M., Lai, K., Chuang, J., Stoica, I.: Quantifying Disincentives in Peer-to-Peer Networks (2003),  
<http://www.cs.berkeley.edu/~istoica/papers/2003/discincentives-wepps.pdf>
12. Forker, L., Stannack, P.: Cooperation versus Competition: do buyers and suppliers really see eye-to-eye? *European Journal of Purchasing and Supply Management* (2000)
13. Fuqua, A., Ngan, T.-W.J., Wallach, D.: Economic Behavior of Peer-to-Peer Storage Networks. In: *Economics of Peer-to-Peer Systems* (2003)
14. Gedik, B., Liu, L.: PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In: *ICDCS* (2003)
15. Gupta, A., Sahin, O.D., Agrawal, D., El Abbadi, A.: Meghdoot: Content-Based Publish/Subscribe Over P2P Networks. In: Jacobsen, H.-A. (ed.) *Middleware 2004. LNCS*, vol. 3231, pp. 254–273. Springer, Heidelberg (2004)
16. Hassan, A., Elie, K.: MAWS: A platform-independent framework for Mobile Agents using Web Services. In: *JPDC* (2006)
17. Idreos, S., Koubarakis, M., Tryfonopoulos, C.: P2P-DIET: One-Time and Continuous Queries in Super-Peer Networks. In: Hwang, J., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) *EDBT 2004. LNCS*, vol. 2992, pp. 851–853. Springer, Heidelberg (2004)
18. Johansson, B., Persson, H.: Self-organised adjustments in a market with price-setting firms. In: *Chaos, Solitons and Fractals* (2003)
19. West, Jr., L.A.: Private Markets for Public Goods: Pricing Strategies of Online Database Vendors. In: *Journal of Management of Information Systems* (2000)
20. Kennickell, B.B.A., Moore, K.: Recent Changes in U.S. Family Finances: Evidence from the 2001 and 2004 Survey of Consumer Finances (2006)
21. Khouja, M., Hadzikadic, M., Rajagopalan, H., Tsay, L.: Application of complex adaptive systems to pricing reproducible information goods. *Decision Support Systems* (2007)
22. Koubarakis, M., Koutris, T., Tryfonopoulos, C., Raftopoulou, P.: Information Alert in Distributed Digital Libraries: The Models, Languages, and Architecture of DIAS. In: Agosti, M., Thanos, C. (eds.) *ECDL 2002. LNCS*, vol. 2458, p. 527. Springer, Heidelberg (2002)
23. Koubarakis, M., Tryfonopoulos, C., Idreos, S., Drougas, Y.: Selective Information Dissemination in P2P Networks: Problems and Solutions. In: *SIGMOD Record* (2003)
24. Koubarakis, M., Tryfonopoulos, C., Raftopoulou, P., Koutris, T.: Data Models and Languages for Agent-Based Textual Information Dissemination. In: Klusch, M., Ossowski, S., Shehory, O. (eds.) *CIA 2002. LNCS (LNAI)*, vol. 2446, p. 179. Springer, Heidelberg (2002)
25. Li, C., Singh, M., Sycara, K.: A Dynamic Pricing Mechanism for P2P Referral Systems. In: *AAMAS* (2004)
26. Nottelmann, H., Fuhr, N.: Evaluating Different Methods of Estimating Retrieval Quality for Resource Selection. In: *SIGIR* (2003)
27. Pietzuch, P., Bacon, J.: Hermes: A Distributed Event-Based Middleware Architecture. In: *Proceedings of the International Workshop on Distributed Event-Based Systems, DEBS* (July 2002)
28. Raje, R., Qiao, M., Mukhopadhyay, S., Palakal, M., Peng, S., Mostafa, J.: Homogeneous Agent-Based Distributed Information Filtering. In: *Cluster Computing* (2002)
29. Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A Scalable Content-Addressable Network. In: *SIGCOMM* (2001)
30. Rowstron, A., Kermarrec, A.-M., Castro, M., Druschel, P.: Scribe: The Design of a Large-scale Event Notification Infrastructure. In: Crowcroft, J., Hofmann, M. (eds.) *COST264 Workshop* (2001)
31. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: *SIGCOMM* (2001)

32. Tang, C., Xu, Z.: pFilter: Global Information Filtering and Dissemination Using Structured Overlay Networks. In: FTDCS (2003)
33. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In: Rauber, A., Christodoulakis, S., Tjoa, A.M. (eds.) ECDL 2005. LNCS, vol. 3652, pp. 25–36. Springer, Heidelberg (2005)
34. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks. In: SIGIR (2005)
35. Tryfonopoulos, C., Koubarakis, M., Drougas, Y.: Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In: SIGIR (2004)
36. Tryfonopoulos, C., Zimmer, C., Weikum, G., Koubarakis, M.: Architectural Alternatives for Information Filtering in Structured Overlays. In: Internet Computing (2007)
37. Üstüner, T., Godes, D.: Better Sales Networks. In: Harvard Business Review (2006)
38. Varian, H.R.: Pricing Information Goods. In: Research Libraries Group Symposium, Harvard Law School (1995)
39. Varian, H.R.: Pricing Electronic Journals. D-Lib Magazine (1996)
40. Varian, H.R.: Versioning Information Goods (1997), <http://people.ischool.berkeley.edu/~hal/Papers/version.pdf>
41. Varian, H.R.: Buying, Sharing and Renting Information Goods. Journal of Industrial Economics (2000)
42. Zhang, R., Hu, Y.C.: HYPER: A Hybrid Approach to Efficient Content-Based Publish/Subscribe. In: ICDCS (2005)
43. Zhu, Y., Hu, Y.: Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services. In: IEEE TPDS (2007)
44. Zimmer, C., Tryfonopoulos, C., Berberich, K., Koubarakis, M., Weikum, G.: Approximate Information Filtering in Peer-to-Peer Networks. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 6–19. Springer, Heidelberg (2008)
45. Zimmer, C., Tryfonopoulos, C., Berberich, K., Weikum, G., Koubarakis, M.: Node Behavior Prediction for LargeScale Approximate Information Filtering. In: LSDS-IR (2007)
46. Zimmer, C., Tryfonopoulos, C., Weikum, G.: MinervaDL: An Architecture for Information Retrieval and Filtering in Distributed Digital Libraries. In: Kovács, L., Fuhr, N., Meghini, C. (eds.) ECDL 2007. LNCS, vol. 4675, pp. 148–160. Springer, Heidelberg (2007)

# Trust Alignment: A Sine Qua Non of Open Multi-agent Systems

Andrew Koster<sup>1,2</sup>, Jordi Sabater-Mir<sup>1</sup>, and Marco Schorlemmer<sup>1,2</sup>

<sup>1</sup> IIIA - CSIC

<sup>2</sup> Universitat Autònoma de Barcelona  
Bellaterra, Spain

**Abstract.** In open multi-agent systems trust is necessary to improve cooperation by enabling agents to choose good partners. Most trust models work by taking, in addition to direct experiences, other agents' communicated evaluations into account. However, in an open multi-agent system other agents may use different trust models and as such the evaluations they communicate are based on different principles. This article shows that trust alignment is a crucial tool in this communication. Furthermore we show that trust alignment improves significantly if the description of the evidence, upon which a trust evaluation is based, is taken into account.

## 1 Introduction

A prerequisite for cooperation is that an agent may reasonably expect this cooperation to succeed. The cooperating agents need to know that their interaction partner will perform the action it agreed to. In many systems this can be enforced by the architecture of the system, however in open systems in which the individual agents maintain their autonomy, such as e-Commerce or smart electricity grids, this type of guarantee is not available and agents may be capable of cheating, lying or performing other unwanted behaviour. In such open multi-agent systems the agents need to choose selectively whom to cooperate with and trust is a fundamental tool for performing this selection.

Unfortunately, it is more complicated than equipping an agent with one of the available computational trust models [1] and expecting it to function in a social environment. Using trust as a method for picking successful cooperation partners relies not only on having a good trust model, but also on communication of trust evaluations with other agents [2]. This communication is far from straightforward, because trust is an inherently subjective concept [3]. In this paper we show that to communicate trust evaluations between agents some form of *trust alignment* is needed.

The subjectivity of trust can be seen in the following example, which also demonstrates why this is problematic for communication: consider an e-Commerce environment in which two agents buy the same bicycle via an online auction. One may evaluate the sale as very successful, because the bicycle was



cheap and in good condition. The other, however, puts more emphasis on delivery time and, since the seller delayed significantly before sending, it gives the seller a negative evaluation. Despite having identical interactions, the two agents differ significantly in their trust evaluations of the seller agent. If one of these agents had asked the other agent for advice regarding the seller, that advice would not have been *accurate* within the receiving agent's frame of reference, because the two agents support their trust evaluations with different aspects of the interaction. This problem extends to all domains in which open multi-agent systems may be applied. If trust evaluations – and other subjective opinions – are to be communicated accurately in such domains, a different set of tools is required than is used for the communication of facts. In [4] this is referred to as trust alignment and a couple of different methods for such alignment are suggested.

In this article we discuss the methods used to solve the problem and show, through experimentation, firstly that trust alignment is necessary for effective communication about trust, and secondly that, for truly effective alignment, the evidence on which a trust evaluation is based needs to be taken into account. This experimentation is detailed in Section 3 and the results are discussed in Section 4 before concluding the article in Section 5. The next section further introduces the problem of trust alignment and the proposed solutions to it.

## 2 Methods for Aligning Trust

Trust alignment is a method of dealing with the problem of interpreting another agent's trust evaluations, despite knowing that such evaluations are entirely subjective. As such it is classified as a problem of semiotic, or pragmatic, alignment [5]. While such problems are described in the field of semantic alignment, very little work has been done on finding solutions. Despite this, the field of semantic alignment provides a valuable framework [6] in which to define the problem of trust alignment. We can define trust alignment as the process of finding a translation of the other agent's trust evaluations, based on shared evidence. Its result is a method to translate other trust evaluations from the same agent, based on non-shared evidence. With evidence we mean an objective description of some artifacts in the environment, such as interactions the agents have participated in. Shared evidence is an objective description of an artifact which both agents have perceived, while non-shared evidence refers to artifacts which the receiving agent has not perceived. By using the shared evidence as a common ground, two agents can communicate their differing trust evaluations based on the same evidence and use these different evaluations of the same object as the starting point for finding a translation.

With this definition we can analyze the various processes which could serve to find such a translation. While many trust models have been proposed for computational agents in a multi-agent system [1], very few consider the interpretation of other agents' evaluations as being problematic. Of those that do, the majority are attempts at distinguishing between honest and dishonest agents.

Approaches such as those described by [7,8] attempt to find lying agents and discard all trust evaluations received from them. However, by discarding this information such methods run the risk of missing out on a lot of information; not because the communicating agent is dishonest, but because it has a different underlying trust model. Especially in an open multi-agent system it cannot be assumed that any agent with a differing opinion is being untruthful, although there may very well be such untruthful agents in the system. Detecting these is a separate and important problem, which such reputation filtering methods deal with, however these methods cannot properly be considered solutions to the problem of trust alignment.

By realizing trust alignment is first and foremost a problem of alignment, a number of common ontologies have been proposed to bridge the gap between different trust models [9,10]. However in practice these ontologies do not have the support of many of the different trust methodologies in development. An ontology alignment service is presented in [11], but all these approaches are limited: they align the meaning of the concepts of trust, but not how an agent arrives at, or uses, a specific evaluation, and thus they do not deal with the fact that trust evaluations are subjective. To clarify this distinction we refer back to the example in the introduction: the agents disagree on *how* to evaluate a target, with one agent giving more importance to cost and quality, whereas the other gives more importance to delivery time. If these agents were to communicate their evaluations then, despite having a shared ontology, they would not be meaningful to the other agent. While a single interaction is generally not considered enough to base a trust evaluation on, such differences in how evaluations are computed are propagated all throughout the model, and eventually two syntactically equal evaluations can *mean* something different to different agents. Therefore, despite the work that has been done on applying common ontologies, for instance in the ART testbed [12], the scope in which this is possible seems limited.

## 2.1 Learning a Translation

The first work to address trust alignment directly is, to our knowledge, [13]. This work describes a trust model that evaluates a trustee with an integer between 1 and 4, where 1 stands for *very untrustworthy* and 4 for *very trustworthy*. The alignment process uses the recommendations from another agent about *known* trustees to calculate four separate biases: one for each possible trust value. First the alignment method calculates the own trust evaluations of the corresponding trustee for each incoming recommendation. The *semantic distance* between the own and other's trust is simply the numerical difference between the values of the trust evaluations. The semantic distances are then grouped by the value of the corresponding received trust value, resulting in four separate groups. Finally the bias for each group is calculated by taking the mode of the semantic distances in the group, resulting in four integers between -3 and 3, which can be used when the agent receives recommendations about unknown trustees. Simply subtract the corresponding bias from the incoming trust evaluation to translate the message. While this is a very simple approach it seems to work surprisingly

well. We will return to this method in Section 3.4, however first we will discuss later developments, based on a similar concept but recognizing that trust evaluations may differ between situations and thus the evidence for such trust evaluations must be taken into account in the translation.

## 2.2 Machine Learning Using Context

Current methods to learn a translation take the context into account, by using machine learning techniques to learn which own trust evaluation corresponds to a recommendation, when taking the evidence supporting the evaluations into account [14,15]. The evidence in these methods is a description in a shared, objective language of the interactions a trust evaluation is based on. For instance, the experimentation described in [14] links an evaluation of a sale interaction with a single propositional variable describing that sale (specifically whether the item was delivered on time or not). The alignment method uses this linked information: the evidence together with both its own and the other’s trust evaluation as input for a machine learning algorithm. This algorithm learns a generalization, which serves to translate future communications in which the receiving agent cannot calculate its own trust evaluation, because the interaction being described is not shared. Insofar as we know there are two approaches which have been shown to work using this technique: BLADE [14] uses a conjunction of propositions to describe the interactions and a Bayesian Inference Learner to learn a generalization and we proposed a method [15] that allows a description of the interactions in first-order logic and an Inductive Logic Programming learner to find the generalization. While these two methods use different machine learning techniques, the largest difference between the two approaches is the representation of the contextual information. BLADE uses a propositional representation, which cannot adequately represent domains involving multiple entities and the relationships among them [16], while first-order logic is suited for this task. We discuss these methods in greater detail in Section 3.4.

## 3 Experiments

All the methods in the previous section have been implemented, however thus far no attempt has been made to show what approach is best used. As such it is an open question whether taking the context into account improves the alignment. Moreover, it has not been evaluated to what extent alignment methods improve communication at all. In this section we answer these questions empirically.

### 3.1 Experimental Setup

The aim of the experiments is to measure the effect of communication about trust on the accuracy of agents’ trust evaluations. We are explicitly not interested in evaluating trust models and whether they choose the correct target. For this there are other methods, such as the aforementioned ART testbed. To measure

the effect of communication we need to compare two situations: (1) an agent’s *estimated* trust evaluations, initially given incomplete information about the environment, but allowing communication about trust, and (2) that same agent’s *most accurate* trust evaluations; given perfect and complete information about an environment. This allows for the comparison between the two evaluations and gives a measure for the *accuracy* of the estimated trust evaluation. By varying the amount of communication allowed and the type of alignment used we can measure the influence that alignment has upon the accuracy of the agents’ trust evaluations.

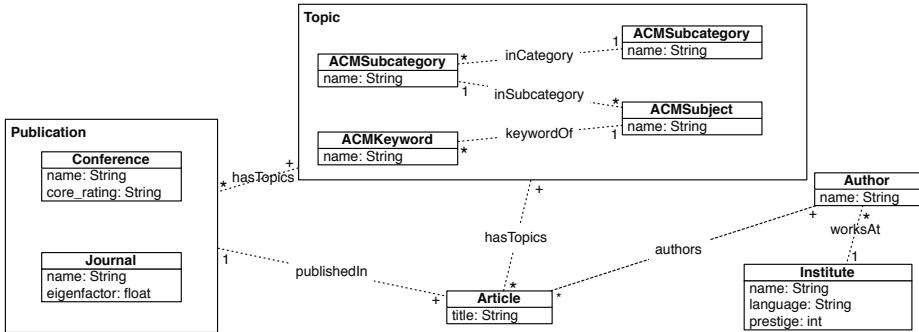


Fig. 1. A UML-like specification of the domain language

Drawing from the LiquidPub project [17], the experiments are focused around a scenario in which agents have to recommend authors to each other, basing these recommendations on the articles they have written. We generate synthetic articles written by between one and five authors each. We specify these articles in a language using a fixed vocabulary, given in Figure 1, that describes properties of the articles. We consider these articles as representations of an interaction between authors, readers and any other stakeholders. We focus on the way readers observe such an interaction through the action of reading the article and forming an opinion about it and the authors. The authors are the trustees to be evaluated, the readers the evaluators and the articles serve as evidence.

In an initialization phase, the articles are divided over the reader agents, such that each reader only receives articles written by a configurable percentage of the author agents. The goal is to give each reader only partial information, so that each of them has incomplete information about only some of the authors in the system, thus creating the need for communication. For this communication two languages are needed. The first is that in which subjective trust evaluations can be communicated. This has a fixed syntax, but the semantics are the subjective evaluations of each agent: the meaning of trust is dependent on each agent’s trust model. Because of the fixed syntax all agents will agree on what type of

<sup>1</sup> Code and documentation can be downloaded at <http://www.megaupload.com/?d=SJL2NLH9> with password: coopis

**Algorithm 1.** Abstract Trust Model

---

```

Input:  $t \in Authors$ , the target author to be evaluated
Input:  $Articles$ , a set of articles, written by  $t$ 
Input:  $Communicated\_Evaluations$ , a set of communicated evaluations from
        other evaluator agents in the system
Input:  $default\_eval$ , a default trust evaluation, used in case no articles have
        been observed and no communicated evaluations have been received
if  $Articles \neq \emptyset$  then
   $article\_ratings := \emptyset$ 
  foreach  $Article\ a \in Articles$  do
     $article\_ratings := article\_ratings \cup evaluate(t, a)$ 
   $trust\_eval := aggregate(article\_ratings)$ 
else if  $Communicated\_Evaluations \neq \emptyset$  then
   $certainty := 0$ 
  foreach  $Evaluation\ e \in Communicated\_Evaluations$  do
    if  $certainty(e) \geq certainty$  then
       $certainty := certainty(e)$ 
       $trust\_eval := value(e)$ 
else
   $trust\_eval := default\_eval$ 
Output:  $trust(t, trust\_eval)$ 

```

---

trust evaluations are allowed, but *why* a trustee is evaluated with any specific value is subjective and this is what needs aligning. To describe the articles we use the same language as the one used to generate them.

After the initialization the experiment runs for  $n$  rounds, in which each round represents the opportunity for the readers to communicate. In each round the agents may pick one other reader agent to communicate with. A communication act may be: a request to either align, or to get the other's trust evaluation of a single author. After  $n$  rounds of communication a measure of each agent's individual accuracy is calculated and, averaging these individual measures, the score of the entire run is determined. This score can then be compared to runs with a different value for  $n$  or using different methods of alignment.

### 3.2 Trust Models

In the experiments, we use five different reader agents, each with its own trust model. All these models use the same general structure, given in Algorithm 1. The models distinguish between direct trust and communicated trust. If the reader has observed any articles written by the author  $T$ , it uses direct trust. This depends on the **evaluate** and **aggregate** functions to calculate a trust evaluation. If no articles have been observed, then communicated trust is used: each communicated evaluation has an uncertainty associated with it, which is dependent on the alignment method used. The agent selects the single communication with the highest certainty to use. If there are also no communicated evaluations available, then a default trust evaluation is used. This is a very basic

trust model and most models in the literature use a more sophisticated method of aggregating information from different sources (e.g. direct trust, reputation, communicated evaluations), however this model is sufficient to show the problems that arise if the agents do not align and to evaluate the different alignment methods. Sophisticated aggregation methods have large advantages at the individual level, because they allow for richer models and more predictive methods, however if two agents use different aggregation methods, it is hard to distinguish whether the difference in trust evaluation is because the agents use a different aggregation method, or because they use different aspects of the interactions.

Work has been done on learning aggregated values [18], however this work is not yet applicable to the more complicated aggregation methods used in modern trust models. The trust alignment methods described in Section 2 avoid this issue by aligning the ratings of individual interactions. The agents can then use their *own* aggregation method, thereby obviating the need to solve the more complex problem of finding an alignment after aggregation. For the **aggregate** we take the average of the article ratings, although as we just explained, an agent could equally well use a probabilistic method such as BRS [19] or a more social network oriented approach, such as Yu & Singh’s model [20].

The **evaluate** function is where each of the reader’s trust models differs. Based on the description of articles in the domain ontology given in Figure 1, each agent has a different way of calculating some values for subjective properties of the article, such as readability or originality. Based on these, the agent calculates the rating of the author, using a list of “if-then-else” rules in Prolog, such as the following:

```
evaluation(Target, Article, 5) :- authors(Article, Authors), member(Target, Agents),
    significance(Article, Sig), Sig > 0.7, originality(Article, Ori),
    Ori > 0.7, readability(Article, Read), Read > 0.7, !.
```

This rule states that if the target agent is an author of the article and the observations of significance, originality and readability are all greater than 0.7 then the evaluation of the author, based on that article has value 5. All five of the readers’ trust models are comprised of such rules, but they only coincide in the structure. The trust models differ in the actual content of the rules, such as the values attributed to different combinations of the subjective properties. Furthermore, the way in which the subjective properties, such as readability, are calculated, is different.

Additionally, one of the readers distinguishes between the first and other authors, using a different set of rules for either case. Another reader distinguishes between articles published in journals and those published in conferences. This leads to five different models, with different complexities for the alignment between them.

### 3.3 Strategy

In addition to the trust model, each agent must have a strategy to choose what to do in each round. While we cannot focus too much on this in the scope of this article, we realize that this choice may have a large influence on the outcome

of the experiment. We therefore implement two strategies for comparison. The first is a simple random strategy. Each agent chooses an author at random. It then chooses a reader agent at random to ask about that author. If it has not previously aligned with that reader, rather than asking for the agent’s evaluation, it asks to align. If it has already aligned, it asks for the other agent’s evaluation of the chosen author.

The second strategy is a non-random strategy in which each agent first chooses the author it has the least certain evaluation of. We use a very simple notion of certainty: an agent’s certainty is equal to the percentage of the author’s articles that the agent has observed. This notion may not be particularly accurate (for instance, if the author has written only very few articles), but it is only a heuristic for selecting which author to obtain more information about. It does not affect the trust evaluation. After choosing the target author, it picks the reader agent that has the most observations of that target and whose opinion has not yet been asked. After choosing the author and evaluator agent, this strategy behaves the same as the random strategy: if the agent has already aligned with the chosen evaluator it asks for a trust evaluation and otherwise it asks to align. While there are many optimizations possible, they are also further distractions from the main tenet of this research. We do not doubt that there are ways of improving the strategy of choosing when to align or with whom to communicate, however the main idea is that if we can show that the trust evaluations are more accurate with alignment than without, performance should only improve if the strategy is optimized.

### 3.4 Alignment Methods

Before discussing the experiments in detail we need to introduce the trust alignment methods we compare.

**Average Bias.** Our first alignment method is a very simple method, which does not take the context into account. When aligning, it calculates the mean difference between the other’s recommendations and the own trust evaluations and use this as a single bias. We will call this method the alignment using an *average distance bias*.

**Abdul-Rahman & Hailes’ Method (AR&H).** AR&H’s model cannot be applied directly, because it requires discrete values to calculate the bias. Because in our models the aggregated trust evaluation is the average of an author’s ratings as the trust evaluation, we do not have discrete values. However, we can apply AR&H’s alignment method at the level of the ratings of individual articles, which are discrete: specifically, in our experiment they are natural numbers between -5 and 5. Furthermore, because we use a real value for the trust evaluation we can refine the method slightly by using the mean, rather than the mode for each bias. Other than that slight refinement, the method applied is the same as that already described in Section [2.1](#).

**Koster et al.’s method.** The third alignment method we test is the one we proposed in [15], using a first-order Inductive Logic Programming (ILP) algorithm. This is one of the two methods designed thus far, based on machine learning algorithms, the other being BLADE [14], which uses a propositional Bayesian Inference Learner. Comparing these two methods is not straightforward, because of the difference in representation. In [21], it is demonstrated empirically that propositional logic decision tree learners (which are propositional ILP algorithms) and Bayesian Inference Learners perform approximately equally, although ILP algorithms perform computationally better in large problems. Unfortunately BLADE is not equipped to deal with the more complex problem we consider here, in which a first-order – rather than a propositional – logic is used to describe articles. To learn relations in this language would require a different, first-order Bayesian network, which falls outside the scope of this work.

The implementation of our method, which we will refer to as Koster et al.’s method, follows the description in [22], which uses the first-order regression algorithm TILDE [23] to learn an alignment. Regression is a form of supervised learning, in which the goal is to predict the value of one or more continuous target variables [24] from a (finite) set of cases. A first-order regression algorithm does this by using, in addition to the numerical cases, an additional description in first-order logic. A case in our situation is a numerical rating of an article, together with a description of that article, communicated using the ontology in Figure 1. The algorithm is implemented in the ACE package [25] and gives as output a set of Prolog clauses which can be used to translate future communications. The technical report describing this version of the alignment method includes some preliminary experimentation. It gives experiments showing under what circumstances the learning algorithm gives good results, but does not place this in a frame of reference in which the algorithm can be compared to other methods, or even with the lack of alignment.

### 3.5 Comparing Alignment Methods

The first experiment aims to compare the alignment methods with each other as well as with the two default modes: no communication at all and communication without alignment. As described in Section 3.2, if an agent has no knowledge of an author, it uses a default trust evaluation. Because the agents have incomplete information about the environment, this case will occur when no, or too little, communication is allowed. The default evaluation can be seen as the agent’s initial evaluation of any author, before learning anything about it and we distinguish between the following options:

- A mistrusting agent.** always gives its most negative evaluation to any agent it has no knowledge of.
- A trusting agent.** always gives its most positive evaluation to any agent it has no knowledge of.
- A neutral agent.** always gives a middle evaluation to any agent it has no knowledge of.



**A reflective agent.** calculates the mean of all its previous trust evaluations of other agents and uses this for any agent it has no knowledge of.

The first three options give a fixed value, independent of the agent’s evaluations of other targets in the system, whereas the last option allows the agent some type of adaptability, depending on what trust evaluations it has so far given to other targets. If the targets it has knowledge of are all bad agents, then it will be more similar to the first option, whereas if they are all good it will be more similar to the second. Of all options for no communication we expect this will be the best choice for an agent, although it is also the only option which requires extra computation.

**Setting up the Experiment.** We start by running a number of experiments to ascertain which parameters should be used for a fair comparison between the alignment models. By changing the total number of articles and the percentage of articles observed by each agent we can change the average number of articles shared by the agents. This mainly influences the functioning of AR&H’s and Koster et al.’s methods. At low numbers of shared articles AR&H’s method outperforms Koster et al.’s, however with around 100 articles shared between any two agents Koster et al.’s method starts to outperform AR&H’s. This difference in performance increases until approximately 500 articles are shared, on average. Running the experiment at higher numbers of shared interactions is unnecessary, because all algorithms have reached peak performance. We opt to run our experiments with 500 shared articles, thus achieving the optimal results obtainable with each of the alignment methods. The goal of the experiment is to measure the influence the different alignment methods have. Therefore we require each agent’s information about the environment to be incomplete. We achieve this by only allowing each reader agent to observe articles by 40% of the author agents. This means that to find out about the other 60% of the authors, communication is required. By having a total of 2000 articles written by different combinations of 50 authors, we can measure the influence of communication while still allowing agents to, on average, share 500 articles. We run each experiment 50 times with different articles to have a decent statistical sample. In this first experiment we vary two parameters: the number of rounds in which agents may communicate and the baseline trust evaluation an agent uses to evaluate targets it has no information of. The results are plotted in Figure 2. The y-axis represents the error with respect to the most accurate evaluation: if the agent were to have perfect information about all articles. Given the probability distribution of a trust model’s evaluations, the error is the probability of the agent’s evaluation of a trustee being between the estimated and most accurate evaluation<sup>2</sup>. It is a measure of the inaccuracy of the alignment method, because the percentage on the y-axis is not the chance that an agent’s evaluation is wrong, but rather a measure of *how* wrong an agent is on average.

---

<sup>2</sup> Calculated as the cumulative probability between the two values.

**Results.** We firstly see in Figure 2(b) that if we use the neutral baseline (using 0 as the default evaluation), then all communication is preferable over no communication. The same is not true if we use the reflective baseline (taking the average of past evaluations of other targets), as seen in Figure 2(a). In this case communication without alignment gives worse results than not communicating at all. This is easily explained: if the observed articles are a representative sample of the population then the mean of trust evaluations based on these will be near the mean of the most accurate trust evaluations. Consequently, always using the default will be quite good. However, the other evaluators' trust evaluations are based on different properties of the articles and may thus be further from the most accurate trust evaluation. The more of these unaligned communicated evaluations an agent incorporates, the less accurate its evaluations will become. We allocate articles at random and therefore each agent does observe a representative sample of them. This same would not be true if the network were not a random network or the location of an agent in the network influenced its trustworthiness: the trustees observed would not be a representative sample of the other agents in the network and the error from using the default would be larger. If this error becomes large enough it would resemble the situation with the neutral baseline, in which case the error from using unaligned communications results in an improvement. We have omitted the experiments using the trusting and distrusting baselines, because their results are very similar to those of the experiment with the neutral baseline and thus add very little information.

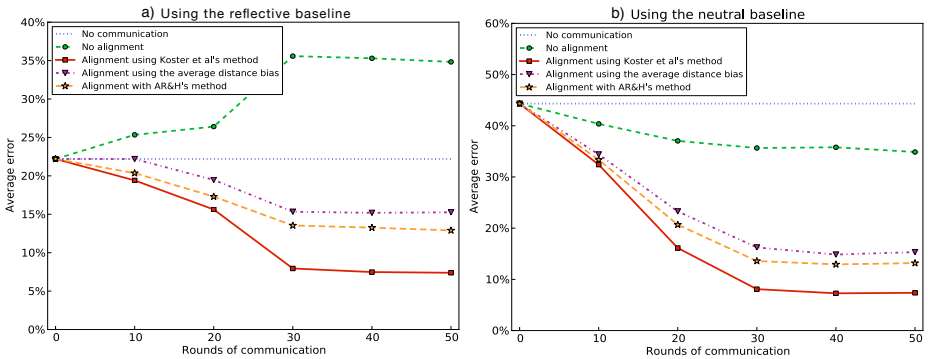


Fig. 2. Average score - with and without alignment

The main result of this experiment is that communication *with* alignment *always* gives significantly better results than either no communication or communication without alignment. In the graphs of Figure 2 we have plotted the average accuracy for all five of the agents, however as discussed in Section 3.2, the individual trust models play a large role in this performance. The different alignment methods give different returns for the individual agents, but always significantly outperform the situations without alignment. Furthermore the differences seen in the graphs are significant. Because the accuracy measure is

not normally distributed we evaluated this by using a Kruskal-Wallis test for analysis of variance [26]. The pair-wise difference is also significant, as tested using Mann-Whitney U-tests<sup>3</sup>. While this seems to indicate that Koster et al.’s method performs slightly better than either of the methods which do not take the context into account, it seems premature to draw this conclusion, given the assumptions underlying the experiment. However, this experiment does serve to show that *some* form of alignment is necessary to communicate about trust. The real advantages of taking the context into account are discussed in Section 3.7, where we deal with untrustworthy communicators.

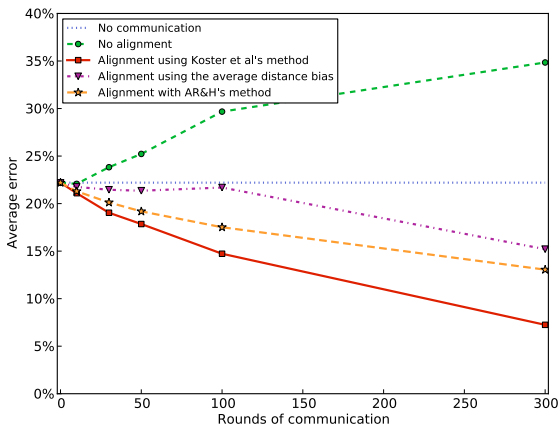


Fig. 3. The random strategy for partner selection

### 3.6 Using a Random Strategy

The first variation on this experiment we explore is to change the strategy for selecting a communication action. The first experiment uses the non-random strategy and we compare these results to the exact same experiment, but using the random strategy. For this experiment we use the reflective baseline and the results up to 300 rounds of communication are plotted in Figure 3. As is to be expected, we see that in the short term picking the communication at random does quite significantly worse than using a heuristic to choose whom to communicate with: after 50 rounds of using the non-random strategy all alignment methods are doing significantly better (see Figure 2(a)) than after 50 rounds of using the random strategy (Figure 3). However in the long run the effect is flattened out and eventually the random strategy achieves the same optimum alignment as the non-random strategy. This implies that, after enough rounds of communication, the optimum is fixed by the alignment method and the strategy does not influence it. To show that the value they converge on really is the lowest average error an agent can achieve using the given alignment method, we run

<sup>3</sup> For all tests we obtain  $p \ll 0.01$ : the probability that the different datasets were obtained from the same population is very small.

the non-random strategy for 150 rounds, which is enough rounds for all possible communications to take place. For all the methods tested we compare this with the outcome after 50 rounds for the non-random strategy and 300 rounds for the random strategy: these values are mutually indistinguishable<sup>4</sup>, showing that even after exhausting all possible communication the alignment is not further improved and truly is an optimum.

The strategy, however, does have a strong influence on how fast this optimum is reached. Using a different strategy will change the speed of convergence, but any good strategy will allow agents to converge on the most accurate evaluations of all agents in the system, just better strategies will converge faster.

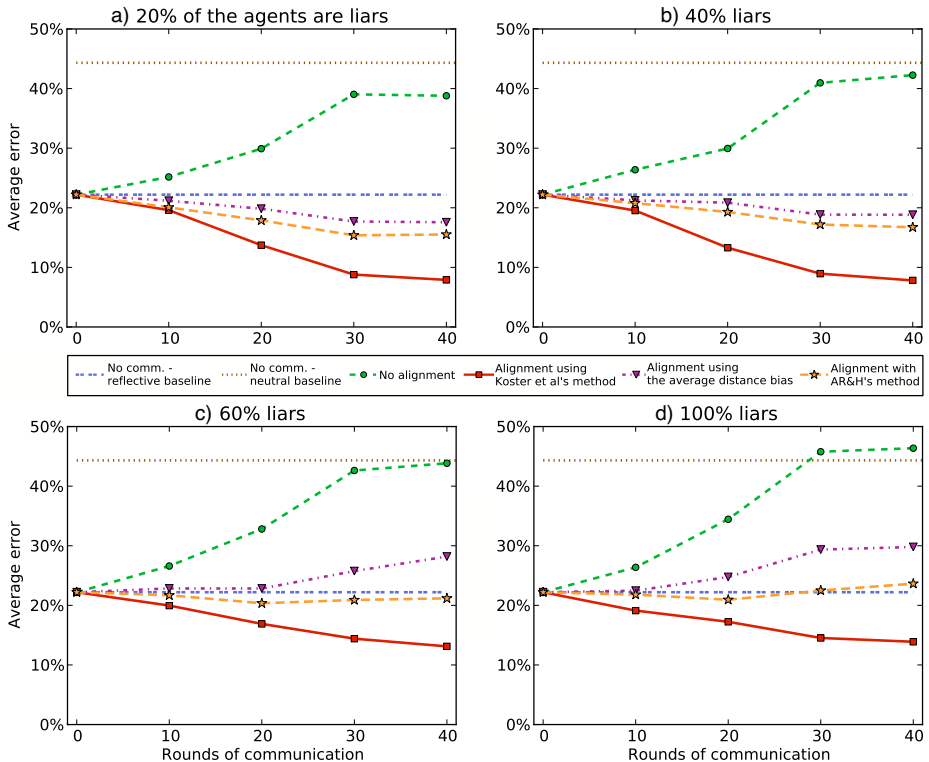
This means that from an agent designer's viewpoint the strategy and alignment method can be completely separated: if an evaluator agent requires information about a target agent, the alignment method defines an optimal accuracy for this information while the strategy defines how many agents on average the evaluator agent must communicate with before it has communicated with the agent giving the most accurate information.

### 3.7 Simulating Lying Agents

In the first experiment we tacitly assumed all agents are truthful and willing to cooperate. If they do not cooperate with the alignment process there is obviously nothing we can do, but assuming other agents are truthful is a rather strong assumption. This experiment is therefore set up to see what happens with the communication if we simulate the other agents passing incorrect information. Note that if the agents are entirely consistent in their lies, AR&H and Koster et al.'s alignment methods will be able to deal with this perfectly, as they learn a translation from the other's trust evaluation. Additionally, Koster et al.'s method is even able to deal with lying if it is not always consistent, but based on some specifics of the underlying article (such as: always lie if the author works at a certain institute). The problem for all alignment algorithms appears if agents just invent a random value. We run another round of experiments, this time increasingly replacing truthful agents by lying ones. A lying agent, rather than giving an actual trust evaluation, communicates random ratings of articles. The results can be seen in Figure 4. The agents using communication use the reflective baseline as their default evaluation in the case they do not have other information available.

**Results.** We focus first on graph (d) in Figure 4 and see that if all agents are lying then communication with no alignment converges to the accuracy of the trust evaluations without communications and using the average of all possible trust evaluations as the fixed evaluation for unknown agents. We can explain this convergence by seeing that the mean of all possible trust evaluations is also the mean value of a random distribution over the possible trust values. A similar thing happens using AR&H's method, which calculates what its own trust evaluation should be if the other agent communicates a certain value. However,

<sup>4</sup> Obtaining  $p \gg 0.05$  for all Mann-Whitney U-Tests.



**Fig. 4.** Slow degradation from a domain with no lying agents to a domain with all lying agents

because the other’s trust evaluations are random, choosing all those at a certain value will give a random sample of the own trust evaluations, the mean of which will, on average, be the mean of all the own trust evaluations, so AR&H’s model stays approximately flat on the default baseline (using the average of all the agent’s own trust evaluations). For similar reasons the average bias does slightly worse, converging to a value between the two baselines. Koster et al.’s method, on the other hand, appears to hardly be affected by the noisy trust evaluations. This shows a large advantage of taking the context into account: Koster et al.’s method maintains its performance, because the communications in the domain language can be used for the alignment method to compensate for the noisy trust evaluations. It ignores the noisy trust evaluations and learns by using *only* the information about the underlying articles. If we were to add noise to this part of the communication as well, Koster et al.’s model would collapse to AR&H’s and thus stay flat as well.

With this explanation of what happens when all agents lie we can see that by slowly adding more liars to the system, the performance of the various algorithms morphs from the system with no liars (Figure 2(a)) to the system with all liars (Figure 4(a)-(d) progressively). To prevent this from happening a further

refinement would be necessary: detecting which agents are the liars and disregarding their communications, as discussed in Section 2.

## 4 Discussion

The experimentation in the previous section demonstrates that trust alignment improves the accuracy of agents' trust evaluations. Koster et al.'s method even works in situations where the communicated evaluations are 100% noise. However, we must take care when interpreting these experiments. The first thing to note is that the trust models used, as described in Section 3.2, are simplifications of those used in the literature. Agents only communicate the evaluations based on their own direct experiences, rather than having an evaluation which is aggregated from a number of different sources. This, however, only strengthens the point we are trying to make: the more complex an agent's trust evaluation can be, the greater the probability that two agents, despite using the same ontology for their trust evaluations, *mean* different things, because the actual way they calculate the evaluations are completely different. The use of more complex trust models thus leads to an even greater need for alignment. Unfortunately, the more complex the trust models, the more information will be required to apply a method such as Koster et al.'s, which requires numerous samples of different types of evidence supporting the trust evaluations. Luckily, the worst case for Koster et al is that the domain information is too complex to use, in which case it will perform similarly to AR&H's method. In such cases there may be other machine learning techniques, such as case based reasoning [27], which is designed to handle large sets of complex data, which could offer a solution.

Additionally the alignment is required to take place before aggregation. This means that regardless of how complex the aggregation method is, as long as what is being aggregated is not too complex, the alignment can work. However, it also means that a large amount of information needs to be communicated. There may be scenarios in which this communication is prohibitive and a simpler form of alignment, such as AR&H's method, or even the average bias, must be used. However, in domains such as e-Commerce, a lot of data is readily made available: on eBay<sup>5</sup> for example, for any transaction it is public knowledge what item was sold and how much it cost. Similarly in social recommender systems, which is how we would classify the example scenario in this paper, people are often willing to explain their evaluation of an experience in great detail (such as on Tripadvisor<sup>6</sup>). This is exactly the type of information that is needed for aligning. If necessary this could be combined with a method of incentivizing truthful feedback, such as described in [28]. This could also be helped to mitigate lies, which is the final point for discussion.

Our model only generates noise in the trust evaluation, not in the description of the evidence. Furthermore, if a hostile agent has knowledge of the aligning agent's trust model, it could tailor its alignment messages so that it can send

<sup>5</sup> [www.ebay.com](http://www.ebay.com)

<sup>6</sup> [www.tripadvisor.com](http://www.tripadvisor.com)

false evaluations undetected. Luckily a lot of work has been done in detecting fraudulent, or inconsistent information, both in the context of trust and reputation [7,8], as well as in collaborative filtering [29]. As briefly mentioned in Section 3.7 such a method could be used in combination with alignment methods. By merging an alignment method with a filtering method the efficacy of both can be significantly improved. Good alignment rules can be used to minimize the useful information discarded, while the filtering methods are well equipped to decide when an agent is not giving any useful information at all.

## 5 Conclusions and Future Work

The experimentation shows clearly that communication without alignment may have a negative influence on the accuracy of an agent’s trust evaluations and thus that alignment is a necessary step when talking about trust. We see that even a simple alignment method such as calculating an average bias, can give a significant boost to the trust model’s accuracy. AR&H and Koster et al.’s methods function at least as well as not communicating even if all other agents are liars. Koster et al.’s method outperforms all other methods tested, by taking *the context* in which a trust evaluation was made into account. This performance, however, comes at a cost: Koster et al.’s model uses a relatively complex learning algorithm and requires communication about not only ratings of individual interactions, but also an objective description of the interaction it is based on. The functioning of this alignment method may very well depend on the expressiveness of the language for describing interactions. If such a language is very basic, then alignment may not be possible and a simpler method must be used. Similarly, privacy issues may arise in scenarios where agents are willing to exchange trust evaluations, but not anything more. In such cases the best we can do is the method taking an average bias. Whether the increased complexity and communication load is worth the added performance should be evaluated per domain. Additionally, the trust models themselves influence the accuracy of alignments. Analyzing the interplay of some different trust models used in practice, as well as more – or less – descriptive domain languages for describing the context, is an important concern for future research. Another promising method for alignment lies in argumentation about trust [30]. Such methods attempt to establish and explain the causal link between what happened in the environment and the trust evaluation it resulted in by giving a formal argumentation framework in which agents can communicate their reasons for trust. Thus far such methods are not yet developed sufficiently to be applied for alignment.

**Acknowledgements.** This work is supported by the Generalitat de Catalunya grant 2009-SGR-1434 and the Spanish Ministry of Education’s Agreement Technologies project (CONSOLIDER CSD2007-0022, INGENIO 2010) and the CBIT project (TIN2010-16306). Additionally the authors would like to thank the anonymous reviewers for their feedback.

## References

1. 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)
2. Conte, R., Paolucci, M.: *Reputation in Artificial Societies: Social beliefs for social order*. Kluwer Academic Publishers (2002)
3. Castelfranchi, C., Falcone, R.: *Trust Theory: A Socio-cognitive and Computational Model*. Wiley (2010)
4. Koster, A.: Why does trust need aligning? In: Proc. of 13th Workshop “Trust in Agent Societies”, Toronto, pp. 125–136. IFAAMAS (2010)
5. Euzenat, J., Shvaiko, P.: *Ontology matching*. Springer, Heidelberg (2007)
6. Schorlemmer, M., Kalfoglou, Y., Atencia, M.: A formal foundation for ontology-alignment interaction models. *International Journal on Semantic Web and Information Systems* 3(2), 50–68 (2007)
7. Teacy, W.T.L., Patel, J., Jennings, N.R., Luck, M.: Travos: Trust and reputation in the context of inaccurate information sources. *Journal of Autonomous Agents and Multi-Agent Systems* 12(2), 183–198 (2006)
8. Şensoy, M., Zhang, J., Yolum, P., Cohen, R.: Context-aware service selection under deception. *Computational Intelligence* 25(4), 335–366 (2009)
9. Pinyol, I., Sabater-Mir, J.: Arguing About Reputation: The IRep Language. In: Artikis, A., O’Hare, G.M.P., Stathis, K., Vouros, G.A. (eds.) *ESAW 2007*. LNCS (LNAI), vol. 4995, pp. 284–299. Springer, Heidelberg (2008)
10. Casare, S., Sichman, J.: Towards a functional ontology of reputation. In: *AAMAS 2005: Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, Utrecht, The Netherlands, pp. 505–511. ACM (2005)
11. Nardin, L.G., Brandão, A.A.F., Muller, G., Sichman, J.S.: Effects of expressiveness and heterogeneity of reputation models in the art-testbed: Some preliminar experiments using the soari architecture. In: Proc. of the Twelfth Workshop Trust in Agent Societies at AAMAS 2009, Budapest, Hungary (2009)
12. Brandão, A.A.F., Vercouter, L., Casare, S., Sichman, J.: Exchanging reputation values among heterogeneous agent reputation models: An experience on art testbed. In: Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, pp. 1047–1049. IFAAMAS (2007)
13. Abdul-Rahman, A., Hailes, S.: Supporting trust in virtual communities. In: Proceedings of the 33rd Hawaii International Conference on System Sciences, vol. 6, pp. 4–7 (2000)
14. Regan, K., Poupart, P., Cohen, R.: Bayesian reputation modeling in e-marketplaces sensitive to subjectivity, deception and change. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Boston, MA, USA, pp. 1206–1212. AAAI Press (2006)
15. Koster, A., Sabater-Mir, J., Schorlemmer, M.: Engineering trust alignment: a first approach. In: Proc. of the Thirteenth Workshop “Trust in Agent Societies” at AAMAS 2010, Toronto, Canada, pp. 111–122. IFAAMAS (2010)
16. De Raedt, L.: *Logical and Relational Learning*. Springer, Heidelberg (2008)
17. Liquid publications: Scientific publications meet the web. September 2, (2010), <http://liquidpub.org>
18. Uwents, W., Blockeel, H.: A Comparison Between Neural Network Methods for Learning Aggregate Functions. In: Boulicaut, J.-F., Berthold, M.R., Horváth, T. (eds.) *DS 2008*. LNCS (LNAI), vol. 5255, pp. 88–99. Springer, Heidelberg (2008)



19. Jøsang, A., Ismail, R.: The beta reputation system. In: Proceedings of the Fifteenth Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy, Bled, Slovenia (2002)
20. Yu, B., Singh, M.P.: An evidential model of distributed reputation management. In: AAMAS 2002: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 294–301. ACM, New York (2002)
21. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. *Machine Learning* 29(2–3), 131–163 (1997)
22. Koster, A., Sabater-Mir, J., Schorlemmer, M.: Engineering trust alignment: Theory and practice. Technical Report TR-2010-02, CSIC-III A (2010)
23. Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: Shavlik, J. (ed.) Proceedings of the 15th International Conference on Machine Learning, pp. 55–63 (1998)
24. Bishop, C.M.: *Pattern Recognition and Machine Learning*. Springer, Heidelberg (2006)
25. Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H.: Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research* 16, 135–166 (2002)
26. Corder, G.W., Foreman, D.I.: *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley (2009)
27. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1), 39–59 (1994)
28. Witkowski, J.: Truthful feedback for sanctioning reputation mechanisms. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010), Corvallis, Oregon, pp. 658–665. AUAI Press (2010)
29. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. *Advances in Artificial Intelligence 2009*. Article no. 421425 (January 2009)
30. Pinyol, I., Sabater-Mir, J.: An argumentation-based protocol for social evaluations exchange. In: Proceedings of The 19th European Conference on Artificial Intelligence (ECAI 2010), Lisbon, Portugal (2010)

# An Architecture for Defeasible-Reasoning-Based Cooperative Distributed Planning

Sergio Pajares Ferrando, Eva Onaindia, and Alejandro Torreño

Universitat Politècnica de València,  
Camino de Vera, s/n  
46022 Valencia, Spain  
{spajares,onaindia,atorreno}@dsic.upv.es

**Abstract.** Cooperation plays a fundamental role in distributed planning, in which a team of distributed intelligent agents with diverse preferences, abilities and beliefs must cooperate during the planning process to achieve a set of common goals. This paper presents a **MultiAgent Planning and Argumentation (MAPA)** architecture based on a multi-agent partial order planning paradigm using argumentation for communicating agents. Agents use an argumentation-based defeasible reasoning to support their own beliefs and refute the beliefs of the others according to their knowledge. In MAPA, actions and arguments may be proposed by different agents to enforce some goal, if their conditions are known to apply and arguments are not defeated by other arguments applying. In order to plan for these goals, agents start a stepwise dialogue consisting of exchanges of plan proposals to satisfy this open goal, and they evaluate each plan proposal according to the arguments put forward for or against it. After this, an agreement must be reached in order to select the next plan to be refined.

**Keywords:** Cooperative distributed planning, Defeasible Reasoning, Argumentation.

## 1 Introduction

A Cooperative Information System (CIS) is a large scale information system that interconnects various systems of different and autonomous organizations, geographically distributed and sharing common objectives [18]. With the emergence of new technologies in computing, such as SaaS, cloud computing, Service Oriented Computing, mash-ups, Web Services, Semantic Web, Knowledge Grid, and other approaches, it is becoming increasingly natural to deal with Agent-based computing or **MultiAgent Systems**. [28]. Agents, as distributed autonomous software entities, are required to engage in interactions, argue with one another, make agreements, and make proactive run-time decisions, individually and collectively, while responding to changing circumstances. For this reason, agents are being advocated as a next-generation model for engineering complex distributed systems.

Planning is the art of building control algorithms that synthesize a course of action to achieve a desired set of goals of the information system. Unlike classical

planning, in many real-world applications agents often have distributed contradictory information about the environment and their deductions are not always certain information, but *plausible*, since the conclusions can be withdrawn when new pieces of knowledge are posted by other agents. For this purpose, argumentation, which has recently become a very active research field in computer science [4,23], can be viewed as a powerful tool for reasoning about inconsistent information through a rational interaction of arguments for and against some conclusion.

**Defeasible Logic Programming (DeLP)** [9] is a framework for reasoning about defeasible information (also known as defeasible reasoning), where tentative conclusions are obtained from uncertain or incomplete information, and conclusions might no longer be valid after new information becomes available. The work in [10] (see section 3) introduces a first approach known as DeLP-POP framework, to integrate DeLP in Partial Order Planning (POP) [21], and the work in [20] (see section 3) extends DeLP-POP framework to a multiagent environment. As an example on how defeasible reasoning is introduced in these frameworks, we can view an agent as a business person who needs to travel between London and Athens, and has to build a plan to get to Athens. One may think the first action to do is to buy a flight ticket through an airline web site. However, another agent who is aware of the latest news on the Internet, might think the business man will not be able to fly due to a strike announcement in London. Under these circumstances, the second agent will put forward an argument against the first one in order to ensure that the business man accomplishes his goal to get Athens.

The motivation for introducing distributed planning in a multi-agent environment is twofold. On one hand, a multi-agent system design can be beneficial in many domains, particularly when a system is composed of multiple entities that are distributed functionally or spatially. On the other hand, distributed execution promotes the efficiency of parallel processing of actions, the robustness of the system to cope with complex planning problems and the simplicity of an incremental construction across a network of interconnected agents, thus avoiding the critical failures and resource limitations of centralized systems. In this paper, we present a **MultiAgent Planning and Argumentation (MAPA)** architecture for cooperative distributed planning in a multiagent DeLP-POP framework, which extends and refines the preliminary work presented in [20]. This paper is organized as follows: section 2 gives a short related work; section 3 describes a background; section 4 introduces the MAPA architecture; section 5 presents the planning protocol of the architecture; and section 6 shows an example of application to validate the MAPA architecture. Finally, we conclude and present some directions for future work.

## 2 Related Work

This subsection is devoted to study the most relevant related works found in the literature: multi-agent argumentation, cooperative distributed planning (without defeasible reasoning) and centralized planning. Some systems that build on

argumentation apply theoretical reasoning for the generation and evaluation of arguments to build applications that deal with incomplete and contradictory information in dynamic domains. Some proposals in this line focus on planning tasks, or also called practical reasoning, i.e. reasoning about what actions are the best to be executed by an agent in a given situation. Dung’s abstract system for argumentation [7] has been used for reasoning about conflicting plans and generating consistent sets of goals [2]. Further extensions of these works present an explicit separation of the belief arguments and goal arguments and include methods for comparing arguments based on the value of goals and the cost of resources [23]. The combination of defeasible reasoning and planning has been used in [22], in which the whole plan is viewed as an argument and then, defeasible reasoning about complete plans is performed. Although the work in [22] combines defeasible reasoning and partial order planning, defeasible reasoning is not used in the same way as [10]. In contrast, [10] uses arguments for warranting subgoals, and hence, defeasible reasoning is used in each step of the planning search process. In any case, none of these works apply to a multi-agent environment.

A proposal for dialogue-based centralized planning is that of [26], but no argumentation is made use of. The work in [3] presents a dialogue based on argumentation to reach agreements on plan proposals. Unlike our proposal, which focuses on an argumentative and stepwise construction of a plan, this latter work is aimed at handling the interdependencies between agents’ plans. The work in [24] introduces a framework to build joint plans supported through the use of *argumentation schemes* as a mechanism of dialogue during the planning search. On the other hand, we can also find some systems that perform argumentation in multi-agent systems by using defeasible reasoning but are not particularly concerned with the task of planning [27].

### 3 Background

The key element of DeLP are defeasible rules (Head  $\neg$  Body), which are used to represent a deductive relation between pieces of knowledge that could be defeated once other piece of knowledge is considered. Specifically, arguments (combinations of defeasible rules and facts) for conflicting pieces of information are built, and then compared to decide which one prevails. For instance, a defeasible rule like "According to Internet news, an airport strike is expected", is denoted as "*strike*  $\neg$  *news*". Note that, if it occurs in London, then it will disrupt the passengers’ plans for flying between London and Athens.

The principle of least commitment in Partial Order Planning makes it one of the more open planning frameworks. This is evidenced by the fact that most existing architectures for integrating planning with execution, information gathering, and scheduling are based on partial order planners. In [25], authors argue that POP-based frameworks offer a more promising approach for handling domains with durative actions, and temporal and resource constraints as compared to other planning approaches. In fact, most of the known implementations

of planning systems capable of handling temporal and durative constraints (including IxTET [12], as well as NASA's RAX [16]) are based on the POP paradigm. Even for simple planning tasks, partial order planners offer a higher degree of execution flexibility. In contrast, none of the known state-space planners can find parallel plans efficiently [14], and planners such as Graphplan [6] only generate a very restricted types of parallel plans. For this reason, partial order planning remains attractive when compared to state-space planning.

An extension of POP with DeLP-style argumentation, denoted DeLP-POP framework, was introduced in [10], where both actions and arguments may be used to enforce some goal, if their conditions (are known to) apply and arguments are not defeated by other arguments applying. Unlike actions, arguments will not only be introduced to intentionally support some step of a plan, but they will also be presented to defeat or defend other supporting arguments in the plan. When actions and arguments are combined in a partial order plan, new types of interferences or threats appear [10]. These interferences need to be identified and resolved to obtain valid plans.

Finally, the work in [20] proposes a preliminary extension of the theoretical DeLP-POP framework to a multiagent environment. Specifically, it proposes a dialogue for argumentative plan search, by which agents exchange plan proposals and arguments for or against such proposals. Unlike [20], the MAPA architecture presented here solves the qualification problem, identifies new types of threats, and extends the agents' knowledge bases by including a set of agent-specific preferences. This allows us to extend and adapt the planning protocol of MAPA to a fully-automated argumentative dialogue between agents so as to reach agreements during the plan construction. Moreover, the MAPA architecture promotes a more practical vision of the extension of DeLP-POP to a multi-agent environment.

## 4 Elements of the MAPA Architecture

In state-based planning, a plan  $\Pi$  is a linear sequence of actions, and thus before each action is added to the plan  $\Pi$ , we know which consistent state will hold. In contrast, MAPA architecture is based on POP<sup>1</sup>, where a partial order plan  $\Pi$  is a set of actions whose execution ordering  $\prec$  is only partially specified (thus encoding multiple linear plans).

The MAPA architecture works on a planning process distributed among several planning agents, which have an incomplete knowledge (i.e. the set of actions and arguments that an agent can propose can be different from other agents'), and have to devise a joint, non-linear plan which may be later executed by them. The following subsections expose (i) the agents' planning model and the notion of argument, (ii) the improvements introduced to deal with the qualification problem and the notion of plan, and (iii) the new definition and handling of threats introduced by the qualification problem.

<sup>1</sup> We consider that POP is the best planning approach concerned with the dynamic multiagent nature due to the ease to join several plan proposals into a single joint plan.

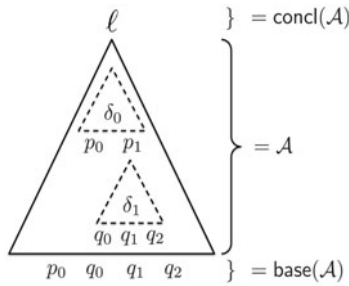
### 4.1 The Agents' Planning Model and Arguments

The planning model of each agent is based on a set of literals  $\text{Lit}$ , such that  $\ell \in \text{Lit}$  is a ground atom and  $\sim\ell \in \text{Lit}$  is a negated ground atom, where  $\sim$  represents the strong negation and  $\bar{\ell} = \sim\ell$ . Each agent  $x$  of the MAPA architecture is initially endowed with a **planning task**  $\mathbb{M}_x = ((\Psi_x, \Delta_x), A_x, F_x, G)$  where:

1.  $\Psi_x \subseteq \text{Lit}$ , represents a consistent set of true facts which describe the initial state of the task.
2.  $\Delta_x$  is a set of defeasible rules  $\delta = \ell_0, \dots, \ell_k \multimap \ell'_0, \dots, \ell'_k$ .
3.  $A_x$  is a set of actions  $\alpha = \langle P(\alpha), X(\alpha) \rangle$  where  $P(\alpha) \subseteq \text{Lit}$  is a set of preconditions and  $X(\alpha) \subseteq \text{Lit}$  is a set of effects.
4.  $F_x$  represents a consistent set of the agent-specific preferences  $F_x \subseteq \{(a, d) \mid (a \in A), d \in [0, 100]\}$ , where the action  $a$  is preferred with the estimated interest degree  $d$ .
5.  $G \subseteq \text{Lit}$  is the set of common goals which have to be satisfied.

The diversity of preferences is addressed by means of agreements between the agents during the planning process. We assume that agents are fully cooperative, so they have no incentives to retain relevant information. In POP,  $\Psi$  (consistent set of agents' initial states of the task) and  $G$  are encoded as dummy actions  $\{\alpha_\Psi \prec \alpha_G\}$  with  $X(\alpha_\Psi) = \Psi$ ,  $P(\alpha_G) = G$ , and  $P(\alpha_\Psi) = X(\alpha_G) = \emptyset$ .

An **argument**  $\mathcal{A}$  for  $\ell \in \text{Lit}$ , is denoted as  $\mathcal{A} = (\{\ell\}, \{\Delta'\})$ , where  $\Delta'$  is a subset of defeasible rules  $\Delta' \subseteq \Delta$ .  $\mathcal{A}$  is consistent if  $\text{base}(\mathcal{A}) \cup \mathcal{A}$  is non-contradictory.



**Fig. 1.** An argument  $\mathcal{A}$  for  $l$  using the two defeasible rules:  $\delta_0 = l \multimap \{p_0, p_1\}$  and  $\delta_1 = p_1 \multimap \{q_0, q_1, q_2\}$

Figure 1 shows an example of an argument proposed  $\mathcal{A}$ , where  $\text{literals}(\mathcal{A}) = \{l, p_0, p_1, q_0, q_1, q_2\}$ . This argument for a literal  $\ell$  does not suffice to warrant  $\ell$ , it depends on the interaction among arguments (see section 5.2), which will grant consistency. Given two arguments  $\mathcal{A}, \mathcal{B}$ , we say  $\mathcal{A}$  *attacks*  $\mathcal{B}$  if the conclusion of  $\mathcal{A}$  contradicts some fact used in  $\mathcal{B}$ , that is, if  $\text{concl}(\mathcal{A}) \in \text{literals}(\mathcal{B})$ . Therefore, the MAPA architecture semantically differentiates between supporting arguments (or **argument steps**) as the arguments specifically used to support some open

condition of the plan, and **attacking arguments** which are only introduced to attack some argument step previously introduced in the plan (i.e. it is not used to support any open condition).

### 4.2 The Qualification Problem and Plan Definition

The qualification problem [13], which is an important problem currently not supported in many planning architectures, is concerned with the impossibility of listing all the preconditions required for a real-world action to have its intended effect. For instance, let  $\alpha$  (e.g. "flying from London to Athens") be an action with  $n$  effects  $\{e_0, e_1, \dots\} \subseteq \text{Lit}$  (e.g.  $e_0 = \text{"be at Athens city"}$ ), which are defeated by the defeasible conditions  $\{d_0, d_1, \dots\} \subseteq \text{Lit}$  (e.g.  $d_0 = \text{"Volcanic ash cloud between London to Athens"}$ ,  $d_1 = \text{"Airport strike in London"}$ ) respectively. Note that, if these defeasible conditions occur, the expected effects of  $\alpha$  would not be achieved. The work in [10] solves this issue by introducing these defeasible conditions as negated preconditions of  $\alpha$  ( $\{\bar{d}_0, \bar{d}_1, \dots\} \subseteq P(\alpha)$ ), which must be derived by arguments.

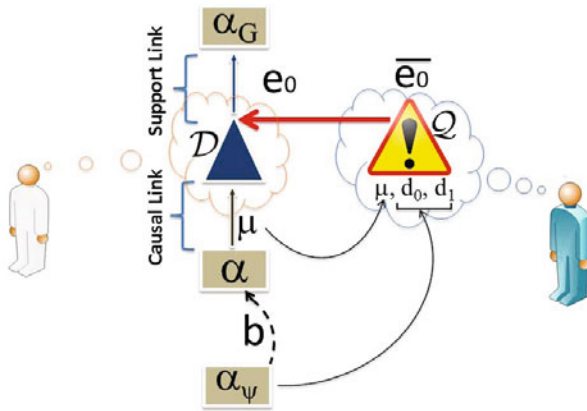


Fig. 2. An example solving the qualification problem

However, an action  $\alpha$  in MAPA architecture follows a specific representation in order to deal with this problem. We introduce a **fictitious effect**  $\mu$  (meaning  $\alpha$  was just executed); then we define  $X(\alpha) = \{\mu\}$  and expand the set of rules  $\Delta$  with  $\{e_k \leftarrow \mu\} \cup \{\bar{e}_k \leftarrow \mu, d_k\}$ , where  $e_k$  represents the effect of the action  $\alpha$  and  $d_k$  is a defeasible condition. For instance, in Figure 2, the precondition  $e_0$  of the action  $\alpha_G$  is initially derived by an argument  $D = (\{e_0\}, \{e_0 \leftarrow \mu\})$  whose  $\text{base}(D) = \mu$  will be satisfied by  $\alpha$ . Then an attacking argument  $Q = (\{\bar{e}_0\}, \{\bar{e}_0 \leftarrow \mu, d_0, d_1\})$ , which is a defeater of  $D$  ( $Q$  attacks  $D$ ), arises from the distributed knowledge among agents. Triangles in Figure 2 represent argument steps (i.e. arguments that support preconditions of action steps), for instance the argument  $D$ , or arguments attacking some other argument, for instance the argument  $Q$ , and both are labeled with the argument name, while rectangles represent action

steps (i.e. actions that support the basis of an argument step) and are labeled with the action name.

The MAPA architecture defines a **plan**  $\Pi$  as a tuple  $\Pi = (A(\Pi), \text{Args}(\Pi), G(\Pi), \mathcal{OC}(\Pi), \mathcal{CL}(\Pi), \mathcal{SL}(\Pi))$ , where  $A(\Pi)$  denotes the set of action steps,  $\text{Args}(\Pi)$  represents the set of argument steps,  $G(\Pi)$  is the the task's common goals,  $\mathcal{OC}(\Pi)$  is a set of ordering constraints, and  $\mathcal{CL}(\Pi)$  and  $\mathcal{SL}(\Pi)$  represent the sets of causal and support links correspondingly. Let  $\ell_1$  be an open goal, motivated by some action step  $\beta \in A$ , i.e.  $\ell_1 \in \text{P}(\beta)$ , and, let  $\ell_2$  be another open goal, motivated by some argument step  $\mathcal{A} \subseteq \Delta$ , i.e.  $\ell_2 \in \text{base}(\mathcal{A})$ . Then, the goal  $\ell_1 \in \text{P}(\beta)$  must be supported by the argument  $\mathcal{A}$ , which will introduce a **support link**  $(\mathcal{A}, \ell_1, \beta) \in \mathcal{SL}(\Pi)$ , where  $\mathcal{SL}(\Pi) \subseteq \Delta \times G(\Pi) \times A$ , while the goal  $\ell_2$  must be satisfied by an action  $\alpha$ , by introducing a **causal link**  $(\alpha, \ell_2, \mathcal{A}) \in \mathcal{CL}(\Pi)$  where  $\mathcal{CL}(\Pi) \subseteq A \times G(\Pi) \times \Delta$ . Note that an argument  $\mathcal{B}$  cannot support another argument  $\mathcal{A}$  with a support link in  $\mathcal{SL}(\Pi)$ , and an action  $\alpha_1$  cannot support another action  $\alpha_2$  with a causal link in  $\mathcal{CL}(\Pi)$ . To get  $\mathcal{B}$  to support step  $\mathcal{A}$ ,  $\mathcal{A}$  must be replaced by  $\mathcal{A} \cup \mathcal{B}$ , and to get  $\alpha_1$  to support action step  $\alpha_2$ , an argument  $(\{e_k\}, \{e_k \rightarrow \mu\})$  must be inserted between  $\alpha_1$  and  $\alpha_2$ , where  $\text{X}(\alpha_2) = \mu$  and  $\text{P}(\alpha_1) = e_k$ . Additionally, unlike in DeLP-POP, ordering constraints are placed between argument steps  $(\mathcal{A}, \mathcal{B}) \in \mathcal{OC}(\Pi)$ , since every action (excepting  $\alpha_G$ ) is preceded by an argument which derives its actual effects.

### 4.3 Interferences among Actions and Arguments

If only actions are taken into account in a planning architecture, then there is only one type of destructive interference that can arise in a plan under construction. This interference is captured by the notion of threat in POP, and occurs when a new action inserted in the plan threatens (deletes) a goal solved by other action steps. When actions and arguments are combined to construct plans, new types of interferences appear that need to be identified and resolved to obtain a valid plan. In multiagent DeLP-POP [20], we identified three types of interferences or threats, that cover all the interferences that may arise in a partial plan: argument-argument, action-argument and action-action threats.

However, since the goals must be initially derived by some argument step in the MAPA architecture, and then its basis must be satisfied by another action step (including the initial step), argument-argument threats cover all the interferences that may arise in a plan dealing with the qualification problem. Nevertheless, MAPA architecture differentiates semantically between:

1. **Planning threats (PlatHreats)**: Threats that arise between two argument steps. For instance, let "w" be an open condition of the plan in Figure 3(c'), then the argument with an admiration is acting as a supporting argument and a PlatHreat will be discovered. These threats override the typical action-action and action-argument threats of [20]. As we will discuss in subsection 5.1, this kind of threats will be discovered and possibly resolved (by promote or demote) in the *POP Search Tree*.



2. **Argumentation threats (ArgThreats)**: Threats that arise when an agent discovers a new defeater which specifically attacks some argument step. Unlike the PlaThreats, here the attacks to some argument step are made by some attacking argument. For instance, in case that the argument with an admiration in Figure 3(c') is an attacking argument (i.e. "w" is not an open goal), Figure 3(c') represents an ArgThreat. Although this kind of threat is also called argument-argument threat in [20], here we rename them to ArgThreats with the aim of distinguishing between PlaThreats and ArgThreats. As shown in subsection 5.2, these threats will be discovered and possibly resolved (by Defeat-the-defeater in Figure 3(c'')) in the *POP Evaluation Tree*.

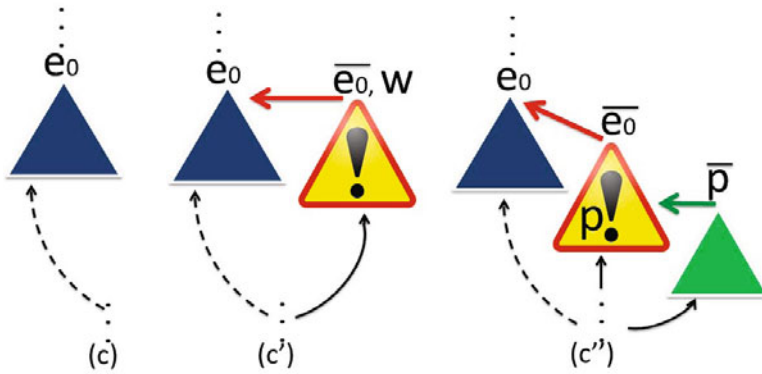


Fig. 3. (c) Selected plan. (c') Threat. (c'') Solution to (c'): Defeat-the-defeater.

## 5 Cooperative Distributed Planning Protocol in the MAPA Architecture

Figure 4 illustrates the planning protocol, which is mainly composed of three different cooperative distributed processes among the planning agents: Plan Generation, Plan Evaluation, and Plan Selection.

Different planning heuristics such as Z-LIFO [11], or the threat detect-&-solve [10] can be used to select the next open goal to solve. In our case, we will consider turn-based dialogues, a mechanism traditionally used in cooperative scenarios where agents only participate during their turn. Additionally, agents can also be modeled to put a veto on information or decisions of other agents. Agents are enumerated, and each process is implemented through a different argumentative dialogue.

### 5.1 Plan Generation

The input is both the selected plan  $\Pi_r$  and the selected open goal (flaw)  $\Phi$ , according to the Plan Selection process (see subsection 5.3) and open goal selection heuristic. The flaw  $\Phi$  can be referred to both goals and PlaThreats. The main goal of this process is to allow agents to propose a **set of refinement plans**

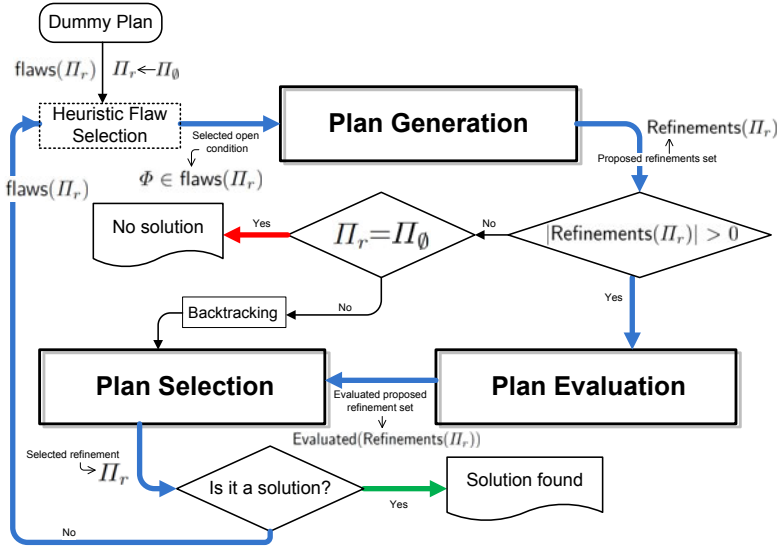


Fig. 4. Planning Protocol in the MAPA architecture

$\text{Refinements}(II_r)$ , where each  $II_r(\xi) \in \text{Refinements}(II_r)$  is a refinement step in the *POP Search Tree* that solves a selected flaw  $\Phi$  such that  $\Phi \in \text{flaws}(II_r)$  and  $\Phi \notin \text{flaws}(II_r(\xi))$ . Following, we explain the two steps involved in this process:

1. PROPOSALS ROUND: Each agent, at its turn, proposes alternative ways to achieve or derive  $\Phi$ . The process ends when all agents have had a turn. Refinements of a plan  $II_r$  are labeled as  $II_r^{(n,i)}(\xi)$ , where  $n \in \mathbb{Z}$  indicates the refinement proposal by the agent,  $i \in \mathbb{Z}$  represents the agent, and  $r \in \mathbb{Z}$  represents the selected plan by the Plan Selection process. Note that, at each turn, an agent can propose as many plans as possible from its knowledge.
2. LEARNING ROUND: Each agent updates its set of actions with the new actions which appear in the refinements proposed by other agents.

The output of this process is a set of plans  $\text{Refinements}(II_r)$  where each  $II_r(\xi) \in \text{Refinements}(II_r)$  extends  $II_r$ . If  $|\text{Refinements}(II_r)| > 0$ , i.e. there is at least one refinement plan, it is used as an input to the Plan Evaluation process (see section 5.2). If  $|\text{Refinements}(II_r)| = 0$ , i.e. there is not any proposal to solve the flaw  $\Phi$ , a backtracking step is performed, pruning the current base plan  $II_r$ .

### 5.2 Plan Evaluation

Roughly, the problem stems from different agents discussing about a given plan; since these agents may have different initial facts and defeasible rules they may not agree on the evaluation of the plan at some step. Along with the *POP Search Tree* of the previous section, the MAPA architecture also considers the notion of *POP Evaluation Tree*.

**Definition 1.** *POP Evaluation Tree:* Let  $\Pi_r(\xi)$  be a refinement of plan  $\Pi_r$  from the previous process. A POP Evaluation Tree for  $\Pi_r(\xi)$ , denoted  $\mathcal{T}_{\Pi_r(\xi)}$ , where there is at least one argument step  $(\mathcal{A}, \ell, \beta) \in \mathcal{SL}(\Pi_r(\xi))$ , is defined as follows:

- The root of the tree is labeled with the plan  $\langle \Pi_r(\xi) \rangle$ .
- Each node of the first level  $\langle \Pi_r(\xi, \xi') \mid \xi' = \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta)) \rangle$ , is a new plan extending  $\Pi_r(\xi)$  with some new defeater  $\mathcal{B}$  that attacks  $\mathcal{A}$ , discovering a new **ArgThreat** in  $\Pi_r(\xi)$ .
- Each node of the second level  $\langle \Pi_r(\xi, \xi', \xi'') \mid \xi'' = \text{Defeater}(\mathcal{C}, \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta))) \rangle$ , is a new plan extending  $\Pi_r(\xi, \xi')$  with some new defeater  $\mathcal{C}$  that attacks  $\mathcal{B}$  (*Defeat-the-Defeater*), solving the **ArgThreat** in  $\Pi_r(\xi)$ .

The input of this process is a set  $\text{Refinements}(\Pi_r)$  of plans proposed by the agents in the previous process. Each plan  $\Pi_r(\xi) \in \text{Refinements}(\Pi_r)$  represents the root of a new *POP Evaluation Tree*  $\mathcal{T}_{\Pi_r(\xi)}$ . Following, we explain the steps involved in this cooperative process:

1. **ATTACK ROUND:** It initiates an evaluation dialogue for the root plan of each  $\mathcal{T}_{\Pi_r(\xi)}$ , where each agent sends as many  $\langle \Pi_r(\xi, \xi') \mid \xi' = \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta)) \rangle$  as they know at their turn (Figure 5). If the agent does not know how to attack a root plan, then it will skip its turn.
2. **DEFENSE ROUND:** It allows the agents to propose ways  $\langle \Pi_r(\xi, \xi', \xi'') \mid \xi'' = \text{Defeater}(\mathcal{C}, \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta))) \rangle$  to solve discovered **ArgThreats** in each  $\langle \Pi_r(\xi, \xi') \mid \xi' = \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta)) \rangle$ . This round only applies to those *POP Evaluation Trees* which have discovered threats (Figure 5).
3. **LEARNING ROUND:** In this stage, each agent will update its sets of initial facts and defeasible rules, by extracting literals  $\ell \in \text{Lit}$  and defeasible rules, from arguments' bases and plan proposals. Unlike the previous Plan Generation process, where agents learn abilities as actions, this step is focused exclusively on the literals and defeasible rules.
4. **EVALUATION:** This stage marks each plan  $\Pi_r(\xi) \in \text{Refinements}(\Pi_r)$  as an undefeated plan, in case that defeater plans  $\langle \Pi_r(\xi, \xi') \mid \xi' = \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta)) \rangle$  have not been discovered, or if they have been discovered but there is a plan  $\langle \Pi_r(\xi, \xi', \xi'') \mid \xi'' = \text{Defeater}(\mathcal{C}, \text{Defeater}(\mathcal{B}, (\mathcal{A}, \ell, \beta))) \rangle$ . Otherwise,  $\Pi_r(\xi)$  is marked as a defeated plan.

The process ends when all the plans in  $\text{Refinements}(\Pi_r)$  have been evaluated. The output of this process is  $\text{Evaluated}(\text{Refinements}(\Pi_r))$ , **the set of evaluated plans** (Figure 5). As shown in the next process, undefeated plans, which constitute the most promising refinements to reach a solution, are preferred to defeated plans. However, defeated plans are kept, since each non-resolved attack could be resolved in a subsequent evaluation process.

### 5.3 Plan Selection

Plan selection can be done through the application of standard domain-independent heuristics for evaluating plans. These heuristics approximate the cost of a solution

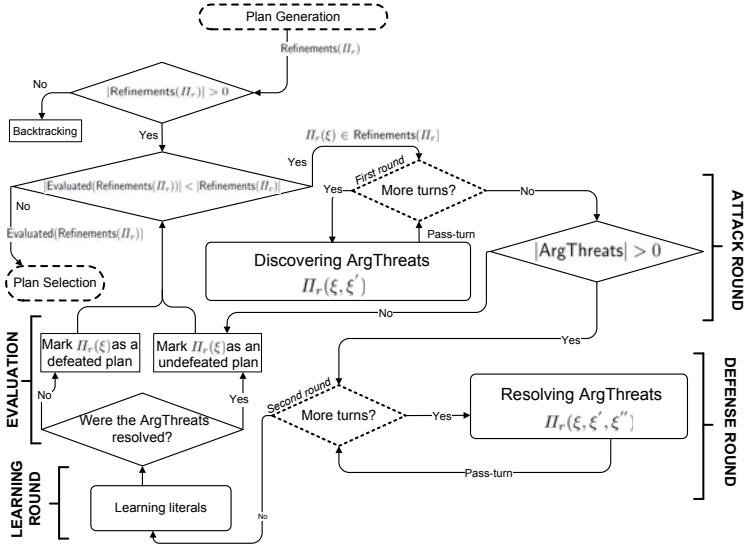


Fig. 5. Plan Evaluation protocol overview

plan in terms of the number of actions, the cost or the duration of the actions. Using this type of heuristics as the *standard rating* ( $R_S$ ) for plan assessment will ignore the dynamic multi-agent nature of the MAPA architecture, where a set of preferences is assumed by each agent. Therefore, a second rating based on the agents' preferences is necessary. We will refer to it as the *preference rating* ( $R_F$ ). Moreover, a third rating in terms of trust in cooperative planning is justified in [15] as a judgement about the risk attached to each component in the plan requiring cooperation, which we will call *trust rating* ( $R_T$ ).  $R_T$  depends on the trust in (i) each argument step and (ii) each action step in the plan, where (i) is the trustworthiness (reputation) of the information sources which are used by the agent in order to have a perception of the environment (coded as facts and defeasible rules [10]), and, (ii) is the result of dividing the number of times the action is successfully executed into the total number of executions of the action. The MAPA architecture stores the execution of each action as a new case [1], recorded as successful if the action is executed correctly, or failure if the action failed during the execution. The success or failure of an action is determined by the achievement of the action effects. For simplicity, we only consider trust in action steps.

Unlike the Plan Generation and the Plan Evaluation process, where agents reason about agent facts, defeasible rules and actions, here agents reason about standard ratings, preference ratings, and trust ratings, considering a compromise between the desire to minimize the computational overhead and that of maximizing the quality of the plan. This process receives as input the set of evaluated plans  $\text{Evaluated}(\text{Refinements}(\Pi_r))$  from the Plan Evaluation process and a set of previously not-selected partial plans  $\text{OtherRefinements}$ , in order to select a new plan  $\Pi_r \in \{\text{Evaluated}(\text{Refinements}(\Pi_r)) \cup \text{OtherRefinements}\}$  as output. Following, we explain the steps involved in this process:

1. **PLAN FILTERING:** The aim is to guide the plan search by selecting the best subset  $\text{FilteredPlans} \subseteq \{\text{Evaluated}(\text{Refinements}(\Pi_r)) \cup \text{OtherRefinements}\}$  (as candidate plans), according to the highest  $R_S(\Pi)$  and  $R_T(\Pi)$ , such that  $\Pi \in \{\text{Evaluated}(\text{Refinements}(\Pi_r)) \cup \text{OtherRefinements}\}$ <sup>2</sup>, where:
  - $R_S(\Pi) = (\text{cost}(\Pi) + \text{heuristic}(\Pi))$ , where  $\text{heuristic}(\Pi)$  is a heuristic estimation of the cost of reaching a solution plan  $\Pi^*$  from  $\Pi$ , and,
  - $R_T(\Pi)$  is the product of the trust values of the action steps in  $\Pi$ .
2. **PLAN RANKING:** The agents ( $i \in \{1, 2, \dots, k\}$ ) calculate their preference ratio for each candidate plan  $\Pi_n \in \text{FilteredPlans}$ . For this purpose, they review whether each action  $a \in A(\Pi_n)$  is preferred by them. If an action  $a_1 \in A(\Pi_n)$  is preferred ( $(a_1, d_1) \in F_i \mid d_1 > 50$ ), then they increase by one the value  $R_F^i(\Pi_n)$ ; if they do not prefer an action  $a_2 \in A(\Pi_n)$ ,  $((a_2, d_2) \in F_i \mid d_2 \leq 50)$ , then they subtract one unit from  $R_F^i(\Pi_n)$ , and otherwise they keep  $R_F^i(\Pi_n)$  unchanged. This stage, which simulates a internal reasoning process for or against to select each plan  $\Pi_n$ , allows each agent to establish a preference relation between the plans in  $\text{FilteredPlans}$ .
3. **PLAN NEGOTIATION:** Since each agent has identified its preferred candidate plans, now the purpose of the negotiation is to reach an **agreement** about the next base plan  $\Pi_r \in \text{FilteredPlans}$ . This stage can range from a simple voting process to a more sophisticated negotiation mechanism.

Finally,  $\{\text{NonSelectedPlans} \subseteq \text{Evaluated}(\text{Refinements}(\Pi_r)) \mid \Pi_r \notin \text{NonSelectedPlans}\}$  is added to the set  $\text{OtherRefinements}$ , and the process returns the agreed plan  $\Pi_r$ . If  $\Pi_r$  is not a solution, the control will be passed to the Heuristic Flaw Selection (see Figure 4). Otherwise, the planning process will end successfully.

## 6 Evaluating the MAPA Architecture within the Context of a Transit Journey Planning Service

Transit users generally know their origin and destination cities. Based on the schedules provided by the transit agencies, users choose the best routes that match their travel needs. For this purpose, a Transit Journey Planning Service (TJPS) (a specialized electronic search engine) is used to find the best route between two locations by using some means of transportation. TJPSs are being widely used by transit agencies accessed through a web user interface on a computer terminal to support clients' requests on public transport information. Most of the existing TJPSs, provided by transit agencies and companies (Google Transit Planner, Transport Direct, Transport for London, Trip Planning Tool etc.)<sup>3</sup>, are based on static schedule data. To the best of our knowledge, these centralized planners (i) do not react to environmental changes such as bad weather, traffic jams or bad railroads, and therefore they do not provide support to defeasible

<sup>2</sup> Undeclared plans are preferred over defeated plans.

<sup>3</sup> <http://www.google.com/transit>, <http://www.transportdirect.info>,  
<http://www.journeyplanner.org>, <http://www.networkedtraveler.org>

reasoning, and (ii) are not able to work in a cooperative distributed environment so there is no choice for exchanging information between them.

However, defeasible reasoning is becoming an increasingly important feature in many environments where context awareness is not fully specified. The work in [5] presents a potential application for distributed defeasible reasoning in ambient computing environments, where the ambient agents, who have different viewpoints, have to face the available context. Similarly, defeasible reasoning is also being applied to semantic web and e-commerce [17]. Here, we present a novel application of cooperative distributed defeasible planning to a TJPS problem.

The MAPA architecture is implemented in Magentix 2<sup>4</sup>, a platform for open Multiagent Systems based on the Apache Qpid<sup>5</sup> implementation of AMQP<sup>6</sup> for communication between agents. This platform incorporates a security module which provides key features regarding security, privacy, openness and interoperability not offered by other current agent platforms.

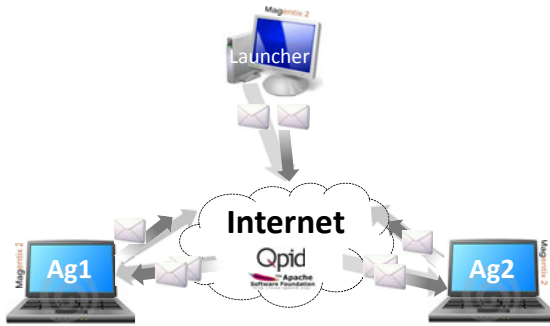


Fig. 6. Deploying the MAPA architecture

## 6.1 Preliminaries

According to the multi-agent systems paradigm, we have implemented a CIS as a collection of software agents (Figure 6) in the MAPA architecture. Each agent simulates an information system, and interacts with the others so as to achieve the common goals, thus forming a multi-agent society. More specifically, we consider a scenario with six different cities and two geographically distributed transit agencies, *Ag1* (Greece transit agency) and *Ag2* (UK transit agency), aimed at providing a customer with a plan to travel from *London* to *Athens* (Figure 7). The agencies are implemented as agents and have different knowledge (knowledge is fully distributed), so two pieces of information derived from each agent may appear to be contradictory. There are several ways to travel between both cities: via car, ship, train or plane. Let's assume that *Ag1* uses *BBC News* as a source of information, but *Ag2* prefers *CNN News* to keep up to date, and

<sup>4</sup> <http://www.gti-ia.dsic.upv.es/sma/tools/magentix2/index.php>

<sup>5</sup> <http://qpid.apache.org/>

<sup>6</sup> <http://www.amqp.org>

both agree on finding a plan that minimizes the journey duration. The planning tasks of the agents are defined in Figure 8, where we consider propositional STRIPS [8] planning representation.

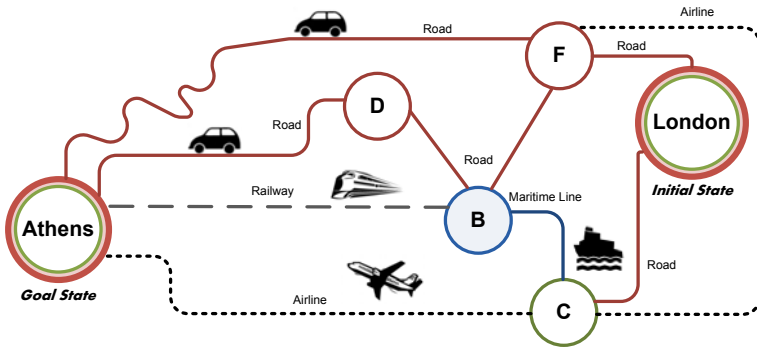


Fig. 7. Scenario of the application example

In what follows, we define the meaning of each literal and action. Literals:

- A, L - Athens, London; B, C, D, F - Other cities,
- *cus*, *car*, *tra*, *pl*, *shi* - a customer, a car, a train, a plane, a ship,
- *r*, *rl*, *al*, *ml* - a road, a railway, an airline company, a maritime line,
- *bw*, *sn*, *wg*, *va*, *ds*, *aeo* - bad weather, snow, wind gusts, volcano ash cloud, dangerous situation, airplane engines work well (after test),
- *br*, *ll*, *esf*, *fp* - bad railroad, landslides, electrical supply failure, flying panic,
- *h*, *tj*, *kudBBC*, *kudCNN* - holidays, traffic jam, kept up to date by BBC news, kept up to date by CNN news, and,
- $\mu_C$ ,  $\mu_P$ ,  $\mu_T$ ,  $\mu_S$  - moved car, moved plane, moved train and moved ship.

Actions are the following (notation:  $X(\alpha) \xleftarrow{\alpha} P(\alpha)$ , i.e. the action effects are indicated on the left side, while the action preconditions on the right side):

1.  $mP(pl, x, y)$ : moving plane 'pl' from location 'x' to 'y' takes 2 time units and 400 cost units.
2.  $mT(tra, x, y)$ : moving train 'tra' from location 'x' to 'y' takes 6 time units and 200 cost units.
3.  $mS(shi, x, y)$ : moving ship 'shi' from location 'x' to 'y' takes 3 time units and 100 cost units.
4.  $fMc(car, x, y)$ : fast-moving car 'car' from location 'x' to 'y' takes 8 time units and 80 cost units.

## 6.2 Implementation

The planning process starts with an empty plan  $\Pi_\emptyset = \{\alpha_\Psi \prec \alpha_G\}$  and  $\text{flaws}(\Pi_\emptyset) = \{(at\ cus\ A)\}$ . First, the MAPA architecture enters the **Plan Generation** process, where four plans are suggested: i) taking the car between D and A,  $\Pi_\emptyset^{(1, Ag1)}(\xi)$ ,

$$\Psi_{Ag1} = \{ (wg\ B\ A); aeo; kudBBC; (at\ cus\ L); fp; (at\ pl\ C); (at\ car\ L); (link\ al\ C\ A); (link\ r\ D\ A); \dots \}$$

$$\Psi_{Ag2} = \{ kudCNN; fp; (at\ cus\ L); (at\ tra\ B); (at\ shi\ C)\ (link\ rl\ B\ A); (link\ ml\ C\ B); (link\ r\ F\ A); \dots \}$$

$$\Delta_{Ag1} = \left\{ \begin{array}{l} \{ (at\ pl\ ?y), (at\ cus\ ?y) \} \rightarrow (\mu_P\ ?x\ ?y); \{ \sim(at\ tra\ ?y), \sim(at\ cus\ ?y) \} \rightarrow \{ (\mu_T\ ?x\ ?y), (br\ ?x\ ?y) \}; \\ \{ (at\ car\ ?y), (at\ cus\ ?y) \} \rightarrow (\mu_C\ ?x\ ?y); \{ \sim(at\ shi\ ?y), \sim(at\ cus\ ?y) \} \rightarrow \{ (\mu_S\ ?x\ ?y), (ss\ ?x\ ?y) \}; \\ (br\ ?x\ ?y) \rightarrow (esf\ ?x\ ?y); (esf\ ?x\ ?y) \rightarrow (sn\ ?x\ ?y); (sn\ B\ A) \rightarrow kudBBC; \sim(va\ C\ A) \rightarrow aeo; \dots \end{array} \right\}$$

$$\Delta_{Ag2} = \left\{ \begin{array}{l} \{ \sim(at\ pl\ ?y), \sim(at\ cus\ ?y) \} \rightarrow \{ (\mu_P\ ?x\ ?y), (ds\ ?x\ ?y) \}; \{ (at\ tra\ ?y), (at\ cus\ ?y) \} \rightarrow (\mu_T\ ?x\ ?y); \\ \{ \sim(at\ car\ ?y), \sim(at\ cus\ ?y) \} \rightarrow \{ (\mu_C\ ?x\ ?y), (tj\ ?x\ ?y) \}; \{ (at\ shi\ ?y), (at\ cus\ ?y) \} \rightarrow (\mu_S\ ?x\ ?y); \\ (ds\ ?x\ ?y) \rightarrow (va\ ?x\ ?y); (va\ C\ A) \rightarrow kudCNN; \sim(ll\ ?x\ ?y) \rightarrow \sim(bw\ ?x\ ?y); \\ \sim(ll\ ?x\ ?y) \rightarrow \sim(bw\ ?x\ ?y); \sim(bw\ B\ A) \rightarrow kudCNN; \sim(sn\ B\ A) \rightarrow kudCNN; \\ (tj\ ?x\ ?y) \rightarrow \{ (h\ ?x)\ (link\ r\ ?x\ ?y) \}; (h\ F) \rightarrow kudCNN; \dots \end{array} \right\}$$

$$A_{Ag1} = \left\{ \begin{array}{l} 1. (\mu_C\ ?x\ ?y) \xleftarrow{fMc} \{ (link\ r\ ?x\ ?y), (at\ car\ ?x), (at\ cus\ ?x) \} \\ 2. (\mu_P\ ?x\ ?y) \xleftarrow{mP} \{ (link\ al\ ?x\ ?y), (at\ pl\ ?x), (at\ cus\ ?x) \} \end{array} \right\}$$

$$A_{Ag2} = \left\{ \begin{array}{l} 3. (\mu_T\ ?x\ ?y) \xleftarrow{mT} \{ (link\ rl\ ?x\ ?y), (at\ tra\ ?x), (at\ cus\ ?x) \} \\ 4. (\mu_S\ ?x\ ?y) \xleftarrow{mS} \{ (link\ ml\ ?x\ ?y), (at\ shi\ ?x), (at\ cus\ ?x) \} \end{array} \right\}$$

$$F_{Ag2} = \{ (mT\ 90); (mP\ 0); (mS\ 60) \}$$

$$F_{Ag1} = \{ (fMc\ 70); (mT\ 80); (mP\ 5) \}$$

$$G = \{ (at\ cus\ A) \}$$

**Fig. 8.** Initial facts, defeasible rules, actions, preferences and common goals

or ii) between F and A,  $\Pi_{\emptyset}^{(2,Ag1)}(\xi)$ , iii) taking the train between B and A,  $\Pi_{\emptyset}^{(1,Ag2)}(\xi)$ , and iv) taking the plane between C and A,  $\Pi_{\emptyset}^{(3,Ag1)}$ . We only show the plan iv)  $\Pi_{\emptyset}^{(3,Ag1)}(\xi) = \{ (mP, (\mu_P\ C\ A), \mathcal{A}^{Ag1}), (\mathcal{A}^{Ag1}, (at\ cus\ A), \alpha_G) \}$  where  $\mathcal{A}^{Ag1} = (\{ (at\ cus\ A) \}, \{ (at\ cus\ A) \rightarrow (\mu_P\ C\ A) \})$  (see Figure 10(a)). The agents learn the actions they did not know from these plans.

Second, the **Plan Evaluation** process starts, where: i)  $\Pi_{\emptyset}^{(1,Ag1)}(\xi)$  is not attacked by any defeater and it is labeled as an undefeated plan. ii)  $\Pi_{\emptyset}^{(2,Ag1)}(\xi)$  is attacked because city F is on holiday, so a traffic jam can be expected in the road between F and A, and, therefore, the effects of the action  $fMc$  may not be satisfied. Since there are not proposals to solve this **ArgThreat**,  $\Pi_{\emptyset}^{(2,Ag1)}(\xi)$  is labeled as a defeated plan. iii)  $\Pi_{\emptyset}^{(1,Ag2)}(\xi)$  receives one attack because snow is expected between B and A, so an electrical failure that damages the railroad between B and A might occur. If this happens, the effects of the action  $mT$  may not be satisfied. Here,  $Ag2$  proposes a Defeat-the-defeater, which justifies that snow conditions are not expected between B and A, and then  $\Pi_{\emptyset}^{(1,Ag2)}(\xi)$  is labeled as undefeated plan. iv)  $Ag2$  attacks  $\Pi_{\emptyset}^{(3,Ag1)}(\xi)$  with  $\langle \Pi_{\emptyset}^{(3,Ag1)}(\xi, \xi') \mid \xi' = \text{Defeater}(\mathcal{B}^{Ag2}, (\mathcal{A}^{Ag1}, (at\ cus\ A), \alpha_G)) \rangle$  where  $\mathcal{B}^{Ag2} = (\{ \sim(at\ cus\ A) \}, \{ \sim(at\ cus\ A) \rightarrow \{ (\mu_P\ C\ A), (ds\ C\ A) \}; (ds\ C\ A) \rightarrow (va\ C\ A); (va\ C\ A) \rightarrow kudCNN \})$  (see Figure 9) because the volcano ashes are expected between the city C and A according to the *CNN News*, but  $Ag1$  moves against  $(ds\ C\ A)$  with  $\langle \Pi_{\emptyset}^{(3,Ag1)}(\xi, \xi', \xi'') \mid \xi'' = \text{Defeater}(\mathcal{C}^{Ag1}, \text{Defeater}(\mathcal{B}^{Ag2}, (\mathcal{A}^{Ag1}, (at\ cus\ A), \alpha_G))) \rangle$  where  $\mathcal{C}^{Ag1} =$



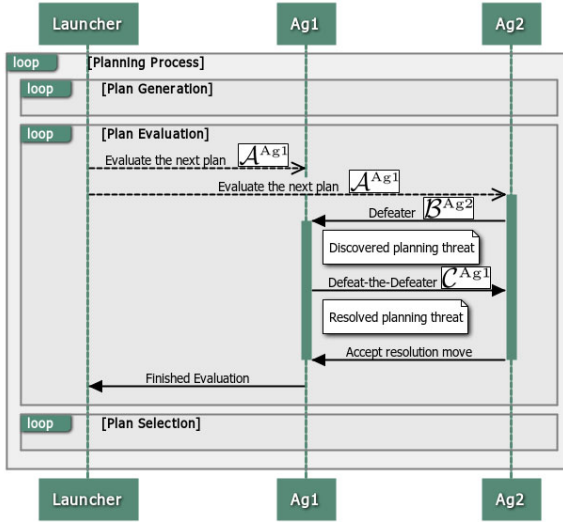


Fig. 9. Discussing about the plan  $\Pi_0^{(3,Ag1)}(\xi)$  in the Plan Evaluation process

( $\{\sim(va C A)\}, \{\sim(va C A) \leftarrow aeo\}$ ) (see Figure 9). It is a Defeat-the-defeater resolution move since  $\sim \text{concl}(C^{Ag1}) \in \text{literals}(B^{Ag2})$  (see Figure 10(a'')), and then  $\Pi_0^{(3,Ag1)}(\xi)$  is labeled as an undefeated plan. The agents learn the literals and defeasible rules, they do not know at the beginning of the turn.

Third, the **Plan Selection** process starts. The best subset of plans is defined as  $\text{FilteredPlans} = \{\Pi_0^{(1,Ag2)}(\xi), \Pi_0^{(3,Ag1)}(\xi)\}$ , since  $\Pi_0^{(2,Ag1)}(\xi)$  was labeled as a defeated plan and  $\text{heuristic}(\Pi_0^{(1,Ag1)}(\xi))$  returns a high value. Finally, agents

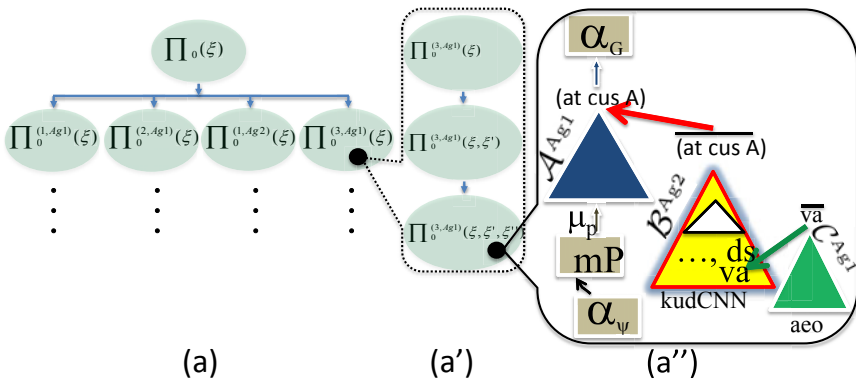


Fig. 10. Screenshots: (a) The POP Search Tree. (a') The POP Evaluation Tree for the plan  $\Pi_0^{(3,Ag1)}(\xi)$ . (a'') Viewing the content of the plan  $\Pi_0^{(3,Ag1)}(\xi, \xi', \xi'')$ .

choose  $\Pi_r = \Pi_{\emptyset}^{(1, Ag^2)}(\xi)$  as they prefer to take the train because of their fear of flying. For space reasons, we omit the rest of the plan search.

## 7 Conclusions and Future Work

We have presented MAPA, a decentralized architecture for cooperative planning in multiagent DeLP-POP, dealing with the qualification problem. It is implemented as three independent cooperation processes between agents of a team who propose, criticize, defend and select alternative plans by means of arguments and actions. For future work, we intend to work in several directions: extending MAPA to other multi-agent scenarios like argumentation-based negotiation or to temporal planning [19]; and evaluating MAPA in applications of dynamic networked cooperative business processes and knowledge-sharing, including the ability to work with and within complex supply chains. Finally, evaluating the efficiency and effectiveness of the MAPA architecture.

**Acknowledgments.** This work is mainly supported by FPU grant reference AP2009-1896 awarded to Sergio Pajares Ferrando and projects TIN2008-06701-C03-01, Consolider Ingenio 2010 CSD2007-00022, and PROMETEO/2008/051.

## References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1), 39–59 (1994)
2. Amgoud, L.: A formal framework for handling conflicting desires. In: Nielsen, T.D., Zhang, N.L. (eds.) *ECSQARU 2003*. LNCS (LNAI), vol. 2711, pp. 552–563. Springer, Heidelberg (2003)
3. Belesiotis, A., Rovatsos, M., Rahwan, I.: Agreeing on plans through iterated disputes. In: *9th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2010)*, pp. 765–772 (2010)
4. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. *Artificial Intelligence* 171(10-15), 619–641 (2007)
5. Bikakis, A., Antoniou, G.: Distributed defeasible contextual reasoning in ambient computing. In: Aarts, E., Crowley, J.L., de Ruyter, B., Gerhäuser, H., Pflaum, A., Schmidt, J., Wichert, R. (eds.) *AmI 2008*. LNCS, vol. 5355, pp. 308–325. Springer, Heidelberg (2008)
6. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2), 281–300 (1997)
7. Dung, P.M.: On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2), 321–358 (1995)
8. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4), 189–208 (1971)
9. García, A.J., Simari, G.R.: Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(2), 95–138 (2004)
10. García, D.R., García, A.J., Simari, G.R.: Defeasible reasoning and partial order planning. In: Hartmann, S., Kern-Isberner, G. (eds.) *FoIKS 2008*. LNCS, vol. 4932, pp. 311–328. Springer, Heidelberg (2008)

11. Gerevini, A., Schubert, L.: Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research* 5, 95–137 (1996)
12. Ghallab, M., Laruelle, H.: Representation and control in ixtet, a temporal planner. In: 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994), vol. 94, pp. 61–67 (1994)
13. Ginsberg, M.L., Smith, D.E.: Reasoning about action II: The qualification problem. *Artificial Intelligence* 35(3), 311–342 (1988)
14. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000), pp. 140–149 (2000)
15. Ibbott, C.J., O’Keefe, R.M.: Trust, planning and benefits in a global interorganizational system. *Information Systems Journal* 14(2), 131–152 (2004)
16. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in interplanetary space: Theory and practice. In: 5th International Conference on Artificial Intelligence Planning and Scheduling (ICAPS 2000), pp. 177–186 (2000)
17. Kontopoulos, E., Bassiliades, N., Antoniou, G.: Visualizing semantic web proofs of defeasible logic in the dr-device system. *Knowledge-Based Systems* 24(3), 406–419 (2011)
18. Mecella, M., Scannapieco, M., Virgillito, A., Baldoni, R., Catarci, T., Batini, C.: Managing data quality in cooperative information systems. In: Chung, S., et al. (eds.) *CoopIS 2002, DOA 2002, and ODBASE 2002*. LNCS, vol. 2519, pp. 486–502. Springer, Heidelberg (2002)
19. Pajares, S., Onaindía, E.: Temporal defeasible argumentation in multi-agent planning. In: 22nd International Joint Conferences on Artificial Intelligence (IJCAI 2011), pp. 2834–2835 (2011)
20. Pardo, P., Pajares, S., Onaindía, E., Godo, L., Dellunde, P.: Multiagent argumentation for cooperative planning in delp-pop. In: 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), pp. 971–978 (2011)
21. Penberthy, J.S., Weld, D.: Ucpop: A sound, complete, partial order planner for adl. In: *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning (KR 1992)*, pp. 103–114 (1992)
22. Pollock, J.: Defeasible planning. In: *Proceedings of the AAAI Workshop, Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*. Carnegie Mellon University (June 1998)
23. Rahwan, I., Amgoud, L.: An argumentation-based approach for practical reasoning. In: *International Workshop on Argumentation in Multi-Agent Systems (ArgMAS)*, pp. 74–90 (2007)
24. Sapena, O., Torreño, A., Onaindía, E.: On the construction of joint plans through argumentation schemes. In: 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), pp. 1195–1196 (2011)
25. Smith, D.E., Frank, J., Jónsson, A.K.: Bridging the gap between planning and scheduling. *The Knowledge Engineering Review* 15(1), 47–83 (2000)
26. Tang, Y., Norman, T., Parsons, S.: A model for integrating dialogue and the execution of joint plans. In: *International Workshop on Argumentation in Multi-Agent Systems (ArgMAS)*, pp. 60–78 (2010)
27. Thimm, M.: Realizing argumentation in multi-agent systems using defeasible logic programming. In: *International Workshop on Argumentation in Multi-Agent Systems (ArgMAS)*, pp. 175–194 (2009)
28. Wooldridge, M.: *Agent-based software engineering*, vol. 144, pp. 26–37 (1997)

# A Case Retrieval Approach Using Similarity and Association Knowledge

Yong-Bin Kang<sup>1</sup>, Shonali Krishnaswamy<sup>1</sup>, and Arkady Zaslavsky<sup>1,2</sup>

<sup>1</sup> Faculty of IT, Monash University, Australia  
{yongbin.kang, shonali.krishnaswamy}@monash.edu

<sup>2</sup> ICT Centre, CSIRO, Australia  
arkady.zaslavsky@csiro.au

**Abstract.** Retrieval is often considered the most important phase in Case-Based Reasoning (CBR), since it lays the foundation for overall performance of CBR systems. Retrieval in CBR aims to retrieve relevant cases that can be successfully used for solving a new problem. To realize retrieval, CBR systems typically rely on a strategy that exploits similarity knowledge, and it is called similarity-based retrieval (SBR). In SBR, similarity knowledge approximates the usefulness of cases for solving a new problem. In this paper, we show that *association analysis* of stored cases can be used to strengthen SBR. We present a new approach for extracting and representing *association knowledge* from the cases using association rule mining. We propose a novel retrieval strategy USIMSCAR that qualitatively enhances SBR by leveraging both similarity and association knowledge. We demonstrate the significant advantages of using USIMSCAR over SBR through an experimental evaluation using medical datasets.

## 1 Introduction

Case-Based Reasoning (CBR) [1] is a widely researched technology for problem-solving in many application domains such as medical diagnosis [2], help-desk service [3], product recommendation [4], and classification [5]. The fundamental premise of CBR is that experience in the form of past cases can be leveraged to solve new problems. It is based on the fact that in many application domains, similar problems usually have similar solutions. In CBR, experiences are stored in a database known as a *case base*, and an individual experience is called a *case*.

Typically, there are four well-organized phases adopted in CBR [1]. The first phase is to *retrieve* one or several cases considered useful for solving a given target problem. Once useful cases are retrieved, the second phase is to *reuse* their solution information. The third phase is to *revise* or *adapt* the solution information to better fit the target problem if necessary. The fourth phase is to *retain* the new solution once it has been confirmed or validated.

Retrieval is often considered the most important phase in CBR, since it lays the foundation for overall performance of CBR systems [6]. Its aim is to retrieve *useful* cases that can be successfully used to solve a new problem. If retrieved

cases are not useful, CBR systems will not eventually produce any good solution for the new problem.

To accomplish the retrieval process, CBR systems typically rely on a retrieval strategy that exploits *similarity knowledge*. This strategy is often called *similarity-based retrieval* (SBR) [7]. In SBR, similarity knowledge aims to approximate the *usefulness* of stored cases as to solving a new problem [8]. This knowledge is usually encoded in the form of *similarity measures*, which are used to compute similarities between a new problem and the cases. By using similarity measures, SBR retrieves useful cases ranked by their similarities to the new problem. The solutions of these cases are then utilized to solve the problem. A limitation of SBR is that it tends to rely strongly on similarity knowledge only, ignoring other available knowledge that can be additionally leveraged for improving its retrieval performance [7,9,8].

While many kinds of learnt and induced knowledge (e.g. statistical [10], domain [8,11], adaptation [7,12] knowledge) have been utilized to enhance SBR, this paper proposes that association analysis of stored cases can improve traditional SBR. We propose a new retrieval strategy USIMSCAR that leverages *association knowledge* in conjunction with similarity knowledge. Association knowledge is formalized to represent certain interesting relationships, shared by a large number of cases, acquired from stored cases using *association rule mining*. The key idea of USIMSCAR thus lies in its usage of both similarity and association knowledge to deliver an improved retrieval strategy for CBR. We show USIMSCAR improves SBR through an experimental evaluation using medical datasets found in UCI ML Repository.

This paper is organized as follows. In Section 2, we present the motivation of our work. In Section 3, we present a background of similarity knowledge and association knowledge. In Section 4, we present our approach for formalizing association knowledge. In Section 5, we present USIMSCAR that leverages similarity and association knowledge. In Section 6, we evaluate USIMSCAR in comparison with SBR. In Section 7, we review the literature work related to this paper. In Section 8, we present our conclusion and future research directions.

## 2 Motivating Scenario

To illustrate our research motivation, we use a medical diagnosis scenario presented in [13]. Consider a case base  $\mathcal{D}$  in Table 1, where each case is represented as a pair of a problem and the corresponding solution. Each problem is described by five attributes (symptoms)  $A_1, \dots, A_5$ , and each solution by an attribute (a diagnosis)  $A_6$ . Our aim is to diagnose the correct disease ‘appendicitis’ for a new patient  $Q$ , since  $Q$  really suffered from ‘appendicitis’ as noted in [13].

To predict a diagnosis for  $Q$ , in principle, SBR may find similar cases to  $Q$  using a similarity metric. Assume that we use the following metric, used in [13], that measures the similarity between  $Q$  and each case  $P \in \mathcal{D}$ ,

**Table 1.** A patient case base

Cases ID	Local Pain ( $A_1$ )	Other Pain ( $A_2$ )	Fever ( $A_3$ )	Appetite Loss ( $A_4$ )	Age ( $A_5$ )	Diagnosis ( $A_6$ )	Similarity to $Q$
$P_1$	right flank	vomit	38.6	yes	10	appendicitis	0.631
$P_2$	right flank	vomit	38.7	yes	11	appendicitis	0.623
$P_3$	right flank	vomit	38.8	yes	13	appendicitis	0.618
$P_4$	right flank	sickness	37.5	yes	35	gastritis	<b>0.637</b>
$P_5$	epigastrium	nausea	36.8	no	20	stitch	0.420
$Q$	right flank	nausea	37.8	yes	14	?	
Weight	0.91	0.78	0.60	0.40	0.20		

$$SIM(Q, P) = \frac{\sum_{i=1}^n w_i \cdot sim(q_i, p_i)}{\sum_{i=1}^n w_i}, \quad (1)$$

$$sim(q_i, p_i) = \begin{cases} 1 - \frac{|q_i - p_i|}{A_i^{\max} - A_i^{\min}}, & \text{if } A_i \text{ is numeric,} \\ 1, & \text{if } A_i \text{ is discrete \& } q_i = p_i, \\ 0, & \text{otherwise,} \end{cases}$$

where  $w_i$  is a weight assigned to an attribute  $A_i$  of  $Q$  and  $P$  by domain experts,  $q_i$  and  $p_i$  are values of  $A_i$ ,  $n$  is the number of attributes of  $Q$  and  $P$  (i.e.  $n = 5$ ), and  $sim(q_i, p_i)$  is a function computing the similarity between  $q_i$  and  $p_i$ .

Once cases similar to  $Q$  are selected using the metric  $SIM$ , SBR determines a diagnosis for  $Q$ . Assume that SBR utilizes the most similar case to  $Q$ . As seen in Table 1,  $P_4$  is thus chosen, since it is the most similar case to  $Q$ . This means that a diagnosis choice for  $Q$  is ‘gastritis’. However, it turns out to be wrong, since  $Q$  actually suffered from ‘appendicitis’, as outlined above. This scenario shows that SBR has a limitation rooted in its nature—its ability tends to be strongly determined by the use of only similarity knowledge.

To address this issue, our idea is to formalize special knowledge, called *association knowledge*, that indicates how certain known problems are strongly associated with certain known solutions in a case base, and to exploit it during the retrieval process in CBR. For example, in Table 1, we observe that the values of  $A_5$  (Age) of three cases  $P_1$ ,  $P_2$ , and  $P_3$  are quite similar, which range from 10 to 13. These values are associated with ‘appendicitis’. Whereas the value of  $A_5$  of  $P_4$  is 35, and it is associated with ‘gastritis’. We note that the former association is *supported* by three cases, while the latter by only one case. If we quantify such associations, it may be usefully exploited in conjunction with similarity knowledge for solving the target problem. For example, assume that each of the above associations is quantified as the proportion of the cases that support it. The former association ( $as_1$ ) is then quantified as 0.6 (3/5), and the latter ( $as_2$ ) as 0.2 (1/5). In Table 1, we see that  $SIM(Q, P_4)$  is 0.637, and  $SIM(Q, P_1)$  is 0.631. We now measure the usefulness of each case with respect to  $Q$  by combining its similarity to  $Q$  and the quantified value of the association that the case supports. Suppose that the combination is implemented via the arithmetic multiplication operation. Then, we measure the usefulness of  $P_4$  as 0.127 by  $SIM(Q, P_4) \times as_2$ , and that of  $P_1$  as 0.379 by  $SIM(Q, P_1) \times as_1$ . Regarding the

computed usefulness, the higher the better. We thus conclude that  $P_1$  is more useful than  $P_4$  so that  $P_1$ 's diagnosis 'appendicitis' can be used as a diagnosis for  $Q$ . This meets our objective of this scenario. This paper presents how to extract and represent association knowledge as well as exploit this knowledge in conjunction with similarity knowledge in order to qualitatively enhance SBR.

### 3 Background to Research on Similarity and Association Knowledge

Prior to presenting the underpinnings of our proposed retrieval strategy, it is essential to provide a background of similarity and association knowledge. This section provides this background. We first present our case representation scheme, which is the basis for formalizing both similarity and association knowledge.

To represent cases, many CBR systems generally adopt well-known knowledge representation formalisms, such as *attribute-value pairs* and *structural representations* [14]. In our work, we choose the attribute-value pairs representation due to its simplicity, flexibility and popularity. Let  $A_1, \dots, A_m$  be attributes defined in a given domain. An *attribute-value pair* is a pair  $(A_i, a_i)$ , where  $A_i$  is an attribute (or feature<sup>1</sup>) and  $a_i$  is a value of  $A_{i \in [1, m]}$ . A *case*  $C$  is a pair  $C = (X, Y)$  where  $X$  is a problem, represented as  $X = \{(A_1, a_1), \dots, (A_{m-1}, a_{m-1})\}$ , and  $Y$  is the solution of  $X$ , represented as  $Y = (A_m, a_m)$ . We call an attribute  $A_m$  a *solution-attribute*. A *case base*  $\mathcal{D}$  is a collection of cases.

#### 3.1 Background to Research on Similarity Knowledge

Similarity knowledge is referred to as knowledge encoded via similarity measures computing the similarities between a new problem  $Q$  and stored cases. SBR mainly exploits this knowledge. Similarity knowledge represents a heuristic for estimating the usefulness of stored cases as to solving  $Q$ . Intuitively, the higher the similarity between  $Q$  and the case  $C$  is, the more useful  $C$  is as to solving  $Q$ . A formulation of similarity measures suitable for cases represented by attribute-value pairs is often based on a widely used principle. This is the *local-global principle* that decomposes a similarity measure by *local similarities* for individual attributes of the cases and a *global similarity* aggregating the local similarities [8]. An accurate local similarity function relies on attribute types. A global similarity function can be arbitrarily complex, but simple functions are usually used. A widely used form is *weighted average aggregation* [8] (e.g. *SIM* in Eq. (1)).

#### 3.2 Background to Research on Association Knowledge

We now present the fundamentals of association knowledge in a *CBR context*. These are *association rule mining* [15], *class association rule mining* [16], and the *soft-matching criterion* [17].

<sup>1</sup> To simplify the presentation, we do not distinguish between terms "attributes" and "features", and use these terms interchangeably.

*Association rule mining* [15] aims to mine certain interesting relationships, called *associations*, in a transaction database. It focuses on discovering a set of highly correlated features shared a large number of transactions in the database. Let  $I$  be a set of distinct literals, called *items*. A set of items  $X \subseteq I$  is called an itemset. Let  $\mathcal{D}$  be a set of transactions. Each transaction  $T \in \mathcal{D}$  is a set of items such that  $T \subseteq I$ . We say that  $T$  *contains* an itemset  $X$ , if  $X \subseteq T$  holds. Every *association rule* has two parts: an *antecedent* and a *consequent*. An association rule is an implication of the form  $X \rightarrow Y$ , where  $X \subseteq I$  is an itemset in the antecedent and  $Y \subseteq I$  is an itemset in the consequent, and  $X \cap Y = \emptyset$ . The rule  $X \rightarrow Y$  has *support*  $s$  in  $\mathcal{D}$  if  $s\%$  of transactions in  $\mathcal{D}$  contain  $X \cup Y$ . This holds in  $\mathcal{D}$  with *confidence*  $c$  if  $c\%$  of transactions in  $\mathcal{D}$  that contain  $X$  also contain  $Y$ . Association rule mining can also be used for discovering interesting relationships among stored cases. In a CBR context, a transaction is seen as a case, and an item is seen as an attribute-value pair. Referring to Table 1, we can mine a rule  $r_1 : (A_1, \text{right flank}) \rightarrow (A_2, \text{vomit})$ . Let  $X$  be an item  $(A_1, \text{right flank})$ . Let  $Y$  be an item  $(A_2, \text{vomit})$ . The support of  $r_1$  is 0.6, since  $X$  and  $Y$  occur together in three out of five cases in  $\mathcal{D}$ . The confidence of  $r_1$  is 0.75, since  $Y$  occurs in three out of four cases that contain  $X$  in  $\mathcal{D}$ . Apriori [15] is one of the traditional algorithms for association rule mining.

Consider a special subset of association rules whose consequents are restricted to a single target variable. Rules in this subset are called *class association rules* (cars) [16]. In a CBR context, cars can be seen as special association rules whose consequents are restricted to hold special items, formed as pairs of a “solution-attribute” (see earlier in this section) and its values. We call such an item a *solution-item*. Thus, a car has the form  $X \rightarrow y$ , where  $X \subseteq I$  an itemset in the antecedent and  $y \in I$  is a solution-item in the consequent. In Table 1, we can mine a car:  $r_2 : (A_2, \text{vomit}) \rightarrow (A_6, \text{appendicitis})$ . But the above rule  $r_1$  is not a car, since  $r_1$  does not contain any solution-item in the consequent. We here emphasize that the aim of building association knowledge is to formalize special knowledge encoding how certain attribute-value pairs of known problems are *associated* with known solutions in a case base. For the purpose, we will use the form of cars, since it is suited well for it. Note that the car  $X \rightarrow y$  encodes an association between an itemset  $X$ , holding attribute-value pairs of known problems, and a solution-item  $y$  holding an attribute-value pair of a known solution.

Consider the association rule  $X \rightarrow Y$ . A limitation of traditional association rule mining algorithms (e.g. Apriori [15]) is that itemsets  $X$  and  $Y$  are discovered based on the equality relation. Unfortunately, when dealing with items similar to each other, these algorithms may perform poorly. For example, consider the sales database of a supermarket. Apriori cannot find rules like “80% of the customers who buy products similar to milk (e.g. cheese) and products similar to eggs (e.g. mayonnaise) also buy bread.” To address this issue, the SoftApriori algorithm [17] was proposed. It uses the *soft-matching criterion*, where itemsets in the antecedent and the consequent are found using *similarity assessment*. By employing the concept of similarity, the soft-matching criterion can be used to model richer relationships among features of cases than the equality relation [17].



## 4 Association Knowledge Formalization

Association knowledge is encoded via special rules that are “cars” whose antecedents are determined based on the “soft-matching criterion”. We call these rules *soft-matching class association rules (scars)*. A scar represents a strongly evident correlation, between certain attribute-value pairs of known problems and a known solution shared by a significant number of relevant cases.

Let  $\mathcal{D}$  be a set of cases, where each case is characterized by attributes  $A_1, \dots, A_m$ . Based on our case representation scheme, presented in Section 3, we call a pair  $(A_i, a_i)_{1 \leq i \leq m-1}$  an *item*. We call a pair  $(A_m, a_m)$  a *solution-item*. Let  $I$  be a set of items. A set  $L \subseteq I$  with  $k = |L|$  is called a  $k$ -itemset or simply an itemset. Let  $sim(x, y)$  be a function computing the similarity between two items  $x, y \in I$ . We say that  $x$  and  $y$  are similar, iff  $sim(x, y) \geq a$  *user-specified minimum similarity (minsim)*. Let  $x$  be an item  $(A_2, 38.6)$ . Let  $y$  be an item  $(A_2, 38.7)$ . Assuming a similarity function for an attribute  $A_2$  is defined as  $sim(x, y) = 1 - \frac{|x-y|}{40}$ ,  $sim(x, y)$  is 0.998. Given two itemsets  $X, Y \subseteq I$  ( $|X| \leq |Y|$ ),  $softSuppR(X, Y)$  is a function defined as  $\sum \frac{sim(x,y)}{|X|}$ , where  $x \in X$  and  $y \in Y$  are two items characterized by the *same* attribute. We say that  $X$  is a soft-subset of  $Y$  ( $X \subseteq_{soft} Y$ ), iff  $softSuppR(X, Y) \geq minsim$ ; or  $Y$  *softly contains*  $X$ . The problem described in each case  $C \in \mathcal{D}$  is also seen as a  $k$ -itemset with  $k = |m - 1|$ , since it is represented as  $\{(A_1, a_1), \dots, (A_{m-1}, a_{m-1})\}$ . The *soft-support-sum* of an itemset  $X$  regarding  $\mathcal{D}$  is defined as  $softSuppSum(X) = \sum_{C \in \mathcal{D}} softSuppR(X, C)$ , where  $X \subseteq_{soft} C$ . The *soft-support* of  $X$  is defined as  $softSupp(X) = \frac{softSuppSum(X)}{|\mathcal{D}|}$ . The *soft-support* for a scar  $X \rightarrow y$  is defined as the fraction of cases in  $\mathcal{D}$  that softly contain an itemset  $X$  and contain a solution-item  $y$ . The *soft-confidence* of this rule is defined as the fraction of cases in  $\mathcal{D}$  that softly contain  $X$  also contain  $y$ . In this paper, a *ruleitem* is of the form  $\langle X, y \rangle$  and basically represents a scar  $X \rightarrow y$ .

The key operation for scars mining is to find all ruleitems that have soft-supports  $\geq minsupp$  (*a user-specified minimum support*). We call such ruleitems *frequent* ruleitems. For all the ruleitems that have the same itemset in the antecedent, one with the highest *interestingness* is chosen as a *possible rule (PR)*. To measure the interestingness of association rules, the support and confidence criteria are typically used. On some occasions, a combination of them is used. Often, a rationale for doing so is to define a single optimal interestingness by leveraging their correlations. We choose an interestingness measure that combines soft-support and soft-confidence such that they are monotonically related. We thus choose the Laplace measure [18]. Given a ruleitem  $r : X \rightarrow y$ , its Laplace measure  $Laplace(r)$  can be denoted as  $\frac{N \cdot softSupp(X \rightarrow y) + 1}{N \cdot softSupp(X \rightarrow y) / softConf(X \rightarrow y) + 2}$ , where  $N = |\mathcal{D}|$ . If  $Laplace(r) \geq a$  *user-specified minimum level of interesting (min-interesting)*, we say  $r$  is *accurate*. A candidate set of scars consists of all the PRs that are frequent and accurate.

Let  $k$ -ruleitem be a ruleitem whose antecedent has  $k$  items. Let  $F_k$  be a set of frequent  $k$ -ruleitems. In  $F_k$ , each ruleitem  $r : X \rightarrow y$  has two fields:  $r.anteSoftSuppSum$  storing soft-support-sum of ruleitems in  $\mathcal{D}$  that softly

```

1:  $F_1 = \text{findFrequentRuleItems}(\mathcal{D});$ 
2:  $SCAR_1 = \text{genRules}(F_1);$ 
3:  $k = 2;$ 
4: while  $F_{k-1} \neq \emptyset$  do
5:    $CR_k = \text{generateCandidatesRuleItems}(F_{k-1});$ 
6:   for each case  $C \in \mathcal{D}$  do
7:     for each  $r : X \rightarrow y \in CR_k$  do
8:       if  $r \subseteq_{\text{soft}} C$  then
9:          $r.\text{anteSoftSuppSum} += \text{softSuppR}(X, C);$ 
10:        if  $y = C.\text{solution}$  then
11:           $r.\text{softSuppSum} += \text{softSuppR}(X, C);$ 
12:        end if
13:      end if
14:    end for
15:  end for
16:   $F_k = \{r \in CR_k \mid \text{softSupp}(r) \geq \text{minsupp}\};$ 
17:   $SCAR_k = \text{genRules}(F_k);$ 
18:   $k++;$ 
19: end while
20:  $SCARS = \bigcup_{k \geq \text{minitemsize}} SCAR_k;$ 
21:  $\text{prSCARS} = \text{pruneRules}(SCARS);$ 
22: Return  $\text{prSCARS};$ 

```

**Algorithm 1.** The algorithm for scars mining

contain  $X$ , and  $r.\text{softSuppSum}$  storing the soft-support-sum of ruleitems in  $\mathcal{D}$  that softly contain  $X$  and also contain  $y$ . Thus,  $\text{softSupp}(r) = \frac{r.\text{softSuppSum}}{|\mathcal{D}|}$  and  $\text{softConf}(r) = \frac{r.\text{softSuppSum}}{r.\text{anteSoftSuppSum}}$ .

Algorithm 1 is the algorithm for scars mining: (1) For 1-ruleitems  $X \subseteq I$ , we find  $F_1$  as  $F_1 = \{\{X\} \mid \text{softSupp}(X) \geq \text{minsupp}\}$ . A set  $SCAR_1$  is then generated by only choosing PRs from  $F_1$  (lines 1 - 2). (2) For each  $k$  subsequent pass, we find a set of new possibly frequent ruleitems  $CR_k$  using  $F_{k-1}$  found in the  $(k-1)^{\text{th}}$  pass. We then scan  $\mathcal{D}$ , and updates the  $\text{anteSoftSuppSum}$  and  $\text{softSuppSum}$  of ruleitems in  $CR_k$ . We then generate a new  $F_k$  by extracting ruleitems in  $CR_k$  whose soft-support  $\geq \text{minsupp}$ . A set  $SCAR_k$  is generated by only choosing PRs from  $F_k$  (lines 3 - 19). (3) From  $SCAR_1, \dots, SCAR_k$ , we choose only sets whose  $i \in [1, k] \geq \text{minitemsize}$  (a user-specified minimum ruleitem size), and store them in a set  $SCARS$ . Our idea is to choose a small representative subset of frequent ruleitems from the large number of resulting frequent ruleitems. The longer the frequent ruleitem, the more significant it is, driven from the studies [19]. We perform a rule pruning on ruleitems in  $SCARS$ . A rule  $r$  is pruned, if  $\text{Laplace}(r) < \text{min-interesting}$ . The set of ruleitems after the pruning is stored in a set  $\text{prSCARS}$  and returned (lines 20 - 22).

## 5 A Unique Retrieval Strategy: USIMSCAR

Given a new problem  $Q$ , the goal of our novel retrieval strategy USIMSCAR is to generate a retrieval result  $RR$ .  $RR$  consists of potentially useful objects that can be used to solve  $Q$  by leveraging both similarity and association knowledge.

Such objects are obtained from both stored cases and scars mined. Let  $\mathcal{D}$  be a set of cases. Let  $prSCARS$  be the set of scars mined from  $\mathcal{D}$ . We below present the USIMSCAR algorithm:

```

1:  $RC = retrieveSimilarCases(Q, \mathcal{D});$ 
2:  $RS = retrieveSimilarScars(Q, RC, prSCARS);$ 
3: for each case  $C \in RC$  do
4:    $r_C = getBestSCAR(C, prSCARS);$ 
5:   if  $r_C \neq \emptyset$  then
6:      $USF(C, Q) = SIM(C, Q) \cdot Laplace(r_C);$ 
7:   else
8:      $USF(C, Q) = SIM(C, Q) \cdot \text{min-interesting};$ 
9:   end if
10:   $object = createObject(C, USF(C, Q));$ 
11:   $RR = RR \cup object;$ 
12: end for
13: for each scar  $r \in RS$  do
14:   $USF(r, Q) = SIM(r, Q) \cdot Laplace(r);$ 
15:   $object = createObject(r, USF(r, Q));$ 
16:   $RR = RR \cup object;$ 
17: end for
18:  $RR = enhanceObjects(RR);$ 
19: Return  $RR;$ 

```

**Algorithm 2.** The USIMSCAR algorithm

- (1) In  $\mathcal{D}$ , we find the  $k$  most similar cases  $RC$  to  $Q$  (line 1). We denote  $SIM(C, Q)$  as the similarity between a case  $C \in \mathcal{D}$  and  $Q$ .
- (2) In  $prSCARS$ , we find the  $k$  most similar scars  $RS$  to  $Q$  (line 2). A question raised is how to define a function  $SIM(r, Q)$  that computes the similarity between a scar  $r$  and  $Q$ . Our answer to it lies in our choice of the “cars representation” for scars mining. Note that scars have the *identical* structure as cases—the antecedents and consequents of scars correspond to the problem and solution part of cases respectively. Thus,  $SIM(r, Q)$  can be defined in the same way as  $SIM(C, Q)$ , where  $C$  is a case in  $\mathcal{D}$ . To generate  $RS$ , we only consider the scars in  $prSCARS$  such that their itemsets in the antecedents are “soft-subsets” of cases in  $RC$ , rather than scanning all scars in  $prSCARS$  for efficiency. We denote such rules as  $RCS$ . Note that each case  $C \in RC$  is chosen as a similar case to  $Q$  ( $C \sim Q$ ). Assuming each scar  $r \in RCS$  has the form  $r : X \rightarrow y$ ,  $X$  is a soft-subset of  $C$  ( $X \subseteq_{soft} C$ ). Since  $C \sim Q$  and  $X \subseteq_{soft} C$ ,  $X \subseteq_{soft} C \sim Q$  can be derived. It implies that  $RCS$  is the collection that is a particular subset (i.e. soft-subset) of cases in  $RC$  similar to  $Q$ .
- (3) For each case  $C \in RC$ , we select a special scar  $r_C \in prSCARS$  (line 4). A rule  $r \in prSCARS$  is chosen as  $r_C$ , if it has the highest interestingness (i.e. the Laplace measure) among those scars in  $prSCARS$  such that their itemsets in the antecedents are “soft-subsets” of  $C$  and their solutions in the consequents are “equal” to the solution of  $C$ . We then compute the usefulness of  $C$  regarding  $Q$  ( $USF(C, Q)$ ) by combining  $SIM(C, Q)$  and  $Laplace(r_C)$  using the multiplication operation. If candidate(s) for  $r_C$  is chosen more than one, let

us say  $m$ , we use the average of the interestingness of these  $m$  scars to compute  $Laplace(r_C)$ . If there is no candidate for  $r_C$ , we use min-interesting for  $Laplace(r_C)$ . Note that in SBR, the usefulness of  $C$  regarding  $Q$  is generally measured by  $SIM(C, Q)$ . In contrast, our combination schemes aim to enhance such usefulness by leveraging  $SIM(C, Q)$  and  $Laplace(r_C)$ . We then cast  $C$  to a generic object  $O$  that can encapsulate any cases and scars.  $O$  has two fields:  $O.inst = C$ ;  $O.usf = USF(C, Q)$ .  $O$  is added to a retrieval result  $RR$  (lines 4 - 11);

(4) For each scar  $r \in RS$ , we compute the usefulness of  $r$  regarding  $Q$  ( $USF(r, Q)$ ) by combining  $SIM(r, Q)$  and  $Laplace(r_C)$  using the multiplication operator. This aims to directly leverage each scar in  $RS$  whose interestingness is high with respect to  $Q$ . For each scar  $r \in RS$ , we cast  $r$  to a generic object  $O$ .  $O$  has two fields:  $O.inst = r$ ;  $O.usf = USF(r, Q)$ .  $O$  is added to  $RR$  (lines 13 - 17).

(5) We further enhance the usefulness of each object in  $RR$  (line 18). This is achieved based on the *solution occurrence* of objects in  $RR$ . Our premise is that if the solution of an object  $O$  is more frequent in  $RR$ ,  $O$  is more useful in  $RR$ . The solution of each object  $O \in RR$  is differently interpreted, according to whether  $O$  was cast from a case  $C$  or a scar  $r$ . If created from  $C$ , its solution indicates the solution of  $C$ ; if created from  $r$ , its solution means the solution in the consequent of  $r$ . Let  $S$  be a set of solutions of objects in  $RR$ . Let  $S_O$  be a set of objects in  $RR$  that have the solution equal to the solution of an object  $O \in RR$ . For each object  $O \in RR$ , we compute  $\delta(S_O)$  as  $\delta(S_O) = |S_O|/|RR|$ . Finally, we enhance  $O.usf$  by multiplying  $\delta(S_O)$ . Each object  $O \in RR$ , with its usefulness regarding  $Q$  ( $O.usf$ ), will be utilized to induce a solution for  $Q$ .

We now illustrate how USIMSCAR performs with the case base  $D$  shown in Table 1, already used in Section 2, and how a solution is induced from  $RR$ . From  $D$ , assume that we generate the following scars shown in Table 2.

**Table 2.** The scars generated

Rules	<i>Laplace</i>	Soft-subset of
$r_1: \{(A_1, \text{right flank}), (A_2, \text{vomit}), (A_3, 38.6), (A_4, \text{yes}), (A_5, 13)\} \rightarrow (A_6, \text{appendicitis})$	0.922	$P_1, P_2, P_3$
$r_2: \{(A_1, \text{right flank}), (A_2, \text{vomit}), (A_3, 38.7), (A_4, \text{yes}), (A_5, 10)\} \rightarrow (A_6, \text{appendicitis})$	0.922	$P_1, P_2, P_3$
$r_3: \{(A_1, \text{right flank}), (A_2, \text{vomit}), (A_3, 38.8), (A_4, \text{yes}), (A_5, 10)\} \rightarrow (A_6, \text{appendicitis})$	0.922	$P_1, P_2, P_3$
$r_4: \{(A_1, \text{right flank}), (A_2, \text{sickness}), (A_3, 37.5), (A_4, \text{yes}), (A_5, 35)\} \rightarrow (A_6, \text{gastritis})$	0.775	$P_4$

USIMSCAR takes the following steps (assume  $k=2$  for steps (1) and (2)): (1) Retrieve the 2 most similar cases to  $Q$ :  $RC = \{P_4, P_1\}$  with  $SIM(P_4, Q) = 0.637$ ,  $SIM(P_1, Q) = 0.631$ . (2) Retrieve the 2 most similar scars to  $Q$ :  $RS = \{r_1, r_4\}$  with  $SIM(r_1, Q) = 0.640$ ,  $SIM(r_4, Q) = 0.637$ . (3) For each case  $C \in RC$ ,  $r_C$  is determined. For  $P_4$ ,  $r_4$  is selected. For  $P_1$ ,  $r_1, r_2$  and  $r_3$  are selected. Thus,  $USF(P_4, Q) = 0.494$ ,  $USF(P_1, Q) = 0.581$ . Then,  $P_4$  with  $USF(P_4, Q)$  and  $P_1$  with  $USF(P_1, Q)$  are cast to new generic objects, and stored in  $RR$ . (4) For each scar  $r \in RS$ , its usefulness regarding  $Q$  is computed:  $USF(r_1, Q) = 0.594$ ,  $USF(r_4, Q) = 0.496$ . Then,  $r_1$  with  $USF(r_1, Q)$  and  $r_4$  with  $USF(r_4, Q)$  are cast to new generic objects, and stored in  $RR$ . (5) Assume that each object in

$RR$  has a field  $s$  holding its solution.  $RR$  has four objects  $RR = \{O_1, \dots, O_4\}$  (see Table 3). As observed, there are only two sets of objects regarding solutions. For each object  $O \in RR$ ,  $O.usf$  is enhanced by weighting  $\delta(S_O) = |S_O|/|RR|$ . The enhancement results are shown under the column ‘final usefulness’ in the table. Finally, if we choose the most useful one with respect to  $Q$ ,  $O_3$  is chosen. Its solution ‘appendicitis’,  $Q$  really had, is thus used as a diagnosis for  $Q$ .

**Table 3.** The retrieval result  $RR$

field: inst	field: usf	field: solution	final usefulness
$O_1.inst = P_4$ ,	$O_1.usf = 0.494$ ,	$O_1.s =$ gastritis	0.247
$O_2.inst = P_1$ ,	$O_2.usf = 0.581$ ,	$O_2.s =$ appendicitis	0.291
$O_3.inst = r_1$ ,	$O_3.usf = 0.594$ ,	$O_3.s =$ appendicitis	<b>0.297</b>
$O_4.inst = r_4$ ,	$O_4.usf = 0.496$ ,	$O_4.s =$ gastritis	0.248

## 6 Evaluation

Our evaluation goal is to empirically show that USIMSCAR can improve SBR regarding retrieval performance. As a target application task, we choose a task highly dependent on the retrieval performance in CBR. One suitable task is to solve *classification* problems on the basis of the case-based approach. The case-based approach for classification is defined as follows [20]: given a new problem  $Q$ , its goal is to retrieve a set of similar cases to  $Q$  from a case base, and classify  $Q$  based on the retrieved cases. Thus, in principle, this approach is strongly dependent on the result obtained through retrieval in CBR. Based on this evaluation approach, we apply USIMSCAR and SBR in predicting an appropriate diagnosis for a new patient through information of patients whose diagnosis is already known with *medical* datasets found in UCI ML Repository[2].

### 6.1 Experimental Setup

SBR is typically realized through the approach using a derivative of the *nearest neighbor* algorithm [6]. This approach is called *k-nearest neighborhood retrieval* or simply *k-NN*. The idea of *k-NN* is that to solve a new problem  $Q$ , useful cases are determined using the  $k$  most similar cases to  $Q$ . For our comparison purposes, we choose the following *k-NN* based approaches available in Weka [21]: (1) *IBk* is a simple implementation of *k-NN*, and relies on the Euclidean distance to find the  $k$  most similar cases to  $Q$ ; (2) *IBkCFS* denotes an approach integrating *IBk* with an algorithm of *feature selection*, a technique for determining relevant features from the original features of cases. The algorithm chosen is *CfsSubsetEval* available in Weka. *IBk* is extended to include feature selection by only considering relevant features when finding the similar cases to  $Q$ ; (3) *IBkLVF* denotes as an approach integrating *IBk* with the feature selector *Consistency-SubsetEval* in Weka; (4) *IBkIG* denotes as an approach integrating *IBk* with an

<sup>2</sup> <http://www.ics.uci.edu/~mllearn/MLRepository.html>

algorithm of *feature weighting*, a technique for predicting optimal weights of the original features of cases. The chosen algorithm is `InfoGainAttributeEval` available in Weka. Integrating IBk with feature weighting is straightforward—features of cases (including  $Q$ ) can be weighted by feature weighting, and then such features are used to find the similar cases to  $Q$ ; and (5) *IBkCS* denotes an approach that integrates IBk with the feature weighting evaluator `ChiSquaredAttributeEval` available in Weka [21]. We also call these  $k$ -NN based approaches *classifiers*, since these will be used for classification tasks.

In the context of the  $k$ -NN classifiers, given  $Q$ , classification is done using two stages: the first is to retrieve a set of similar cases  $RR$  to  $Q$  using similarity knowledge, and the second is to classify  $Q$  using the solutions (classes) in  $RR$ . In the context of USIMSCAR, these stages can be seen as follows. The first is to retrieve a set of *useful cases and rules*  $RR$  regarding  $Q$  using similarity and association knowledge, and the second is to classify  $Q$  using information driven in  $RR$ . Our work is focused on the first stage. The second stage is often covered by *voting* [14]. We choose two well-known voting schemes to perform this stage: (1) *majority voting*—the majority solution in  $RR$  is chosen as a solution for  $Q$ , and (2) *distance weighted voting*—each object  $O \in RR$  gets to vote on the solution of  $Q$  with a vote weighted by its similarity to  $Q$  (in the context of the  $k$ -NN classifiers) or usefulness regarding  $Q$  (in the context of USIMSCAR).

Table 4 provides a summary of the medical datasets used in our experiments.

**Table 4.** The medical datasets used in the experiments

Dataset	No of Cases	No of Attributes	Attr Type		No of Classes
			Numeric	Nominal	
Breast Cancer (BC)	286	9		9	2
Breast Cancer Wins (BCW)	683	10	10		2
Breast Tissue (BT)	106	9	9		6
Pima Indian Diabetes (PID)	768	9	9		2
StatLog Heart Disease (SHD)	270	13	7	6	2
New Thyroid (THY)	215	5	5		3

For our evaluation criteria, we use two metrics widely used for evaluating classifiers. The first is the *classification accuracy* that has been very often assumed to be the best performance indicator for evaluating classifiers. It measures the proportion of correctly classified instances out of all the tested instances. However, this accuracy does not take into account the *cost of making wrong decisions*. To supplement this lack, we choose *F-measure*. F-measure is defined as the harmonic mean between *precision* (P) and *recall* (R), denoted as  $F\text{-measure} = \frac{2PR}{P+R}$ . P represents the proportion of the instances, which truly have a class  $X$ , among all those classified as a class  $X$ . R indicates the proportion of the instances, classified as a class  $X$ , among all those instance having a class  $X$ . A high F-measure value indicates that both P and R are reasonably high.

USIMSCAR and the five  $k$ -NN based classifiers compared (simply 5CF) are all tested with six medical datasets by using *10-fold cross-validation*. In this validation practice, each dataset is partitioned into ten subsets. Of the ten subsets,

a single subset is retained as *testing data*, and the remaining nine subsets are used as *training data*. The cross-validation process is then repeated ten times (the folds), with each of the ten subsets used exactly once as the testing data.

The similarity knowledge, used in the experiments, is encoded as a similarity measure using the global-local principle explained in Section 3.1. Given a new problem  $Q$  and a case  $C$ , their similarity is defined as the function  $SIM$  in Eq. 1. We configured that this function is equally used in 5CF and USIMSCAR. To perform Algorithm 1, we use the following parameters:  $minsupp = 0.1$  (10%);  $minsim = 0.95$  (95%);  $min-interesting = 0.7$  (70%); and  $minitemsize = 0.5 * N$ , where  $N$  is the total number of the attributes of instances in the training data.

## 6.2 Results and Analysis

For each dataset, we first report the mean number of scars generated by performing Algorithm 1, where the mean is attained by applying 10-fold cross-validation: BC:228, BCW: 1513, BT: 6013, PID: 524, SHD: 676, and THY: 1073. We found that the number of the scars generated increases with the increase in the number of the attributes of instances for each dataset. However, interestingly, the number of the scars acquired from BT is the highest as 6010, although the number of instances in the training data is the lowest as 96 ( $106 * 0.9$ ). This was occurred, since some items in BT relatively appear very frequently.

To test approaches (USIMSCAR and 5CF), we need to find a best value for the top  $k$  that indicates the number of the most similar cases to a new problem  $Q$ . We test 5CF using various values for the  $k$ , ranging from 1 to 15, since we observed that increasing  $k$  beyond 15 hardly changed the results. To run USIMSCAR, we also need to find a value for the top  $k$  that indicates the number of the most similar cases and scars to  $Q$ . We tested USIMSCAR using the same value range for finding it. Our comparison purposes, we finally use the best result obtained from the use of the best choice of the  $k$  in terms of classification accuracy (CA) and F-measure (FM).

**Results Using Majority Voting.** We first present the results of USIMSCAR and 5CF using majority voting in terms of CA. Table 5 shows a summary of the results. The best approach is denoted in boldface for each dataset.

As observed in Table 5, USIMSCAR performs best for three dataset (BCW, BT, and THY). For BCW, its improvement over 5CF ranges from 0.44% to 0.73%. For BT, the improvement is up to 6.60%. For THY, the improvement is consistently equal to 1.4%. Whereas USIMSCAR occupies 2nd place for the datasets (BC, PID, and SHD). Thus, it outperforms the other four classifiers for the datasets. For BC, PID, and SHD, it performs better than those classifiers with a range of 1.75% - 2.45%, 0.39% - 1.43%, and 0.74% - 1.48%, respectively.

Table 6 shows a summary of the results of USIMSCAR and 5CF in terms of FM. The best one is also denoted in boldface for each dataset. As can be seen, USIMSCAR outperforms 5CF with three (BCW, BT, and THY) out of the six datasets. For BC, and BCW, its improvement ranges from 0.09% to 4.9%, and 0.47% to 0.78%, respectively. For THY, its improvement is consistently equal

**Table 5.** The best results using majority voting in terms of classification accuracy (%)

Dataset	USIMSCAR	IBk	IBkCFS	IBkLVF	IBkIG	IBkCS
BC	75.874	74.126	<b>76.224</b>	73.776	73.427	73.427
BCW	<b>97.657</b>	97.218	97.218	97.218	96.925	97.218
BT	<b>71.698</b>	<b>71.698</b>	65.094	67.925	69.811	70.755
PID	75.781	74.349	<b>77.214</b>	75.000	74.870	75.391
SHD	83.333	82.593	81.852	82.222	<b>85.185</b>	81.852
THY	<b>97.674</b>	96.279	96.279	96.279	96.279	96.279

**Table 6.** The best results using majority voting in terms of F-measure (%)

Dataset	USIMSCAR	IBk	IBkCFS	IBkLVF	IBkIG	IBkCS
BC	<b>68.744</b>	65.147	68.653	64.896	63.840	65.147
BCW	<b>97.419</b>	96.946	96.946	96.946	96.643	96.946
BT	71.178	<b>71.465</b>	63.249	65.127	67.689	68.435
PID	72.212	70.751	<b>74.137</b>	71.923	71.463	71.923
SHD	83.078	82.339	81.561	82.000	<b>84.966</b>	82.725
THY	<b>96.883</b>	95.084	95.084	95.084	95.084	95.084

to 1.80%. USIMSCAR occupies 2nd place for the other three datasets (BT, PID, and SHD). Thus, it performs better than the other four classifiers for the datasets. For BT, PID, and SHD, it outperforms these classifiers with a range of 2.74% - 7.93%, 0.29% - 1.46%, and 0.35% - 1.52%, respectively.

Through Tables 5 and 6, we find that USIMSCAR outperforms 5CF for the six datasets in 27 out of 30 comparisons in terms of both CA and FM. It indicates that using majority voting USIMSCAR achieves better performance in 90% of the cases than 5CF in terms of the metrics.

**Results Using Weighted Voting.** We now analyze the experimental results of USIMSCAR and 5CF using weighted voting in terms of CA. Table 7 shows a summary of the results. The best one is denoted in boldface for each dataset. As can be observed, USIMSCAR outperforms 5CF for all the six datasets. The improvement of USIMSCAR over them for each dataset is as follows. For BC, BCT, BT, PID, SHD, and THY, it is 4.90% - 6.64%, 0.29% - 0.58%, 6.64% - 13.20%, 10.42% - 13.15%, 5.93% - 7.78%, and 1.40%, respectively.

Table 8 shows a summary of the results of USIMSCAR and 5CF in terms of FM. The best one is also denoted in boldface for each dataset. As observed, USIMSCAR outperforms 5CF for all the six datasets. The improvement of USIMSCAR over them for each dataset is as follows. For BC, BCT, BT, PID, SHD, and THY, it is 10.19% - 12.12%, 0.32% - 0.62%, 6.16% - 14.16%, 13.83% - 15.15%, 6.10% - 7.83%, and 1.80%, respectively.

Through Tables 7 and 8, we find that USIMSCAR outperforms 5CF for the six datasets in all 30 comparisons in both CA and FM. This means that USIMSCAR achieves better performance in 100% of the cases than the classifiers in terms of both metrics.



**Table 7.** The best results using weighted voting in terms of classification accuracy (%)

Dataset	USIMSCAR	IBk	IBkCFS	IBkLVF	IBkIG	IBkCS
BC	<b>79.021</b>	73.427 •	73.776 •	72.727 •	72.378 •	74.126 •
BCW	<b>97.657</b>	97.218	97.218	97.365	97.072	97.218
BT	<b>78.302</b>	71.698 •	65.094 •	67.925 •	69.811 •	71.698 •
PID	<b>87.500</b>	74.479 •	77.083 •	75.391 •	74.479 •	74.349 •
SHD	<b>89.630</b>	82.222 •	82.222 •	81.852 •	83.704 •	82.593 •
THY	<b>97.674</b>	96.279	96.279	96.279	96.279	96.279

**Table 8.** The best results using weighted voting in terms of F-measure (%)

Dataset	USIMSCAR	IBk	IBkCFS	IBkLVF	IBkIG	IBkCS
BC	<b>74.251</b>	64.053 •	65.245 •	62.517 •	62.127 •	64.053 •
BCW	<b>97.421</b>	96.946	96.946	97.104	96.798	96.946
BT	<b>77.626</b>	71.465 •	63.466 •	65.127 •	67.689 •	68.435 •
PID	<b>86.140</b>	71.177 •	74.055 •	72.307 •	70.995 •	72.307 •
SHD	<b>89.553</b>	82.034 •	81.942 •	81.723 •	83.451 •	81.723 •
THY	<b>96.883</b>	95.084	95.084	95.084	95.084	95.084

Using the results shown in Tables 5 - 8, to find whether the performance improvement of USIMSCAR in terms of CA and FM is statistically significant over 5CF, we carried out statistical tests. A common approach for measuring a significant test for a difference between two *proportions* is the *Z-test* [22]. We performed statistical tests using the *Z-test* at 90% confidence. In the tables, ‘•’ indicates that USIMSCAR attains a significant improvement over the target classifier at 90% confidence. As observed in Tables 5 and 6, using majority voting, there is no statistical difference between USIMSCAR and 5CF in terms of both CA and FM. However, as seen in Tables 7 and 8, 67% of comparisons (20 out of 30 comparisons) between USIMSCAR and 5CF are statistically significant in terms of both CA and FM. We note that insignificant improvement of USIMSCAR over 5CF does not mean that there is no differences between them, merely that the tests were unable to detect significance differences. As Keen [23] indicated, such improvement still can be important if it occurs repeatedly in many contexts. Thus, the insignificant improvement of USIMSCAR may be still valuable, and this is where a wide range of tests will be additionally carried out.

We emphasize that for all the six datasets, USIMSCAR using weighting voting ( $U_{WV}$ ) outperforms USIMSCAR using majority voting ( $U_{MV}$ ) in terms of both CA and FM. As observed in Table 9,  $U_{WV}$  attains better performance than  $U_{MV}$  in the 4.63% of the occasions in terms of CA on average. It attains better performance than  $U_{MV}$  in the 5.34% of occasions in terms of FM on average. This findings indicate that it is more useful to utilize the quantified “usefulness information” of objects in the retrieval result *RR* of USIMSCAR, rather than utilizing merely distribution information of the solutions in *RR*. Recall that  $U_{WV}$  is configured to perform using the usefulness of objects in *RR*, which is quantified using both similarity and association knowledge. Whereas  $U_{MV}$  is configured to perform using distribution information of solutions in *RR*.

**Table 9.** USIMSCAR with two voting schemes

(a) In classification accuracy				(b) In F-measure			
Dataset	$U_{MV}$	$U_{WV}$	Diff	Dataset	$U_{MV}$	$U_{WV}$	Diff
BC	75.874%	79.021%	3.147%	BC	68.744%	74.251%	5.507%
BCW	97.657%	97.657%	0%	BCW	97.419%	97.421%	0.002%
BT	71.698%	78.302%	6.604%	BT	71.178%	77.626%	6.448%
PID	75.781%	87.500%	11.719%	PID	72.212%	86.140%	13.928%
SHD	83.333%	89.630%	6.297%	SHD	83.078%	89.553%	6.475%
THY	97.674%	97.674%	0%	THY	96.883%	96.883%	0%
Mean	83.670%	88.297%	4.628%	Mean	81.586%	86.979%	5.393%

## 7 Related Work

SBR has been successfully applied in various application domains, such as medical diagnosis [2] and help-desk service [3], to predict useful cases with respect to solving the target problem. As mentioned in Section 6, SBR is typically implemented through  $k$ -NN.

Over the years, researchers have extensively studied  $k$ -NN to enhance its accuracy. For example, it is shown that  $k$ -NN can be integrated with feature selection [24] or feature weighting [25]. As other trends, to enhance SBR, much work focuses on integrating SBR with *machine learning*, *domain knowledge*, and *adaptation knowledge*. The evolution of machine learning has resulted in retrieval approaches that combine SBR with rule-induction (RI) methods to improve SBR [26,9]. RI systems learn domain-specific knowledge, from stored cases, often represented as IT-THEN rules. Once this knowledge is available, SBR is augmented with rule-based reasoning using this knowledge. Several researchers propose a retrieval approach, in which similarity assessment during SBR is integrated with domain knowledge [8]. Aamodt [11] proposes an approach that cases are enriched with domain knowledge. Domain knowledge often represents the knowledge about the environment in which the target system operates, e.g. facts, heuristics. These approaches aim to guide the retrieval of relevant cases using domain knowledge. Some work tries to enhance SBR using adaptation knowledge. For example, the *adaptation-guided retrieval* (AGR) approach is proposed [7], in which adaptation knowledge indicates whether a case can be easily modified to fit the new problem. In AGR, matches between the target problem and cases are done, only if there is enough evidence existed in adaptation knowledge that such matches can be catered for during retrieval. USIMSCAR differs from the above approaches in three aspects:

- *Knowledge acquisition*: The acquisition of both domain knowledge and adaptation knowledge is usually known as a very complex and difficult task [8], thus often leads to *knowledge bottleneck phenomenon*. The acquisition is also very often done with the support of domain experts [7,8]. However, the acquisition of association knowledge (AK) is straightforward, since AK is acquired from stored cases. This is also efficient, since acquisition is automatically done using association rule mining.

- *Knowledge formalization*: AK is formalized by capturing interesting associations, between known problem features and known solutions, shared by a large number of cases. In this context, this formalization can be compared to feature selection and feature weighting, since they focus on estimating the relevancy of certain problem features highly correlated to specific known solutions from the CBR viewpoint. However, they are often based on finding only relationships between single individual features and each solution, ignoring interesting relationships between *combinations* of features and each solution. The latter relationships may be curial in a certain circumstance. For example, two individual features may be strongly related to a certain solution but together may not, or vice versa. In contrast, AK can represent both kinds of relationships, thereby providing enriched relationships between the features of problems and solutions. This benefit originally comes from the use of association rule mining, which enables to extract all interesting correlations and frequent patterns derived in a case base.
- *Knowledge exploitation*: The uniqueness of USIMSCAR lies in the use of AK in conjunction with similarity knowledge (SK). This exploitation can be compared to the usage of the rules generated by RI methods from an analysis of cases. We note that this usage is classified into two schemes: one for weighting features [9], and the other for generating a solution for a given problem without using knowledge derived from similarity measurement between a given problem and cases (i.e. SK) [26]. In contrast, we leverage combined information inherent in both SK and AK for the retrieval process in CBR.

## 8 Conclusion and Future Work

In this paper, we proposed a new retrieval strategy USIMSCAR that outperforms similarity-based retrieval (SBR). Its uniqueness lies in leveraging association knowledge in conjunction with similarity knowledge during CBR retrieval. Also, we proposed a unique approach for extracting and representing association knowledge using association rule mining techniques. We evaluated USIMSCAR in comparison with SBR using medical datasets found in UCI ML Repository. The experimental results demonstrated that USIMSCAR is an effective retrieval strategy for CBR that qualitatively outperforms SBR.

USIMSCAR can be extended to cases, where each problem is represented by complex structures. In CBR, case problems can be characterized by not only attribute-value pairs, but also more complex structures like object-oriented (OO) or hierarchical (HR) representation [6]. The OO representation utilizes the data modeling approach of OO paradigm such as “is-a”. In the HR representation, a case problem is characterized through multiple levels of abstraction, and its attribute values reference nonatomic cases. For USIMSCAR to treat the case problems characterized by such representations, two issues must be addressed—how to formalize similarity knowledge and association knowledge. To address the former, one may use the similarity approaches proposed in [27]. To address the latter,

one may integrate the soft-matching criterion and extended Apriori algorithms [28,29] for association rule mining in OO data and HR data. USIMSCAR can also be extended to cases, where each case problem is associated with more than one solution. This can be simply generalized into the occasion—a case problem is associated with one solution. The generalization is possibly done by splitting a case  $C$  into  $k$  number of sub cases, according to its  $k$  number of solutions. All these cases are forced to have the same case ‘id’ of  $C$ . For example, there is a case  $C = (X, Y)$  (id =  $C$ ), where  $X$  is a problem and  $Y$  is a corresponding solution. Assume that  $X$  consists of two treatments {2-tylenol, 2-aspirin}. For  $C$  to be used in USIMSCAR, we split  $C$  into two sub cases, whose ids are the same of  $C$ . By doing so, we obtain two cases:  $C = (X, 2-tylenol)$  and  $C = (X, 2-aspirin)$ . Then, USIMSCAR can run for these cases without modification.

## References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7, 39–59 (1994)
2. Ahn, H., Kim, K.-j.: Global optimization of case-based reasoning for breast cytology diagnosis. *Expert Syst. Appl.* 36, 724–734 (2009)
3. Kang, Y.-B., Zaslavsky, A., Krishnaswamy, S., Bartolini, C.: A knowledge-rich similarity measure for improving it incident resolution process. In: *Proceedings of the 2010 ACM SAC*, pp. 1781–1788. ACM (2010)
4. Bradley, K., Smyth, B.: Personalized information ordering: a case study in online recruitment. *Knowledge-Based Systems* 16, 269–275 (2003)
5. Nilsson, M., Funk, P., Sollenborn, M.: Complex Measurement Classification in Medical Applications Using a Case-Based Approach. In: Ashley, K.D., Bridge, D.G. (eds.) *ICCBR 2003*. LNCS, vol. 2689, pp. 63–73. Springer, Heidelberg (2003)
6. De Mantaras, R.L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M., Aamodt, A., Watson, I.: Retrieval, reuse, revision and retention in case-based reasoning. *Knowl. Eng. Rev.* 20, 215–240 (2005)
7. Smyth, B., Keane, M.T.: Adaptation-guided retrieval: questioning the similarity assumption in reasoning. *Artif. Intell.* 102, 249–293 (1998)
8. Stahl, A.: Learning of knowledge-intensive similarity measures in case-based reasoning. PhD thesis. Technical University of Kaiserslautern (2003)
9. Cercone, N., An, A., Chan, C.: Rule-induction and case-based reasoning: hybrid architectures appear advantageous. *IEEE Trans. on Know. and Data Eng.* 11, 166–174 (1999)
10. Park, Y.J., Kim, B.C., Chun, S.H.: New knowledge extraction technique using probability for case-based reasoning: application to medical diagnosis. *Expert Systems* 23, 2–20 (2006)
11. Aamodt, A.: Knowledge-intensive case-based reasoning in CREEK. In: Funk, P., González Calero, P.A. (eds.) *ECCBR 2004*. LNCS (LNAI), vol. 3155, pp. 1–15. Springer, Heidelberg (2004)
12. Hoffmann, A., Khan, A.S.: A new approach for the incremental development of retrieval functions for CBR. *Applied Artificial Intelligence* 20, 507–542 (2006)
13. Castro, J.L., Navarro, M., Sánchez, J.M., Zurita, J.M.: Loss and gain functions for CBR retrieval. *Inf. Sci.* 179, 1738–1750 (2009)

14. Cunningham, P.: A Taxonomy of Similarity Mechanisms for Case-Based Reasoning. *IEEE Trans. on Knowl. and Data Eng.* 21, 1532–1543 (2009)
15. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: *SIGMOD 1993*, pp. 207–216. ACM (1993)
16. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: *Proceedings of the 4th KDD*, pp. 443–447 (1998)
17. Nahm, U.Y., Mooney, R.J.: Mining soft-matching association rules. In: *Proceedings of CIKM 2002*, pp. 681–683 (2002)
18. Geng, L., Hamilton, H.J.: Interestingness measures for data mining: A survey. *ACM Comput. Surv.* 38, 9 (2006)
19. Hu, T., Sung, S.Y., Xiong, H., Fu, Q.: Discovery of maximum length frequent itemsets. *Inf. Sci.* 178, 69–87 (2008)
20. Jurisica, I., Glasgow, J.: Case-Based Classification Using Similarity-Based Retrieval. In: *International Conference on Tools with Artificial Intelligence*, p. 410 (1996)
21. Witten, I.H., Frank, E.: *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, San Francisco (2000)
22. Richard, C.S.: *Basic Statistical Analysis*. Allyn & Bacon (2003)
23. Keen, E.M.: Presenting results of experimental retrieval comparisons. *Inf. Process. Manage.* 28, 491–502 (1992)
24. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. *J. Mach. Learn. Res.* 3, 1157–1182 (2003)
25. Park, J.H., Im, K.H., Shin, C.K., Park, S.C.: MBNR: Case-Based Reasoning with Local Feature Weighting by Neural Network: Special Issue: Soft Computing in Case Based Reasoning. *Applied Intelligence* 21, 265–276 (2004)
26. Auriol, E., Wess, S., Manago, M., Althoff, K.-D., Traphöner, R.: INRECA: A Seamlessly Integrated System Based on Inductive Inference and Case-Based Reasoning. In: Aamodt, A., Veloso, M.M. (eds.) *ICCBR 1995*. LNCS, vol. 1010, pp. 371–380. Springer, Heidelberg (1995)
27. Bergmann, R., Stahl, A.: Similarity measures for object-oriented case representations. In: Smyth, B., Cunningham, P. (eds.) *EWCBR 1998*. LNCS (LNAI), vol. 1488, pp. 25–36. Springer, Heidelberg (1998)
28. Kuba, P., Popelinsky, L.: Mining frequent patterns in object-oriented data. *Technical Report*, Masaryk University, Czech Republic (2005)
29. Pater, S.M., Popescu, D.E.: Market-Basket Problem Solved With Depth First Multi-Level Apriori Mining Algorithm. In: *SOFA 2009*. 3rd International Workshop on Soft Computing Applications, pp. 133–138 (2009)

# FlexCon – Robust Context Handling in Human-Oriented Pervasive Flows

Hannes Wolf, Klaus Herrmann, and Kurt Rothermel

Institute of Parallel and Distributed Systems,  
Universitätsstraße 38, D-70569 Stuttgart, Germany  
{hannes.wolf,klaus.herrmann,kurt.rothermel}@ipvs.uni-stuttgart.de

**Abstract.** Workflows are increasingly becoming a universal means for driving and coordinating complex processes, not only in the business world but also in areas like pervasive computing. *Pervasive flows* run in parallel with the user’s real-world actions and are synchronized using automatically collected context information about her current activities (context events). Respective workflows cannot be rigidly defined since the user needs to retain her flexibility and must not be obstructed by the workflow. However, if the order of activities is not defined until the activities are actually executed, correctly assigning the uncertain context events becomes a major challenge. We propose FlexCon – a novel event assignment approach for such human-oriented workflows that is based on hybrid workflow models and Dynamic Bayesian Networks. FlexCon exploits the dependency between context events to provide more accurate information as to which events need to be consumed by which workflow activities. Our evaluations show that FlexCon improves the event accuracy on average by 54% and the number of successful completed flows on average by 88%. Thus, FlexCon represents a major step towards unobtrusive pervasive applications.

## 1 Introduction

Workflows are an adequate means for modeling the functionality and the temporal flow of complex activities in many areas of information technology. Traditionally, they are rooted in business applications [1], where processes span diverse departments of the same company or different companies. In recent years, however, an influx of workflow technologies into the domain of pervasive systems has started. They have been proposed as useful tool for environments with intensive human interaction [2]. Gradually, workflows have become more flexible, supporting [3] and also gained context-awareness [4].

In the ALLOW project [5], we investigated the concept of *Adaptable Pervasive Flows* (in short *flows*) as a means for rendering complex applications and the environment (technical equipment, information systems etc.) adaptive to the mobile user. The basic idea of flows is that a process that a user has to execute is modeled as a flow. This flow is running in the background in parallel to the actual real-world actions of the user and can support her in different ways.

The key to this is that the flow requires as little explicit input from the user as possible in order to offer unobtrusive support. First, the flow can guide the user through the process by giving her feedback in critical situations (e.g. visual or audio). Second, it may take over certain routine tasks automatically (e.g. documenting the actual process for legal or quality-related reasons). Third, the flow may prepare the environment (e.g. configuring electronic devices etc.) in advance to minimize disturbances and the work load of the user. Fourth, the flow may automatically adapt itself in case it recognizes that the planned execution will fail (e.g. due to lack of a required resource). All of this is possible since the flow “knows” the prospective future activities associated with the process – they are part of the flow model.

To achieve this with minimal explicit input, the flow system monitors what the user does in the real world in order to automatically synchronize with her actions and drive the flow forward respectively. This monitoring, which provides the majority of the input to the flow, is done by means of *activity recognition* [6,7] using different types of sensors. The respective data is provided to the flow as so-called *context events*.

As an example, consider a nurse in a hospital that has a flow modeling the morning routine of a patient. That flow involves different activities like **washing**, **dressing**, **measuring blood pressure**, and **disinfecting hands**. The actions associated with these activities in the real world are detected by the activity recognition system and provided to the flow as context events. Some of the activities have a clear ordering relation, e.g. she has to wash the patient *before* she dresses him. Others are mandatory but may occur at different stages of the flow (e.g. **measuring blood pressure**). Yet other activities like **disinfecting hands** may be carried out multiple times as needed. Thus, a respective flow must not be defined rigidly with a fixed predefined order between activities. It must allow the nurse the flexibility of executing the process in her own way and in the way required by the specific situation.

When the flow arrives at a certain activity, there may be different ways of continuing (different possible next activities). Depending on the context events received, the flow has to decide which of these paths the user took. Supplying the correct context events to the right flow activities is a tough challenge under these conditions. First, events may be noisy, duplicated, deleted due to failure or delayed. This is a fundamental problem that occurs irrespectively of the flexibility in the flow models [8]. Second, due to the gained flexibility, it may not be completely clear which activity is actually executed next and thus awaits a respective event.

To solve this fundamental problem of pervasive flow technology in general, we propose FlexCon, a system that leverages a combination of imperative (rigid) and declarative (flexible) flow models, Dynamic Bayesian Networks (DBN), and particle filters to reduce the uncertainty of context events. FlexCon builds probabilistic models of the dependencies between events and uses this information to improve the processing of probabilistic context events. We evaluated FlexCon based on a real-world hospital study and found that it decreases the context

event uncertainty by up to 73%. While standard flow technology exhibits a high flow failure rate<sup>1</sup> of 82% under the conditions explained above, the failure rate of FlexCon is only 65% on average of all flows. This presents a major step forward in the areas of workflow-based pervasive computing.

The rest of the paper is structured as follows: In Section 2 we will investigate the related work in relevant areas. We present the basic models of context and workflows in Section 4. After that we introduce our FlexCon approach in Section 5 and evaluate it based on a real-world scenario in Section 6. Finally, we present our conclusions in Section 7.

## 2 Related Work

In the following, we will investigate the state-of-the-art in the relevant areas of research. We will first discuss activity recognition systems and how they deal with context uncertainties. While our work is not directly associated with this area, it does provide a new approach for handling the uncertainties perceived in activity recognition systems on a higher layer by exploiting application knowledge. Subsequently, we will take a closer look at the field of context-aware workflows.

There have been numerous studies on activity recognition in the health-care domain [9,10,11]. The major factors for decreasing the uncertainty in the recognition results are the selection of appropriate sensors and exploiting available application models. Biswas et al. [11] specifically remark that the recognition process can benefit from the knowledge of domain experts. A flow is a very detailed representation of expert application knowledge, that FlexCon uses to increase the accuracy of events.

Barger et al. [9] studied a health status monitoring application that learns behavioral patterns of a user from his daily activities using a number of motion sensors. But their system lacks an application model too, leading to missed events and false positives and a rather low recognition accuracy for uncommon situations.

Najafi et al. [10] have built a monitoring system for elderly people using one acceleration sensor, and detecting position transitions and mode of locomotion. While this approach performs very well for single transitions in a specific test scenario, the sensing quality decreases over extended periods of time due to the lack of an application model.

The presented approaches all use sophisticated activity recognition techniques, but do not consider the kind of application knowledge a flow provides, thus, neglecting the huge potential.

The integration of context information into classic workflows used in enterprises has first been suggested by Wieland et al. [4], who provide interfaces for accessing context information from within a workflow. This approach was later extended to deal with Quality of Context [12]. Here, a policy language is used to define the acceptable amount of uncertainty in context information and to

---

<sup>1</sup> A flow fails if context events are assigned to false activities or are discarded due to false recognition results.



filter out information that does not match the specified criteria. This approach is based on the idea to simply prevent the workflow from receiving uncertain information. However, if a workflow does not receive the information at all, this can be just as detrimental as receiving false information. We go one important step further by improving the information such that it becomes useful for the flow.

Adam et al. [13] proposed fuzzy logic to enable *soft decisions* in workflows based on the input provided to the workflow. However, they did not consider uncertainties or ambiguities in the input information. Their approach aims at making a better decisions from the business perspective.

PerFlows, presented by Urbanski et al. [14], are context-aware workflows that are suitable for pervasive scenarios and provide flexible activity scheduling and processing. However, they require heavy user interaction to work properly. This is disadvantageous in scenarios where the workflows shall run in the background in order not to obstruct the user. In our own previous work [15], we presented an approach for dynamic context-awareness suited for pervasive flow-based applications. But this approach also neglects the handling of uncertain context information.

In 2010, we have proposed FlowCon [15], the first system that was able to decrease the uncertainty of events by learning the dependencies between events and by explicitly exploiting the temporal structure encoded in imperative flows (flows with a strict temporal ordering among activities). FlowCon can increase the number of successfully executed flows by a factor of 6 to 8 under normal conditions. With the FlexCon system presented in this paper, we build on this work and apply related technologies to hybrid flow models that are more flexible and allow users to act more freely. Overall, this represents a significant step forward in this field.

### 3 Scenario

The application scenario we use to evaluate FlexCon is from the health care domain. We studied the processes conducted by nurses in a geriatric ward in Mainkofen, Germany over a period of 14 days.

Through a mining process, we extracted workflows from the observations made at the ward. The respective processes have not been defined as workflows before. However, in this highly structured working environment, workflows are implicitly followed in order to fulfill a number of standards in terms of patient care. In total we collected 32 datasets from 15 different nurses, where each dataset covers the care of 3 to 5 patients, yielding a total of 130 observed workflow executions.

The purpose of applying a system of adaptable pervasive flows in this institution is twofold: First, the activities shall be automatically documented for the records for quality control, process improvement, and legal reasons. Second, the flow system shall give guidance in case the standard procedures are not followed in order to avoid mistakes and help inexperienced personnel in learning the procedures.

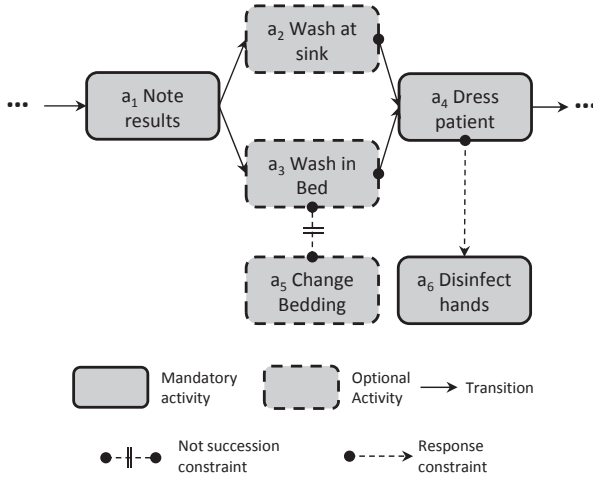


Fig. 1. Example workflow from a hospital scenario

A typical workflow, e.g. from the morning routine, consists of 30 to 50 activities of which about 20% have no strict order. The entire navigation in such a flow depends on context events (i.e. the correct next activity is chosen based on the context events received). An example fragment is depicted in Figure 1. Solid boxes depict mandatory activities that need to be executed unconditionally while dashed boxes are optional. For example,  $a_2$  and  $a_3$  are optional activities. The execution of a flow instance is valid if, one of the optional activities, both or non of them have been executed, while  $a_1$ ,  $a_4$ , and  $a_6$  need to be executed for the flow to be successful. Solid arrows are transitions that imply a strict ordering between the activities:  $a_1$  must be followed by either  $a_2$  or  $a_3$  and  $a_4$  must follow both  $a_1$  and  $a_2$ . The dashed lines are constraints that define certain restrictions on the execution order of the related activities. The figure depicts two examples: the semantics of the not succession constraint between  $a_3$  and  $a_5$  is that a valid flow execution must not contain both activities. It may contain either one or none of them, and if one is executed, it can be executed arbitrarily often. As we will explain in Section 4, constraints can be arbitrary linear temporal logic expressions. Some of them have been translated into a graphical representation.

The flow shown in Figure 1 is a fragment of a larger flow that models the actual processes found in the Mainkofen nursing ward. Its overall semantics is the following: When a nurse arrives at this fragment, she must document the results ( $a_1$ ) of the preceding steps, which include some regular morning examinations, such as measuring blood pressure. As these examinations are carried out without assistance of an electronic device, the flow ensures that the nurse will not forget the results during the following steps. Then she has to take a decision: she may wash the patient at the sink ( $a_2$ ) or in his bed ( $a_3$ ), depending on the patients condition and mood. In FlexCon actually both activities are entered as soon as

$a_1$  has completed. Depending on the incoming context events, either one or both are executed. If the nurse decides to wash the patient in his bed ( $a_3$ ), she cannot change the bedding ( $a_5$ ) since the patient still never leaves his bed during the whole procedure (this is done in a different flow). After the nurse has completed the washing activity, she needs to dress the patient ( $a_4$ ). When she dressed him, she must disinfect her hands ( $a_6$ ) at some later point in time, possibly after a number of other intermediate activities. But, she may disinfect her hands at any point in time, while the flow is being executed. This is beneficial in two ways. First, the nurse can flexibly decide to disinfect her hands multiple times, e.g. during washing the patient, also allowing the system to keep track of her personal hygiene as well as the patients. Second, the flow can guide the nurse to disinfect her hands before she continues to care for another patient, this way enforcing the hospitals hygiene rules.

## 4 Flow and Context Models

In this section, we will first define our model of context information before we give a definition of hybrid flows.

### 4.1 Context Model

As the flows should not obstruct the nurse in her daily routine, they are solely driven by context events. Therefore, adequate sensors and an activity recognition and context management system (CMS), must be available to gather context information and provide the context events. However, state of the art activity recognition systems have some drawbacks. Either, they require the precise deployment of (expensive) sensors, or the setup and training of the system is tedious. Cheaper activity recognition systems, e.g. based on standard smart phones, only provide moderate recognition rates, at best. While the former technology might be applicable in high-cost environments such as an operating room, we have to rely on the latter kind in the area of cost-sensitive every-day patient care.

In the scope of our scenario, we assume that in practice the type of different events FlexCon is interested in, is a finite set.

**Definition 1 (Event Type).** *A type of situation that can be recognized in the real world is referred to as an event type  $u \in U$ , where  $U$  denotes the universe of all event types that the CMS can measure.*

An event type describes the abstract semantics of an context event. For example, *nurse walking* could be an event type. Events of this type are created whenever a nurse changes her mode of locomotion to *walking*. Event types that represent semantically similar context can belong to a common *event type set*, and each event type belongs to at least one event type set.

**Definition 2 (Event Type Set).** *An event type set  $E \subset U$  contains a number of event types  $E := \{u_1, \dots, u_m\}, m > 0$ . A single event type can be a member of different event type sets.*

The event type set containing all event types for a nurse’s locomotion modes could be, e.g.  $\{\text{nurse walking}, \text{nurse sitting}, \text{nurse standing}\}$ . The purpose of an event type set is twofold: First, it allows the flow modeler to simply select the most appropriate context the activity should respond to. As seen below, a flow model defines a function that maps every activity to a number of distinct event type sets. Second, the related semantics of all event types in an event type set allows for a more accurate recognition: Event types that are not contained in one of the expected event type sets of the current flow activity are likely to be out of scope. When executed the flow registers the event type sets of a running activity at the CMS and receives *event instances*.

**Definition 3 (Event Instance).** *An event instance  $e \in U_e$  is an instance of a specific event type  $u \in U$ .  $U_e$  is the universe of all event instances occurring in the system.  $e$  belongs to a specific event type  $u \in E$ , and the uncertainty about which exact event type in  $E$   $e$  belongs to is given by a probability distribution  $I_E^e : E \mapsto [0, 1]$ , where  $\sum_{u \in E} I_E^e(u) = 1$ .*

$I_E^e$  is our basic model of uncertainty. Instead of saying that an event instance is of type  $u$ , the CMS provides the distribution  $I_E^e$ , and  $I_E^e(u)$  is the probability that  $e$  is of type  $u \in E$ . For example if  $u = \text{nursewalking}$  and  $u \in E$  then  $I_E^e(\text{nursewalking}) = 0.52$  indicates that the probability of  $e$  being of type *nurse walking* is 52%.

## 4.2 Hybrid Flow Model

A *flow model* is a template for a specific type of flow. A runnable instance of such a model must be created whenever a flow is to be executed. We call this a *flow instance*. In the following, we also refer to such an instance simply as a *flow*. The flow instance is executed by a flow engine. In our work we employ *hybrid flow models* that contain *transitions* as well as *constraints* between activities and, thus, are a mixture of classical imperative production workflows [1] and declarative flexible [16] workflows. Transitions are annotated with boolean *conditions* over the possible set of context events while *constraints* consist of *linear temporal logic* expressions that describe the acceptable temporal relation of two or more tasks (e.g.  $a$  must be executed before  $b$ ). If a flow modeler currently wants to use a mixture of both modeling paradigms he is required to add this flexibility in a hierarchical way [17]. He must decompose the application into a number of hierarchical layers, usually representing a different level of abstraction and choose the best modeling paradigm for each layer. Our hybrid flow model, allows the use of both paradigms directly on all abstraction levels and can also be applied to applications where the hierarchical decomposition is not possible or introduces further complexity.

**Definition 4 (Hybrid Flow Model).** A hybrid flow model  $\mathcal{F}$  is a 4-tuple  $\mathcal{F} := (A, T, C, L)$ , consisting of a set of activities  $A$ , a set of transitions  $T$ , a set of conditions  $C$ , and a set of constraints  $L$ .

**Definition 5 (Activity).** An activity  $a$  represents an atomic piece of work within a flow. This includes invoking web services, internal computations, notifying a human about a task, or receiving context events indicating changes in the real world.. The set  $A := \{a_1, \dots, a_n\}$  defines all activities of a flow. An arbitrary number of event types can be added to each activity. Let  $\epsilon_a : \mathbb{N} \mapsto \mathcal{P}(U)$  be the event type assignment function for  $a$ , where  $\mathcal{P}(U)$  denotes the powerset over the universe of events types. Further, let  $k$  be the number of event types associated with  $a$ , then  $\epsilon_a(i)$  yields the  $i$ -th event type for  $i \leq k$ , and  $\emptyset$  for  $i > k$ . We write  $\epsilon_a$  for short when referring to the set of all event type sets assigned to  $a$ . Furthermore activities may be marked as mandatory.

Activities in a flow may be executed arbitrarily often and in any order, such as  $a_6$ . A flow can successfully complete its execution when all mandatory activities have been executed at least once. Transitions and constraints limit this flexibility and impose structural ordering on the flow activities, such as the response constraint between  $a_4$  and  $a_5$ . When an activity is started it registers at the CMS for context events of its event types  $\epsilon_a$ . As example, let  $e \in U_e$  be an event instance of type  $u$  that the activity  $a_1$  **note results** in the flow requires to complete its execution. Let  $u = \text{write}$  and  $u \in E_\alpha$ , where  $E_\alpha$  contains the event types  $\{\text{wash}, \text{dry}, \text{write}, \text{fetch}, \text{disinfect}\}$  representing some activities of the nurse. When the engine enters the execution of  $a_1$ ,  $E_\alpha$  is registered at the CMS.

**Definition 6 (Transition).** Given a set of activities  $A$ , the set of all transitions within a flow is  $T \subseteq A \times A$ . A transition  $t = (a_x, a_y)$  represents a directed control flow dependency from  $a_x$  to  $a_y$  with  $a_x, a_y \in A$ . A transition is annotated with exactly one transition condition, that is referred to as  $c(t)$ . Further, we define  $d_{in}(a_i) := |\{(a_x, a_y) \in T | a_i = a_y\}|$  and  $d_{out}(a_i) := |\{(a_x, a_y) \in T | a_i = a_x\}|$  as degree of incoming and outgoing transitions for an activity.

The transitions allow certain control flow variants (cf. Figure [III](#)): linear sequences ( $d_{out}(a_4) = 1$ ), parallel branching ( $d_{out}(a_1) > 1$ ) and joins like for ( $d_{in}(a_4) > 1$ ), and combinations of those. Conditional decisions can be made taking the transition conditions into account.

**Definition 7 (Context Condition).** A context condition  $c$  is inductively defined as  $c \rightarrow u | (c_1 \vee c_2) | (c_1 \wedge c_2) | \neg(c_1)$  with  $u \in U$  and  $c_1, c_2$  are already valid conditions and the common semantics for the probabilistic logical operators.

The condition  $c(t)$  for  $t = (a_x, a_y)$  is evaluated when  $a_x$  has received an event instance  $e$  for every  $\epsilon_a$ . We insert the received event instances and check  $c[u/I_E^e(u)] \geq t_n$  against the navigation threshold  $t_n$ . If the equation is fulfilled, the condition evaluates to true and the activity  $a_y$  is executed.

**Definition 8 (Constraint).** A constraint  $l$  is an expression in linear temporal logic (LTL) that defines the temporal ordering of one or more activities in the flow.  $l$  is inductively defined as  $l \rightarrow a$  (logical or) |  $(l_1 \vee l_2)$  (logical or) |  $(l_1 \wedge l_2)$  (logical and) |  $\neg(l_1)$  (logical negation) |  $(l_1 \rightarrow l_2)$  (logical implication) |  $\diamond(l_1)$  (eventually) |  $\square(l_1)$  (globally) |  $l_1 U l_2$  (strong until), where  $a \in A$  and  $l_1, l_2$  are already valid constraints. The literals given in the expression  $l$  denote the completion of the respective activity  $a$  in the flow.

Constraints can be grouped in different classes of constraints such as existence, (negative) relation, (negative) order [16] and provided in a graphical representation (c.f. Figure 11). At runtime they are converted to finite state machines (FSM) [18] and can be checked online for violations. If the FSM is in an accepting state the constraint is *valid*. When the FSM is not in an accepting state the constraint is *temporary violated*. The subsequent execution of further activities can eventually lead to a valid constraint. For example consider the response constraint for  $a_4$  and  $a_6$ . It is *valid* as long as  $a_4$  has never been executed. After  $a_4$  has been executed, the constraint becomes *temporary violated* until  $a_6$  has been executed. A constraint is permanently violated if the FSM reaches an error state and no sequence of activities can fulfill the constraint anymore. The response constraint can never be permanently violated. In contrast, the not succession constraint becomes *permanently violated* if both  $a_3$  and  $a_5$  have both been executed in the same flow instance. A flow can successfully complete its execution iff all constraints are valid. Arbitrary constraints are possible, but the common constraints are given in a graphical notation for the ease of modeling. For example, the not-succession constraint depicted in Figure 11 would be defined as  $\square(a_3 \rightarrow \neg(\diamond(a_5)))$ .

The execution of the flow model yields a flow trace. When an activity is completed, this is recorded in the flow trace along with the event instances it received.

**Definition 9 (Flow Trace).** A flow trace  $\mathcal{T}$  is a sequence of completed activities  $\mathcal{T} := (a_1, \dots, a_j)$  in ascending order of completion times. The event instances each activity has received are also stored within the trace. Let  $\theta(\mathcal{T}, a, u) \mapsto e$  be a function that yields the event instance  $e \in u$  associated with activity  $a$  in trace  $\mathcal{T}$ .

From a single trace, it is possible to reconstruct the actual execution of a flow instance and which context information, i.e. event instances, lead to this execution. All traces are stored in a flow history documenting the executions for later analysis. We use the flow history of a flow model as the data set for training the FlexCon algorithm later.

## 5 FlexCon

We will first provide an overview of the working principle and architecture of FlexCon using a concrete example based on the scenario and flow we presented. Following that, we explain our method to create the DBN from the flow in detail and how we adopted particle filtering techniques for our approach.

## 5.1 Overview

The main goal of FlexCon is to decrease the uncertainty of an event instance  $e$ . This means, if  $e$  is of type  $u$ , then FlexCon shall collect additional evidence for this fact and increase the probability  $p = I_E^e(u)$  for the event type  $u$  in the given distribution. To achieve this we use the flow as additional source of information. The flow model provides information concerning the structure (activities, transitions, constraints) of the flow and, thus, about the expected temporal relation of respective context events. The flow instance provides information given by its execution state, i.e. the current state of the activities and the already received context events.

Let us assume that the flow engine has started the execution of  $a_1$ , and receives the event of the types associated with  $a_1$ , including  $E_\alpha$  (c.f. Section 4.2). In a system without FlexCon, the flow engine would simply compare the probability  $p = I_{E_\alpha}^e(u)$  with the engine's *navigation threshold*  $t_n$  and execute the respective transition if  $p > t_n$ . This simple approach is depicted in Figure 2 on the left. FlexCon, in the other hand, uses the information encoded in the flow model and the flow instance to infer additional evidence for the fact that  $e$  is of type  $u$ . Thus, it improves the probability distribution  $I_{E_\alpha}^e$  that is the basis for the threshold comparison, leading to a more robust flow navigation.

FlexCon uses *Dynamic Bayesian Networks* to interpret context events depending on the current state of the flow. A DBN is a probabilistic data structure that is flexible enough to represent the current flow state, the already received events, and the relation between the events according to the transitions and constraints of the flow model. FlexCon builds the structure of the DBN from the flow model and trains the DBN using traces of previously executed flows. This is shown in Figure 2 on the lower right. We explain the details of the construction algorithm in the next section.

When a flow instance is executed, every incoming context event  $e$  is sent to the DBN. Any such event is associated with a probability distribution  $I_E^e$  (cf. Definition 3). The DBN infers an *additional* conditional probability distribution  $I_E^e$  for  $e$  over  $E$ . The distribution  $I_E^e$  given by the CMS and  $I_E^e$  given by the DBN are combined, yielding an overall distribution  $I_{E_\alpha}^e$  which is then used by the flow engine to make its navigation decision. Our evaluations show that if  $e \in u$  then, on average,  $I_{E_\alpha}^e(u) > I_E^e(u)$ . Hence, FlexCon reduces the uncertainty contained in the original distribution such that the flow engine can make more correct threshold decisions.

Using exact inference to get  $I_E^e$  from a complex DBN, such as the one built from the flow model, is computationally infeasible. Therefore, we use an approach based on *particle filters* [19] to increase the performance. We adapted the standard particle filter approach to reduce the computational effort, which allows us to use more particles on a more sparse DBN network and achieve more accurate inference results. We present a detailed description of the inference algorithm in Section 5.3.

## 5.2 Dynamic Bayesian Network - Structure and Learning

A Bayesian Network  $\mathcal{BN} = (\bar{X}, D)$  is a directed acyclic graph representing a joint probability distribution over a number of random variables (RVs)  $\{X_1, \dots, X_n\} =$

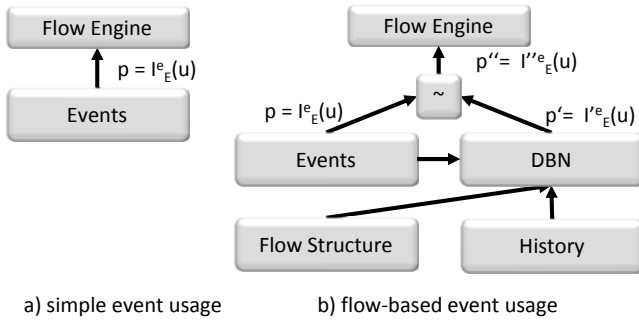


Fig. 2. Architecture overview

$\bar{X}$ .  $\bar{X}$  represents the nodes and the edges  $D \subseteq \bar{X} \times \bar{X}$  define a conditional dependency from the source RV to the target RV. In FlowCon, we used BNs as the flows where based on imperative models that specify the complete execution order. Therefore, the simple static BNs were sufficient. The hybrid model in FlexCon, however, introduces much more freedom for the users to drive the flow forward in different ways and, thus, more dynamics. The static BN model does not support such a dynamically changing probabilistic process. Therefore, FlexCon employs Dynamics Bayesian Networks which are tailored for dynamically changing systems.

In a DBN [19,20], the state of the RV changes over time and the observed values for the RV in the current *time slice*  $\bar{X}_t$  depend on the observations of one or more previous time slices. This dependency is expressed by the *transition model*  $\mathcal{TM} = P(\bar{X}_t | \bar{X}_{t-1})$ . When we write  $X_{1,0}$ , we refer to the RV  $X_1$  in the time slice  $t = 0$ . Additionally, a DBN has a prior distribution  $\mathcal{PD} = P(\bar{X}_0)$  for time  $t = 0$ , such that the definition of a DBN is given as follows:  $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD})$  [2].

**DBN Construction.** Let  $\mathcal{F}_1 = (A, T, C, L)$  be the flow model from our example in Section 3. For each  $a \in A$  and each  $E \in \epsilon_a$ , FlexCon creates a node in the DBN. More formally, the function  $\chi : A \times \mathcal{P}(U) \rightarrow \bar{X}$  maps an activity  $a$  and an event type set  $E$  to a unique RV  $X$  of the DBN. Let further  $\bar{\chi}(a, \epsilon_a)$  be the set of all RVs associated with activity  $a$ .  $\chi(a, E) = X$  with  $E \in \epsilon_a$  is discrete and can assume the same values present in the event type set  $E$  plus a *null* class, represented by  $\perp$ . For example, let us consider  $a_1$  and  $E_\alpha \in \epsilon_{a_1}$  (c.f. Section 4.2). The respective random variable  $\chi(a_1, E_\alpha) = X_\alpha$  can assume any value from  $\{wash, dry, write, fetch, disinfect, \perp\}$ .  $\chi(a, E)_t$  and  $\bar{\chi}(a, \epsilon_a)_t$  refer to the respective RVs in time slice  $t$ .

The *time slices* in our DBN are defined with respect to the execution state of the flow: Every time an activity completes its execution and the flow state is changed accordingly, we enter the next time slice in the DBN. FlexCon creates the transition model (the time dependencies) from the transitions and constraints

<sup>2</sup> Since FlexCon has no hidden variables, there is no need for a sensor model as it is usually found in the DBN definition.



in the flow model. Both of them enforce an execution order on the set of activities. We map these order relations to the transition model, introducing directed edges (dependencies) from one time slice to the next. The strength of these dependencies is learned from flow traces (past flow executions) in a subsequent step. In the following, we describe the construction and learning phases first for transitions and then for constraints.

A transition  $t = (a_x, a_y) \in T$  between two activities represents a very strong dependency as  $a_y$  can only be executed when  $a_x$  has been completed. Therefore, we create a dependency in the network for a pair of RVs if a transition exists between the respective activities as follows.

$$(\chi(a_x, E_x)_t, \chi(a_y, E_y)_{t+1}) \in P(\bar{X}_{t+1} | \bar{X}_t) \iff ((a_x, a_y) \in T) \wedge (E_x \in \epsilon_{a_x}) \wedge (E_y \in \epsilon_{a_y}).$$

For example, consider the activities  $a_1$  and  $a_2$  in Figure 4. They have a transition and, therefore, each  $X \in \bar{\chi}(a_2, \epsilon_{a_2})_{t+1}$  would have  $\chi(a_1, E_\alpha)_t$  as parent node, because  $E_\alpha \in \epsilon_{a_1}$ .

As constraints usually provide a less strict ordering of activities it is more difficult to derive the correct dependencies for the transition model. These dependencies can be different for each execution trace of the same flow. Let  $l_1 = \square(a_3 \rightarrow \neg(\diamond(a_5)))$  represent the *not-succession* constraint in the example in Figure 4. First, FlexCon assumes that there is a bidirectional dependency between all the activities that are contained as literals in the expression ( $a_3$  and  $a_5$  in the example). Hence, FlexCon adds  $(X_{3,t}, X_{5,t+1})$  and  $(X_{5,t}, X_{3,t+1})$ , with  $X_3 \in \bar{\chi}(a_3, \epsilon_{a_3})$  and  $X_5 \in \bar{\chi}(a_5, \epsilon_{a_5})$  as dependencies in the DBN. In a second step, FlexCon determines the type of dependency that has to be included in the transition model  $\mathcal{TM}$ . If the sequential execution of the originating activity  $a_3$  and the target activity  $a_5$  of the dependency *permanently violates* the constraint (as is the case in the example), FlexCon marks this dependency as *negative*. Negative dependencies are handled differently in the learning process as described below. If the sequential execution leads to a *valid* or *temporarily violated* constraint (c.f. Section 4.2), the dependency is handled like a transition. If the subsequent execution of the two activities has no influence on the constraint, we do not add a dependency at all. The latter is the case for the *response* constraint between  $a_4$  and  $a_6$  in Figure 4, where the execution of  $a_6$  has absolutely no dependency on the execution of  $a_4$ .

**DBN Learning.** In order to learn the strength of dependencies in the DBN, we use the flow history as training data, counting the occurrences of all event pairs and learning their joint probability distribution. The portion of the flow history that is relevant for the learning is controlled by a sliding window algorithm taking only a number of recent traces into account. This helps in controlling the effectiveness of the learning procedure in the face of a changing behavior of the flow system.

For dependencies originating from flow transitions, the simple counting algorithm as explained above is sufficient. For constraints, we have to apply a different mechanism: In order to learn the strength of negative relations, we increase the count of the *null*-class for every trace where no such event sequence

could be observed. This leads to a reduced probability of any other event type of the respective event type set. As an example, consider the *not succession* constraint of  $a_3$  and  $a_5$  again. The execution of  $a_3$  will indicate that  $a_5$  is never going to happen in any valid execution of this flow instance. Therefore, we reduce the belief of the DBN that any of the events associated with  $a_5$  is likely to be recognized. An inexperienced nurse may execute the activity sequence  $a_3, a_5$  nonetheless, but the flow can provide guidance for this case, preventing the nurse from violating the constraint  $l_1$ .

**DBN Initialization.** Finally, we need to initialize the DBN for  $t = 0$ , and provide the prior distribution  $\mathcal{PD} = P(\bar{X}_0)$ . This distribution is also extracted from the flow history: We search for traces of the respective flow model and create individual distributions for all the activities the flow has been started with at least once. For  $\mathcal{F}_1$ , this includes  $a_1, a_5$  and  $a_6$ , and the distribution for  $E_\alpha \in \epsilon_{a_1}$  could have the following values:  $P(\textit{wash}) = 0.01$ ,  $P(\textit{dry}) = 0.01$ ,  $P(\textit{write}) = 0.85$ ,  $P(\textit{fetch}) = 0.05$ ,  $P(\textit{disinfect}) = 0.01$  and  $P(\perp) = 0.07$ . In most of the cases the correct *writing* activity has been recorded. In some cases, *fetch* has been misinterpreted, while sometimes there was no meaningful evidence at all ( $\perp$ ). The rates for the uncommon activities (*wash*, *dry*, *disinfect*) are even lower.

### 5.3 Clustered Particle Filtering

In order to exploit the knowledge encoded in the DBN for a specific flow model, a process called *inference* has to be executed. That is, the posteriori distribution of the variables (nodes) has to be calculated given real *evidence*. In our case, the evidence are the real context events received from the CMS in time slice  $t$ , and the inference is done by computing all the conditional probabilities for the variables in time slice  $t + 1$ . Exact inference is infeasible for complex DBNs like the ones generated from flows. Even more so, as this process is running in parallel to the flow execution: Whenever new evidence is available, the inference has to be done to get the probability distributions for the upcoming context events. Therefore, FlexCon uses a heuristic approach that is based on particle filters [19]. That is, we use a large number of random samples (the particles) from the distribution of the DBN at a certain time slice  $t$  and propagate them through the DBN to approximate the individual distributions associated with each node in the following time slice of the DBN. A particle filter approximates the exact distribution by generating a set of particles  $N(\bar{X})$  for all random variables. The higher the number of particles the better the approximation of the real distribution. But the computation time grows linearly with the number of particles.

To propagate and calculate probabilities in the DBN the filter executes the following four steps. To initialize the filter, it first generates an initial particle set  $N(\bar{X}_0)$  sampled from the prior distribution  $\mathcal{PD} = P(\bar{X}_0)$  given by the DBN. In a second step each particle is propagated to the next time slice ( $t = 1$  in this

case) according to the distribution given by the conditional probability table. In the third step, the particles are weighted with the evidence available at the current time slice. Each particle is multiplied with the probability of the current observation. In the final step, the set of particles is resampled according to the weight of the individual particles. A detailed description of the basic principles has been published by Russel and Norvig [19].

We modified this standard algorithm as explained in the following, to accommodate it to the needs of FlexCon. The result is a *clustered particle filter* that is similar to the F3 filter presented by Ng et al. [21]. First of all, a single particle in FlexCon does not represent a full sample of  $X$  but only a sample of a subset of the variables  $\bigcup_{E \in \epsilon_a} \chi(a, E)$ , i.e. all variables of a single activity. Therefore, we call it clustered particle filtering, where each cluster can also be identified by  $N(\bar{\chi}(a, \epsilon_a))$ . This is an useful abstraction for a number of reasons. Each time slice in the DBN covers the completion of a single activity in the flow. Therefore, it is enough to process particles of that activity. All other particles are only propagated as they may be needed later on. This allows us to increase the total number of particles as the average processing load per particle is decreased. The unprocessed particles can be directly transferred to the same node in the next time slice, without the need for a dependency between these nodes.

For example, consider the trace  $\mathcal{T}_1 = (a_1, a_6, a_3, a_4, a_6)$ . After executing  $a_1$ , the particles from  $\bar{\chi}(a_1, \epsilon_{a_1})_0$  are propagated to  $\bar{\chi}(a_3, \epsilon_{a_3})_1$  since there is a transition  $(a_1, a_3)$ , while  $\bar{\chi}(a_6, \epsilon_{a_6})_0$  are just passed to  $\bar{\chi}(a_6, \epsilon_{a_6})_1$ , without further processing.

The second modification changes the propagation and weighting steps. Usually the full set of evidence, i.e.  $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{X}_t)$ , is available for propagating the particles in time slice  $t$ . As we only process the particles for a single activity  $a$  and only observe the received events for this activity as evidence, we can only rely on the conditional probability  $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{\chi}(a, \epsilon_a)_t)$ , instead. This means that we cannot use the evidence of events that have been observed "outside" of the current cluster  $N(\bar{\chi}(a, \epsilon_a)_t)$ . As a consequence we introduce an small error in the inference. However, the majority of  $X \in \bar{X}$  will be independent from the variables in  $\bar{\chi}(a', \epsilon_{a'})$ , because there is no dependency defined by the flow. Therefore the introduced error is rather low and we actually discuss in Section 6 that not using this evidence makes FlexCon a bit more robust. Alternatively, it would also be possible to *sample* the evidence from the current distribution  $N(\bar{X} \setminus \bar{\chi}(a, \epsilon_a))$  of the other activities, but this also introduces inference errors.

After the propagation phase, the actual observations (i.e. the received event instances) become available to the DBN. We can then weight the particles multiplying the number of particles  $|N(\chi(a, E) = u)|$  for a specific event type  $u$  with the actual probability of the event type given by  $I_E^e(u)$ . Based on the computed weights all the particles for  $\bar{\chi}(a, \epsilon_a)$  are resampled according to the distribution of the weighted particles.

The third modification is the actual processing of the received event instance  $e$  in order to decrease its uncertainty. This step is accomplished after the

propagation of the particles and before the weighting. We compute the conditional probability weights for  $I_E^e$  from the particles in  $\chi(a, E)$ , where the weight

$$p' = \frac{|N(\chi(a, E) = u)|}{|N(\chi(a, E))|}$$

for  $I_E^e(u)$  is just the relative particle frequency, as the distribution in the sample  $N(\bar{\chi}(a, \epsilon_a))$  represents a sufficient approximation of the correct conditional probability distribution. All probabilities  $p = I_E^e(u)$  are added to the respective  $p'$  and the resulting distribution is normalized again, yielding  $I_E^{\prime\prime e}(u)$ .

---

**Algorithm 1.** Clustered Particle Filter Algorithm
 

---

```

Input:  $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD})$ ,  $a$ ,  $e[]$ 
if  $N(\bar{X}) = \emptyset$  then
   $N(\bar{\chi}(a, \epsilon_a)) \leftarrow \text{createInitialParticleSet}(\mathcal{PD})$ 
end if
5: for all  $e \in e[]$  do
   $\text{weightEvent}(e, I_E^e, \chi(a, E))$ 
   $\text{weightParticles}(N(\chi(a, E)), I_E^e)$ 
   $N(\chi(a, E)) \leftarrow \text{resampleParticles}(N(\chi(a, E)))$ 
end for
10:  $\text{propagateParticles}(N(\bar{\chi}(a, \epsilon_a)), \mathcal{TM})$ 

```

---

Algorithm 1 depicts the standard particle filter algorithm including the changes introduced by FlexCon. The input to the algorithm includes the  $\mathcal{DBN}$ , the currently completed activity  $a$  and the set of event instances  $e[]$ ,  $a$  has received.

## 6 Evaluation

For our evaluation, we have generated flows according to a probabilistic pattern-based model [22] that has the same properties as the flows observed in the real-world hospital scenario. We do this to get a number of flows that is large enough to achieve statistical relevance. The flows we generate have the same average number of activities and the same structural properties. Essentially, the ratio between activities that have normal transitions and activities that are connected to other activities by constraints is equal.

Use of *flow patterns* [23] allows us to generate imperative flows based on structures commonly found in human-centric flows. We generate these flows and randomly add a respective portion of unconnected *constraint-based activities* (CBAs) to the flow. Next, we randomly generate constraints and use these to connect the CBAs to the imperative parts of a flow. Finally, the resulting flows are validated by generating traces from them. Flows that produce deadlocks (two or more activities blocking each other due to conflicting constraints) are discarded.

Overall, we generated 165 structurally different flows and 200 traces per flow for our evaluations.

The simulation has three important independent parameters. The first one is the *navigation threshold*  $t_n$  of the flow engine as defined in Section 4. For a higher navigation threshold the flow engine accepts less uncertainty in the context events it receives. We tested  $t_n$  from 0.4 to 0.6 in steps of 0.05.

The second parameter is the *average recognition rate*  $arr$  of the CMS. When a context event  $e$  is created in the CMS,  $arr$  is the average probability assigned to the correct event type in the distribution  $I_E^e$  by the CMS. The remaining probability  $1 - arr$  is geometrically distributed to the other event types of the respective event type set  $E$ .

The *variance*  $v$  is the third simulation parameter. It represents the noise added to the distribution  $I_E^e$  created by the CMS. The probability of each event type  $u \in E$  is varied by  $\pm v/2$ , and  $I_E^e$  is normalized again. We evaluated the system for variance values between 0.05 and 0.6 in steps of 0.05.

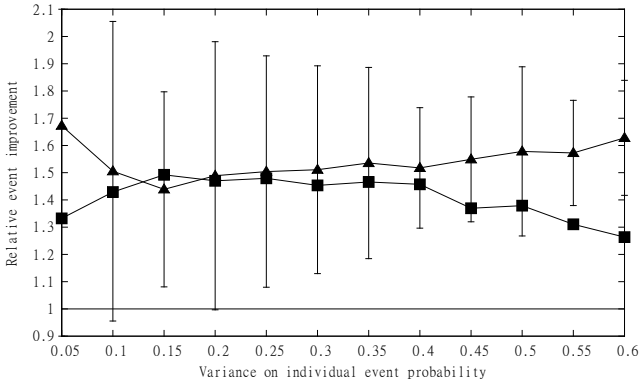
To assess the performance of FlexCon we use the *relative event improvement* and the *number of completed flows* as our two metrics. The relative event improvement  $r$  is defined as  $r = I_E''^e(u)/I_E^e(u)$  for the correct event type  $u$ . If  $r > 1.0$ , then FlexCon was able to provide additional evidence for the occurrence of the correct event type  $u$ , and the flow engine has a higher chance of making the correct navigation decision.

The number of completed flows is simply the percentage of all traces that did complete their execution successfully. We did include the learning of the model in the simulations and the execution starts without a flow history. To put our system further into perspective, we directly compare the results with our previous measurement of the same metrics in FlowCon. Note that the flows in FlowCon are purely imperative. That is, activities are connected by transitions and there are no constraints that leave the decision about the ordering of the activities to the user. Thus, the task of FlowCon is much easier than that of FlexCon due to the additional flexibility of the flows.

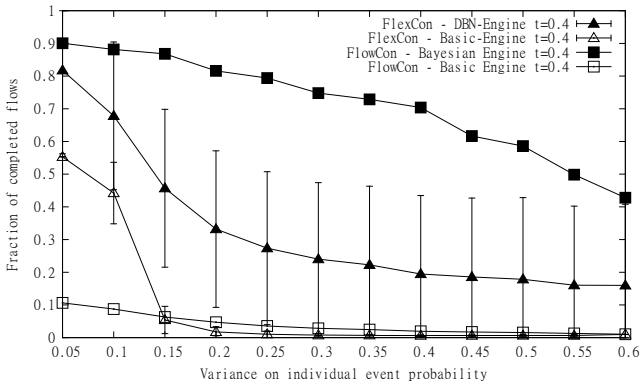
## 6.1 Results and Discussion

The evaluation results are depicted in Figure 3. We only show the results for  $t_n = 0.4$  and  $arr = 0.45$  for clarity. Furthermore, these conditions closely resemble the situation in the hospital and they can be compared best to our previous work.

Figure 3(a) depicts the comparison of the relative event improvement rates for FlowCon and FlexCon. The average event improvement is better for almost all variance values. Even for the higher variances of  $v \geq 0.4$ , where the improvement of FlowCon declines, FlexCon is able to maintain a good improvement, mainly due to the changed method of accuracy improvement: While FlowCon uses all the observed event instances as evidence for calculating the probability of the current event, FlexCon only applies the evidence for the current particle for particle propagation, i.e. independently from other particles. When we misinterpret an event instance from a preceding node this has less impact on the particle filter, as



(a) Comparison of event improvement



(b) Comparison of flow completion

**Fig. 3.** Simulation Results - Comparison between FlowCon and FlexCon

only the propagated particles from this node are influenced, but not the particles from other preceding nodes. Where in FlowCon the whole conditional probability for the current event can be distorted, in FlexCon only a partial result suffers from the misinterpretation. However, if only one parent exists for a given node in the DBN, FlexCon is also sensitive to this kind of misinterpretation, leading to  $r < 1.0$  making the result worse.

The high standard deviation for the event improvement on the flows can be explained by the flows' flexible structure. If two subsequently executed activities are not connected by a constraint or transition, we cannot improve the event in any way as there will be no connection in the DBN between the respective nodes. So according to the flow structure, we have a very high improvement for the dependent events but none for the independent ones.

Figure 3(b) shows the comparison of the flow completion rates, between FlowCon, FlexCon and the respective basic flow engines which do not take any action to decrease the event uncertainty. *FlowCon - Basic* and *FlexCon - Basic* simply

execute the same flows without uncertainty reduction. Both basic systems fail at very low variance values. For  $v \geq 0.15$  less than 6% of the flows can be completed successfully for both basic flow engines. The high values for the basic FlexCon flow engine compared to the basic FlowCon flow engine for  $v = 0.05$  and  $v = 0.1$  result from a changed method of generating the event instance distribution.

The FlexCon DBN-Engine manages to complete 45% of the flows at  $v = 0.15$  and this performance decreases slowly for higher  $v \geq 0.2$ . It is still able to complete 20% of the flows at  $v = 0.6$ .

Again, the standard deviation on the number of completed flows is rather high, for the same reason as above. Some of the flows allow a very good event improvement leading to a reliable execution, after the training phase of the DBN is complete. Those flows (about 5% of the tested flows) exhibit a completion rate of well over 80% and are the main reason for the high standard deviation. Most of the flows are close to the average, and can complete their execution in about 30% of the cases.

## 7 Conclusions and Future Work

We have proposed FlexCon – a system that leverages the application knowledge encoded in workflows to make them more robust against inaccurate and noisy input data. FlexCon uses Dynamic Bayesian Networks and particle filters to reduce the uncertainty of the real-world context events received by *pervasive flows*. Our evaluations show that the uncertainty of an event received by a flow is reduced by 54% on average and the percentage of successfully completed flows is increased by 23-40%.

FlexCon is an important step towards applying flow technology as a part of pervasive systems. In real-world scenarios, found e.g. in the health care domain, users need to be supported in their activities without obstructing them. Thus, the flows need to automatically synchronize with their activities based on collected data such that users are not required to communicate with the flow explicitly. Especially in the health care domain, any explicit interaction (using touch screens etc.) may have severe implications in terms of hygiene.

The sensor data that is used to infer the current activity of a user is characterized by a high level of noise and inaccuracy. FlexCon offers a way to infer more reliable information from this data and, thus, render the respective flows more robust.

In our future work, we will investigate, if more sophisticated approaches to map the flow to a DBN yield a better event improvement. Furthermore we will optimize the number of particles used during the execution to speed up performance of the approach. Depending on the success we will adapt the prototype for a smart-phone, deploy in the hospital and study the usefulness. Furthermore we study the impact of a different uncertainty model on the recognition accuracy and the algorithm performance.

## References

1. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall PTR (2000)
2. Dadam, P., Reichert, M., Kuhn, K.: Clinical Workflows - The Killer Application for Process-oriented Information Systems? In: Proc. 4th Int'l Conference on Business Information Systems, pp. 36–59. Springer, Heidelberg (2000)
3. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. *Computer Science—Research and Development* 23(2), 99–113 (2009)
4. Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards context-aware workflows. In: Pernici, B., Gulla, J.A. (eds.) CAiSE 2007 Proceedings of the Workshops and Doctoral Consortium, vol. 2. Tapir Academic Press, Trondheim Norway (2007)
5. Herrmann, K., Rothermel, K., Kortuem, G., Dulay, N.: Adaptable Pervasive Flows—An Emerging Technology for Pervasive Adaptation. In: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, pp. 108–113. IEEE Computer Society (2008)
6. Kunze, K., Lukowicz, P.: Dealing with sensor displacement in motion-based onbody activity recognition systems. In: Proceedings of the 10th International Conference on Ubiquitous Computing. UbiComp 2008, pp. 20–29. ACM, New York (2008)
7. Bahle, G., Kunze, K., Lukowicz, P.: On the use of magnetic field disturbances as features for activity recognition with on body sensors. In: Lukowicz, P., Kunze, K., Kortuem, G. (eds.) EuroSSC 2010. LNCS, vol. 6446, pp. 71–81. Springer, Heidelberg (2010)
8. Wolf, H., Herrmann, K., Rothermel, K.: Robustness in Context-Aware mobile computing. In: IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2010), Niagara Falls, Canada (October 2010)
9. Barger, T., Brown, D., Alwan, M.: Health-status monitoring through analysis of behavioral patterns. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 35(1), 22–27 (2005)
10. Najafi, B., Aminian, K., Paraschiv-Ionescu, A., Loew, F., Bula, C., Robert, P.: Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. *IEEE Transactions on Biomedical Engineering* 50(6), 711–723 (2003)
11. Biswas, J., Tolstikov, A., Jayachandran, M., Fook, V.F.S., Wai, A.A.P., Phua, C., Huang, W., Shue, L., Gopalakrishnan, K., Lee, J.E.: Health and wellness monitoring through wearable and ambient sensors: exemplars from home-based care of elderly with mild dementia. *Annales des Télécommunications* 65(9-10), 505–521 (2010)
12. Wieland, M., Käppeler, U.P., Levi, P., Leymann, F., Nicklas, D.: Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In: Tagungsband INFORMATIK 2009 Ü Im Focus das Leben, 39. Lecture Notes in Informatics (LNI), Jahrestagung der Gesellschaft für Informatik e.V (GI), Lübeck (2009)
13. Adam, O., Thomas, O.: A fuzzy based approach to the improvement of business processes. In: First International Workshop on Business Process Intelligence (BPI 2005), pp. 25–35 (September 2005)
14. Urbanski, S., Huber, E., Wieland, M., Leymann, F., Nicklas, D.: Perflows for the computers of the 21st century. In: IEEE International Conference on Pervasive Computing and Communications, PerCom 2009, pp. 1–6 (March 2009)



15. Wolf, H., Herrmann, K., Rothermel, K.: Modeling Dynamic Context Awareness for Situated Workflows. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 98–107. Springer, Heidelberg (2009)
16. Pesic, M., Schonenberg, H., van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: IEEE International Enterprise Distributed Object Computing Conference, p. 287 (2007)
17. Aalst, W.M., Adams, M., Hofstede, A.H., Pesic, M., Schonenberg, H.: Flexibility as a Service, pp. 319–333. Springer, Heidelberg (2009)
18. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: Proceedings, International Conference on Automated Software Engineering (ASE 2001), pp. 412–416. IEEE Computer Society (2001)
19. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall (2002)
20. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis. University of California, Berkeley (2002)
21. Ng, B., Peshkin, L., Pfeffer, A.: Factored particles for scalable monitoring. In: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, pp. 370–377. Morgan Kaufmann (2002)
22. Chiao, C., Iochpe, C., Thom, L.H., Reichert, M.: Verifying existence, completeness and sequences of semantic process patterns in real workflow processes. In: Proc. of the Simpósio Brasileiro de Sistemas de Informação, Rio de Janeiro, pp. 164–175. UNIRIO, Brazil (2008)
23. Lau, J.M., Iochpe, C., Thom, L.H., Reichert, M.: Discovery and analysis of activity pattern co-occurrences in business process models. In: ICEIS, vol. 3, pp. 83–88 (2009)

# An Artifact-Centric Approach to Dynamic Modification of Workflow Execution\*

Wei Xu<sup>1</sup>, Jianwen Su<sup>2</sup>, Zhimin Yan<sup>1,4</sup>, Jian Yang<sup>3</sup>, and Liang Zhang<sup>1</sup>

<sup>1</sup> School of Computer Science, Fudan University, China

<sup>2</sup> Department of Computer Science, UC Santa Barbara, USA

<sup>3</sup> Department of Computing, Maquaire University, Australia

<sup>4</sup> Real Estate Information Center, Hangzhou, China

**Abstract.** Being able to quickly respond to change is critical to any organizations to stay competitive in the marketplace. It is widely acknowledged that it is a necessity to provide flexibility in the process model to handle changes at both model level as well as instance level. Motivated by a business policy rich and highly dynamic business process in the real estate administration in China, we develop a dynamically modifiable workflow model. The model is based on the artifact-centric design principle as opposed to the traditional process-centric approach. Runtime execution variations can be specified as execution modification rules, which lead to deviations to the normal executions. With the support of rules and declarative constructs such as *retract*, *skip*, *add*, and *replace*, ad-hoc changes can be applied to execution at anytime depending on the runtime data and the instance status gathered through the use of artifacts in our model.

## 1 Introduction

Changes of business policies and operational routines can lead to unavoidable business processes restructuring. Currently available workflow management systems (WfMS) have provided basic support in modeling and enactment, but with limited flexibility in handling changes, especially unanticipated and just-in-time changes. Such ad hoc changes happen frequently in many organizations and are witnessed in business sectors such as banking, clinic trails, and other administrative intensive task management. For example, in a bank reporting system, it may decide to send the reports via email instead of postal mail for some special transactions; it may decide to inform the customers and produce reports quarterly instead of monthly unless the customer declares otherwise and is willing to pay extra fees.

Various process models, languages, and mechanisms in the past offer flexibility in business processes. From language perspective, there are declarative workflow languages [10] which adopt a complete declarative approach to specify process logic without explicitly modeling of control flow and data flow as in the procedural workflow. From mechanism perspective, FLOWer [2] and ADEPT [13] provide deviation operations such as “undo”, “redo”, and “skip”, and verification techniques to ensure applicability of operations at runtime. From the modeling perspective, most work is based on the traditional

---

\* Supported in part by NSF grant IIS-0812578, NSFC grant 60873115 and a grant from IBM.

activity-centric workflow models. Recently, there is a growing interest in *artifact-centric* workflow models where business artifacts (objects with lifecycle) are the modeling focus. The artifact-centric approach provides much needed ease in managing runtime change since runtime status information (of enactments) are readily available in artifacts.

Providing process flexibility to support foreseen and unforeseen changes is a very active research area recently. It is widely recognized that runtime process execution flexibility is crucial for managing business process lifecycle and it needs special attention [20,15,19]. The real challenge in managing dynamic workflow execution is to provide a systematic and integrated support for the process designer to specify how the process would react to various runtime changes and for the process to evolve gracefully in a controlled, incremental, and predictable manner.

In this paper we will go through an analysis of a real estate administration workflow system to illustrate various of business rule and policy changes that can happen at runtime, and their impact on the running workflow system. We develop an artifact-centric model, DEZ-Flow, which is a hybrid model for utilizing the benefits of both declarative and procedural workflow languages. Since artifacts (with activities, policies) are first class citizens in the model, runtime status can be easily captured. This provides a solid foundation for managing just-in-time and ad hoc changes. We then present a rule-based language that is used to specify logical conditions when the deviations are needed and how the deviations shall take place. These rules are managed separately from the process model; updating and changing can be easily applied to the rules without affecting the main workflow model. Finally, we present a scheduler that works together with a flexible task performer to manage just-in-time changes happened to running instances based on the artifacts and rules.

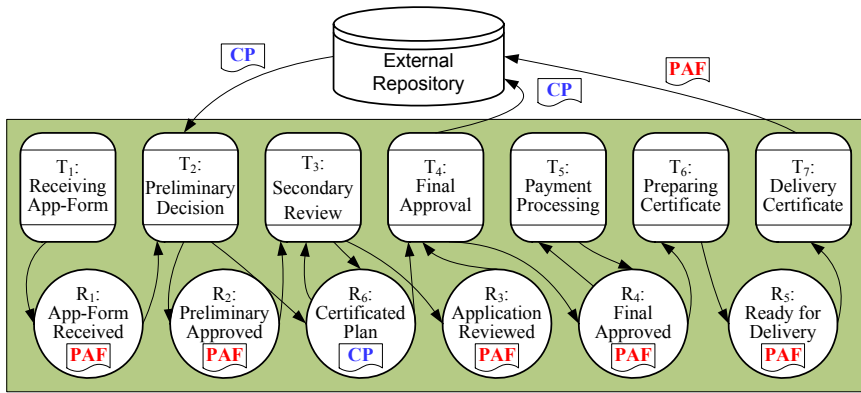
In summary, we present a novel and functional mechanism to handle ad hoc and just-in-time changes at runtime. The workflow schema of the running workflow can stay the same, the changes only lead the affected instances to execution deviations from the original execution path according to the rules. This approach is particularly suitable for temporary changes that exhibit long tail characteristics, and the changes can evolve or disappear over time. Utilizing a hybrid model, i.e., a procedural workflow schema defining “sunny day” processing and a declarative language for specifying changes, distinctions among various types of changes (e.g., as classified in [20,19,15]) disappear. Thus DEZ-Flow provides a uniform and flexible way for handling runtime changes.

The remainder of the paper is organized as follows. We provide a motivating example in Section 2. In Section 3, we focus on the artifact-centric workflow model EZ-Flow. We then provide a detail discussion in Section 4 on the extension of EZ-Flow to support dynamic modification of execution. In Section 5, we analyze the flexibility and effectiveness of DEZ-Flow. Finally, we conclude the paper in Section 6.

## 2 Motivating Example

We will in this section present a simplified example from a real business process developed for the Real Estate Administration (REA) in Hangzhou, China.

The process *Certificate Approval for the Preselling of Apartments* (or *CAPA*) is to evaluate if the city government would issue a certificate to a building contractor for selling her buildings according to the law and regulations [8]. A simplified CAPA process



**Fig. 1.** The Example Workflow CAPA

schema is shown in Fig. 1, which adopts an artifact-centric workflow model (described in Section 3). For the CAPA process in Fig. 1, there are 7 tasks (represented as rounded boxes),  $T_1$ : *Receiving App-Form*,  $T_2$ : *Preliminary Decisions*,  $T_3$ : *Secondary Review*,  $T_4$ : *Final Approval*,  $T_5$ : *Payment Processing*,  $T_6$ : *Preparing Certificate*,  $T_7$ : *Deliver Certificate*; and two artifacts *Certificated Plan* (CP) and *Presale Application Form* (PAF) that can go through the process by applying tasks. PAF is the “core” artifact class (explained in Section 3). The states of artifacts are reflected in repositories ( $R_1$  through  $R_6$ , represented as circles). For example, the task *Secondary Review* changes the state of a PAF artifact from *Preliminary Reviewed* repository to its next repository *Application Reviewed*. Every REA bureau in China heavily relies on business processes such as CAPA for their administrative tasks. Taking a medium city Hangzhou for instance, there are about 300,000 cases annually.

A big challenge facing the REA authority is the adaptability of the deployed CAPA workflow. For a highly dynamic business like the real estate market in China, legislations and national policies can change in an ad hoc way. Some policies are temporally developed to either restrict or encourage certain groups of building contractors; while others may have long term effects to regulate the housing market. Consider some cases happened recently:

- **Case 1:** special scheme can be introduced under special circumstances. For example, a green channel is opened for the project of natural disaster victim resettlement. In this case the three reviewing tasks *Preliminary Decision*, *Secondary Review*, and *Final Approval* can be omitted in the review process. Similar policies can also be applied to other cases, e.g., to elderly people and disabled persons so that they can complete the transaction sooner.
- **Case 2:** a new policy is put in place to differentiate groups of building contractors. A new task *Pre-qualification Check* needs to be created and added in the process for every building contractor who has never applied for approval before.
- **Case 3:** a new policy is introduced to the application for affordable housing for low-income families. For any applications falling into this category, the task *Payment Processing* will be replaced by a similar but different payment task.

- **Case 4:** a new policy is introduced stating that during the review process, the accumulated selling area in a PAF must not exceed the planned total area in the corresponding CP for a building project. Otherwise the application under the consideration needs to be retracted to a previous case depending on the circumstances.

This is a typical case of *retract* in daily activities involved in administrative approval in China. The arbitrary nature of the retraction in this case makes it very difficult to model the reaction paths in the workflow schema.

The above merely lists a few newly introduced policies that can affect the process structure. In fact, different cities may have different criteria for defining affordable housing and for deciding on disaster victims. These adjustments can be temporary and vary under different circumstances. Currently, changes and variations are handled either manually or through workflow reconstruction. Typically, a REA bureau of a medium city such as Hangzhou can have over 500 workflow schemas.

It is desirable to provide support for business process change. In this paper, we are specifically interested in a holistic solution for a runtime and just-in-time change handling mechanism that requires only a small deviation from original schema to support ad hoc changes and avoids workflow schema reconstruction. An advantage of this approach is that the changes introduced can be discarded and/or changed again easily.

### 3 EZ-Flow: An Artifact-Centric Workflow Model

In this section, we describe the technical model of EZ-Flow (named ArtiFlow in [8]) needed for presenting the dynamic modification mechanism in the next section. The central notions in EZ-Flow include “artifacts” and “classes”, “tasks”, and “workflow schemas”. In addition to these concepts, we also describe a simple execution engine that manages (schedules) the execution of workflows.

In EZ-Flow, key business entities are modeled as artifact classes. Unlike business documents or objects, artifacts include both the data for the workflow as well as states of workflow execution [9]. Each (*artifact*) *class* has a *name* and a set of associated *attributes*. Every attribute has a *name* and a *type* for data values of the attribute. Attribute types include *string*, *integer*, *real*, *Boolean*, and a class name. The domains of these types are standard, attribute values for a class name are elements in an infinite set of artifact *identifiers*. Since classes can reference each other through attributes, we further require that a set of classes is *well formed* which means that (1) every class has a distinct name, and (2) every class referenced in an attribute of some class is also a class in the set (closure of referencing).

**Example 1.** Consider the artifact class PAF in the CAPA workflow discussed in Section 2. PAF includes the contractor (applicant) name, pre-sale project information, and real estate bureau office approval results as follows.

PAF (*contractorName*:string,  
*projectName*:string, *projectType*:string, *sellingArea*:float, *cp*:CP, *payment*:float,  
*preliminaryApp*:boolean, *reviewApp*:boolean, *finalApp*:boolean)

where CP is the following artifact class that includes the certificated plan number and planned total area for sale, relevant for CAPA workflow:

CP (*cpNo*:string, *planArea*:float)

■

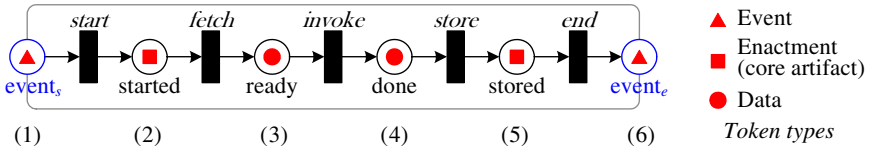


Fig. 2. Executing One Task: An Illustration

Each workflow schema has a primary artifact class, called a *core* artifact class. A core artifact carries both data and the enactment (execution state etc.) it is involved in. This reflects the general principle that artifact lifecycle describes business process [9]. If a workflow needs other artifacts, these artifacts are called *auxiliary* artifacts.

In a workflow specification in EZ-Flow, artifacts are manipulated by *tasks* in their lifecycles. Each task has a unique name and a *signature* that is a set of artifact classes that have been defined. An invocation of a task acts on exactly one artifact from each class in its signature. A task can read and write its artifacts.

The execution of a task is accomplished via a sequence of actions. Fig. 2 shows the process of executing a task using a model similar to Petri nets with three types of tokens, “event”, “enactment”, and “data” tokens. An *event* token includes the contents of an event that are used by the task as input or correlation. An *enactment* token contains the core artifact that records the information and progress of the enactment. A *data* token contains all needed artifacts for actually performing the task.

A task execution is triggered by an *event* (similar to BPMN). The *start* transition initiates the task execution by getting the core artifact using the contents in the event token. An enactment token is then generated and placed in the “started” place (step (1) to (2) in Fig. 2). The *fetch* transition receives all auxiliary artifacts for the execution using the information in the enactment token, forms a data token, and places it in the “ready” place (step (2) to (3) in the figure). The task is then performed, indicated by the *invoke* transition (step (3) to (4)). In completing the execution, all auxiliary artifacts are *stored* back to the appropriate repositories (step (4) to (5)), and then the execution is wrapped up and *ends* (step (5) to (6)). Note that the actual execution of the task itself is represented in the *invoke* transition in Fig. 2 as a unit step.

We organize events into *event types*. An event type has a unique name, is associated with one artifact class, and may contain a set of attributes to describe the event contents. Similar to artifact classes, attribute types are either the usual data types or class names. Each *event (instance)* of an event type must have a unique event identifier, and a value for each attribute of the corresponding type. Each task is triggered by one event type, and produces one event upon completion.

During the lifecycle of an artifact, there may be a sequence of tasks that act on it, between each pair of consecutive tasks the artifact must be stored in a “repository.” A *repository* has a unique name and an associated artifact class. At runtime a repository may contain a (possibly empty) set of artifacts of the associated class.

**Definition.** A *workflow schema* is a tuple  $W = (C, \Gamma, E, \Sigma, F, R, L)$ , where

- $C$  is the *core* artifact class of the workflow,  $\Gamma$  is a set of *auxiliary* artifact classes not containing  $C$  such that  $\Gamma \cup \{C\}$  is well-formed,

- $E$  is a set of event types (with distinct names),
- $\Sigma$  is a set of tasks with distinct names,
- $F$  maps each task  $t$  in  $\Sigma$  to a pair  $(e_i, e_o)$  of event types where  $e_i$  triggers  $t$  and  $e_o$  is produced by  $t$ , satisfying the condition that each event type can trigger at most one task and be produced by at most one task,
- $R$  is a set of repositories with distinct names partitioned into: a set  $R_C$  of repositories for the core artifact  $C$ ,  $R_\Gamma$  consisting of exactly one repository for each class in  $\Gamma$ , and a single external repository  $r_{\text{ext}}$ , and
- $L$  is a set of triples  $(x, y, g)$  where (1) either  $x \in R$  and  $y \in \Sigma$  or  $x \in \Sigma$  and  $y \in R$ , and (2)  $g$  is a *guard* on the edge  $(x, y)$ .

In a workflow schema, an event is *external* if it is not produced by any task in the workflow schema. In a system with many workflows, each artifact class is assumed to be the core for at most one workflow schema. (This assumption simplifies naming conventions and can be easily removed.)

The explicit identification of a core artifact class in a workflow was in the recent workflow models GSM [6] and ArtiNet [7]. In fact, this model extends the model in [7] by adding events and conditions on artifacts. Such conditions are called *guards*, and are Boolean conditions on attribute values among the relevant artifacts.

**Example 2.** The CAPA workflow in Section 2 is a workflow schema under the above definition, where PAF is the core artifact, and CP an auxiliary artifact. The seven tasks  $(T_1, \dots, T_7)$ , six repositories  $(R_1, \dots, R_6)$  and an external repository are shown in Fig. 1, where repositories  $R_1, \dots, R_5$  are for PAF and  $R_6$  for CP. The figure also shows the edges between repositories and tasks, but the guards are omitted. Finally there are nine event types in CAPA, the set of task-triggering event-producing event triples is:  $\{(T_1, E_{\text{appForm}}, E_{\text{readyForApproval}}), (T_2, E_{\text{readyForApproval}}, E_{\text{preliminaryApp}}), (T_3, E_{\text{preliminaryApp}}, E_{\text{reviewedApp}}), (T_4, E_{\text{reviewedApp}}, E_{\text{finalApp}}), (T_5, E_{\text{payment}}, E_{\text{paid}}), (T_6, E_{\text{paid}}, E_{\text{readyForDelivery}}), (T_7, E_{\text{readyForDelivery}}, E_{\text{archive}})\}$ . ■

We now describe the semantics of EZ-Flow, based on which the dynamic modification mechanism is developed.

Given a workflow schema  $W$ , we first construct an ArtiNet [7] (a variant of Petri nets)  $N$  corresponding to  $W$  as follows. Each repository in  $W$  is a place in  $N$ , each event type in  $W$  is also a place, each external event also adds a transition with no input and one output place that is the event place, each task in  $W$  is replaced by a sequence of transitions and places as shown in Fig. 2, each link  $(r, \sigma, g)$  from a repository  $r$  to a task  $\sigma$  becomes a link from  $r$  to the *fetch* transition inside  $\sigma$ , each link  $(\sigma, r, g)$  from a task  $\sigma$  to a repository  $r$  becomes a link from the transition *store* inside  $\sigma$  to  $r$ . Note that the first and last places in the sequence are the corresponding event places.

To describe the operations of a workflow, we use “snapshots” to represent “markings” in the sense of ArtiNets. We further construct a snapshot as a database.

A snapshot database contains the following tables: one for each artifact class, each event type, and one stores all workflow enactments, and one stores a set of relationships between an artifact and the enactment using the artifact. Since each workflow has a core artifact class (and one class cannot be the core for two or more workflow schemas), we

<i>PAF</i>	artifactID	contractorName	projectName	sellingArea	cp	...	finalApp	<i>CP</i>	artifactId	cpNo	planArea
	paf02	Greentown	Sunlight Villa II	20000	cp01		true		cp01	Hz2009	100000
	paf01	Greentown	Sunlight Villa I	55000	cp01		true			0827655	

<i>Enactments</i>	coreID	Task	Place	<i>Artifacts</i>	artifactID	enactment	place
	paf02	$T_1$ :Receiving App-Form	ready		paf01	paf01	$R_4$ :Final Approved
	paf01	$T_4$ :Final Approval	event <sub>e</sub>		paf02	paf02	$T_1$ :Receiving App-Form
					cp01	paf01	$R_6$ :Certificated Plan

<i>Epayment</i>	eventId	projectName	payment	<i>Enactment</i>

Fig. 3. A Snapshot  $S_0$  Example for CAPA Workflow

will use an artifact ID from the core class to serve as the workflow enactment identifier. We give details below, assuming  $W$  is a workflow schema.

- For each artifact class  $C$  in  $W$ , there is a table  $C(\underline{artifactID}, A_1, \dots, A_n)$  that stores all artifacts in the class that are currently “active”, where  $A_1, \dots, A_n$  are all attributes in  $C$ ,  $artifactID$  stores the identifier and serves as the key. (An artifact is *active* if it is used in an enactment that has not yet completed.)
- For each event type  $Ev$ , the snapshot includes a table  $Ev(\underline{eventID}, E_1, \dots, E_n, Enactment)$  to store all events that happened but have not yet been processed, where  $eventID$  stores a distinct event identifier,  $E_1, \dots, E_n$  are a listing of all attributes in the event type and  $Enactment$  indicates the workflow enactment (core artifact ID) associated with the event.
- The snapshot contains a table  $Enactments(\underline{coreID}, Task, Place)$  to store all enactments that have not completed, where  $coreID$  is the enactment identifier (the core artifact identifier),  $Task$  denotes the current executing task in the enactment, and  $Place$  reflects the state of the task execution and has one of the six “internal” places (in Fig. 2) including  $event_s$ ,  $started$ ,  $ready$ ,  $done$ ,  $stored$  and  $event_e$ , and
- Finally, the snapshot contains a table  $Artifacts(\underline{artifactID}, enactment, place)$  to indicate which enactment an artifact belongs to and which place (either a place in a task, or a repository) the artifact currently is.

Note that an artifact first occurs in a snapshot when it is created by the workflow or is fetched by a task in the workflow. Once an artifact is in a snapshot, it stays in the snapshot database until the enactment operating on it completes.

A part of a snapshot of the CAPA workflow is shown in Fig. 3. Since the workflow has two artifact classes, two artifact tables are included, for  $PAF$  and  $CP$  resp. It also includes a table for workflow  $Enactments$ . Only one of the nine event tables is shown.

Transitions in EZ-flow are defined in a way extended from ArtiNet [7], due to the presence of data contents in artifacts, guards, and events. Basically, from one snapshot to the next must be caused either by:

1. **An external event arrives.** When one external event comes, an event token will be put into the  $event_s$  place of the associated task, or
2. **Fire a transition.** A transition fired within a task must be one of the following (Fig. 2): *start*, *fetch*, *invoke*, *store*, and *end*. The *start* transition consumes an event token and places a enactment token in the “started” place; the *fetch* transition consumes this token, the core artifact, and auxiliary artifacts, then puts one token in



coreID	Task	Place
paf02	$T_1$ :Receiving App-Form	done
paf01	$T_4$ :Final Approval	event <sub>e</sub>

coreID	Task	Place
paf02	$T_1$ :Receiving App-Form	ready
paf01	$T_5$ :Payment Processing	event <sub>s</sub>

eventId	projectName	payment	Enactment
$e_1$	Sunlight Villa	75000	paf01

(a)Firing an Invoke Transition
(b)Token Generation From an Event

Fig. 4. Snapshot Derived from  $S_0$

“ready” for performing the task (as a simple *invoke* transition). Similarly, *store* transition puts the artifacts back to their repositories and a token in “stored” that will be used to generate an event token in the event<sub>e</sub> place.

The transition firings on snapshots can be easily mapped to updates on the database.

**Example 3.** Consider the example snapshot in Fig. 3. The first row of *Enactments* table shows that a current executing task of *paf02*, the *Receiving App-Form*, and is on its (internal) state *ready*. So a transition *invoke* can be fired in this snapshot. Once it fired, the result snapshot is shown as Fig. 4(a): the value of *Place* in the first row of the table *Enactments* is updated to the place *done*. Fig. 4(b) shows another possible result snapshot which can be derived from the original snapshot shown in Fig. 3. In this case, an external event  $E_{\text{payment}}$  has been put into the event<sub>s</sub> place of task *Payment Processing*. In snapshot shown in Fig. 4(b) a new record  $e_1$  is inserted in the table *Epayment*. Also, in the table *Enactments* the record *paf01* is updated. ■

The EZ-Flow engine consists of two components: (1) A *task scheduler* that responds to events and decides to launch tasks according to the workflow schema. (2) A *task performer* that actually manages the execution of tasks, i.e., performs the tasks according to the semantics as illustrated in Fig. 2. Note that an engine has one instance of scheduler running, each time the scheduler decides on executing a task, a new instance of a task performer is spawned that takes care of performing this particular task. An earlier implementation of the engine (scheduler) was described in [8].

More specifically, events can be generated by tasks (e.g., an  $E_{\text{readyForApproval}}$  event produced by task *Receiving App-Form* in CAPA) or externally (e.g., an  $E_{\text{appForm}}$  event as a result of an applicant submitting an App-Form to the REA office). When an event arrives, it will be put into a queue with its contents recorded in the corresponding table in the snapshot database. The EZ-flow scheduler consumes one event at a time from the queue and spawns a new task performer to execute the task associated to the event. Once a task performer is launched, it manages the task execution by moving through its internal states (event<sub>s</sub>, started, ready, done, stored, and event<sub>e</sub>). At the last state, a new event is also generated and put into both the snapshot database and the queue.

## 4 Dynamic Modification of Execution

In this section, we present a new mechanism to support dynamic, i.e. runtime, modification of execution for EZ-Flow workflow schemas. The mechanism is an initial step towards understanding how workflow executions can be incrementally modified to accommodate changing requirements in artifact-centric models. We believe that such a

dynamic modification framework is an effective tool for addressing the “long tail” problem. We present the technical details of the mechanism, including constructs to change workflow executions, an execution engine to support such modifications at runtime, and a simple rule-based language to specify modifications.

We introduce four types of execution modification constructs: *skip*, *replace*, *add*, and *retract*. These constructs change the execution of workflow in various ways. Roughly speaking, the *skip* construct simply omits the execution of a task; the *replace* construct executes a replacement task instead of a task. The *add* construct executes a specified additional task immediately before a task. The *retract* construct “rewinds” the execution by moving the execution point to a previously completed task. Note that while *retract* appears similar to “roll-back”, it does not attempt to “erase” the completed execution. It is a conceptual level “redo” operation that allows a business manager to, e.g. re-check a case, re-examine qualification conditions, etc. Therefore, it does not need any compensation mechanism. (In some cases, the law requires that all performed actions be recorded on the book.)

In this section, we extend EZ-Flow to a dynamic version named “DEZ-Flow” to support these constructs. Technically, we need to modify EZ-Flow in three aspects.

1. Extending the snapshot database by including more information on execution,
2. Enrich the *fetch* transition that will now use the extended snapshot to locate and obtain the needed auxiliary artifacts, and
3. Modify the workflow scheduler and task performer that will handle execution modifications on the fly when requested.

Execution modification constructs can only be sequentially applied to a single enactment, i.e., each enactment has at most one execution modification at one time. To enforce this, the table *Enactments* is added three new columns: “*currConstruct*” to record the construct (rule) currently being applied, “*addiTask*” to store the additional task (name) involved, and “*addiPlace*” to indicate the execution state (internal place) of executing the additional task. Specifically, when an execution leaves task *A* to apply a modification construct (rule), the following would happen. For *replace* and *add* constructs, *currConstruct* is set to the rule name, *addiTask* is assigned to an associated additional task *B*, and *addiPlace* records the current execution state (place) of *B*. For *skip* construct, *currConstruct* is set to the construct name, (but *addiTask* and *addiPlace* are not used). When the execution returns to *A*, *addiTask* and *addiPlace* are reset to “none”. And after *A* completes *currConstruct* is also set to “none”. For *retract* construct, none of the three columns is used since it is done in a single step.

We now briefly outline the modification to the *fetch* transition. Recall that in EZ-Flow, the *fetch* transition in a task simply retrieves all auxiliary artifacts. The *retract* construct may move the current execution control point back to an earlier task. When this happens, an auxiliary artifact may not be in the original repository as the normal execution would expect. For example, if the construct “*retract* to  $\sigma_1$  from  $\sigma_3$ ” is applied and the first execution of  $\sigma_1$  has already moved an artifact from the external repository to the internal one (for the auxiliary artifact), the second execution of  $\sigma_1$  will not be able to *fetch* the artifact from the external repository. Also, if  $\sigma_2$  is executed immediately after  $\sigma_1$ , it may also change the location of some artifacts. To ensure that *fetch* transition

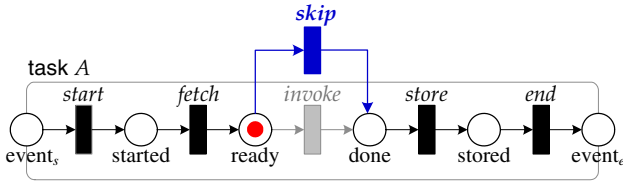


Fig. 5. Semantics of skip Construct



Fig. 6. Semantics of replace Construct

is successful during the re-runs, the *fetch* transition needs to know the current location of each auxiliary artifact. Fortunately, this location information already exists in the *Artifacts* table. Thus the *fetch* transition is slightly generalized from the version in EZ-Flow to always retrieve artifacts from the current locations.

We now discuss the details of the four constructs.

**Skip** *execution of a task.* A skip construct alters the execution of a task in the following manner. Assuming that the task *A* is to be skipped (i.e., the condition is satisfied), Fig. 5 shows the modified execution path of a task. The altered execution almost coincides with the original execution path (Fig. 2) except that the *invoke* transition (shown in gray in Fig. 5) is replaced by a new “skip” transition. The *skip* transition simply marks the task as “skipped” (in the log), makes no changes on the artifact(s), and incurs no resource consumption (e.g., human).

**Replace** *execution of one task by execution of another task with the same signature.* The semantics of the replace construct is illustrated in Fig. 6 where the execution of task *A* is replaced by the execution of task *B* (assuming the condition is satisfied). Similar to skip, replace also modifies the original execution path by having an alternative to the *invoke* transition. Replace is accomplished by a pair of transitions. Since task *B* has an isomorphic sequence of places and transitions, a transition “*begin-replace*” redirects the execution to the “ready” place in task *B* so the *invoke* transition of *B* can proceed (i.e., performing task *B*). Upon completion of *invoke*, a transition “*end-replace*” routes the execution back to task *A*’s place “done”, and the normal execution resumes.

**Add** *execution of a task before executing another task.* Fig. 7 shows the changed execution path after an adding task *B* construct is applied to task *A*. The add construct inserts an execution of task *B* right before the *fetch* transition of *A*. Specifically, a new transition “*begin-add*” routes the execution from the “started” place of task *A* to the “started” place of task *B* (enactment token). This allows additional artifacts to be

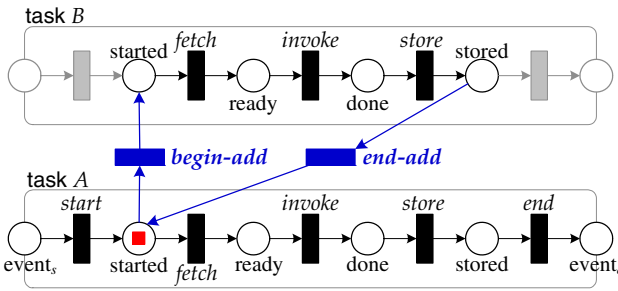


Fig. 7. Semantics of add Construct

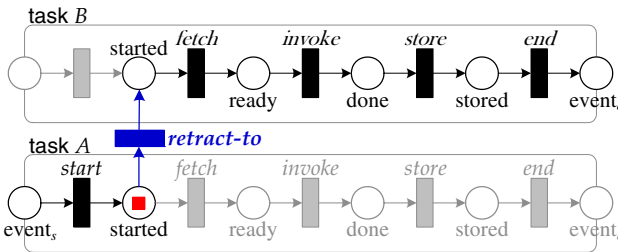


Fig. 8. Semantics of retract Construct

fetches for executing task *B*. After *B* completes and the additional artifacts are stored back, the execution returns the enactment token back to the “ready” place of task *A* using an “end-add” transition. The normal execution resumes.

**Retract** to an earlier task. The retract construct simply routes back to an earlier task so that the execution path since that task can be re-executed. Fig. 8 shows the execution path when at executing task *A*, the execution needs to be retracted to task *B*. Technically, this is accomplished simply by a transition *retract-to* to move the enactment token from the “started” place of task *A* to the “started” place of task *B*. When the workflow schema has no auxiliary artifact classes, the *retract-to* transition is rather simple and Fig. 8 illustrates the semantics well. Note that, when workflow schema has one or more auxiliary artifacts, augmented snapshots and the generalized *fetch* transition are used, which allow the retract construct to be correctly supported.

There are also two modalities when applying a modification construct: “must” and “may”. In the *must-do* modality, the modification construct must be applied, i.e., the alternative execution path will be taken (if condition is satisfied). In the *may-do* modality, the decision for modified execution is external to the workflow engine but the decision must be made before the *invoke* transition. Since the transitions prior to *invoke* do not have side-effects, the transition(s) is(are) simply rolled back and the alternative path is taken if the decision is communicated to the workflow engine.

We next explain how these modification constructs are performed and present the design of a new workflow engine. for dynamically modifiable workflow execution.

Fig. 9 shows (a) the original EZ-Flow engine and (b) the new engine. Recall that the old engine consists of a scheduler and 0 or more task performers (one for each task being performed). Furthermore, as Fig. 9(a) illustrates, the scheduler spawns a task performer

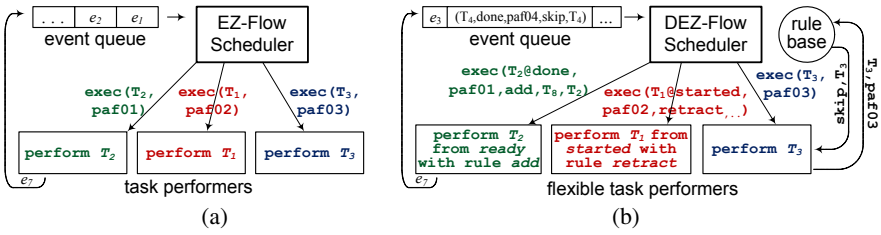


Fig. 9. Dynamic Workflow Scheduler

in response to an event, and the task performer simply performs the task from start to finish (according to Fig. 2). Fig. 9(b) shows the new engine consisting of: a dynamic scheduler, 0 or more flexible task schedulers, and a rule base. Roughly speaking, the rule base maintains a set of rules, each describing one modification construct and its enabling condition, and answers queries from task performers with applicable rules. The dynamic scheduler and flexible task performer implement the execution modification according to the rule.

Workflow administrators and business managers may create execution modification rules (a rule language shown at the end of this section). The *rule base* manages execution modification rules (to allow rules to be created, updated, deleted) and responds to queries from flexible task performers to find applicable rules. When a task performer executes a new task  $\sigma$ , it sends a query of the form “ $(\sigma, \alpha)$ ” to the rule base where  $\alpha$  is the corresponding enactment. The rule base then finds the best applicable rule if it exists. As hinted in Fig. 9 each return result includes the name of the operator and task names. Note that this process may involve querying the snapshot database for checking pre-conditions. If no applicable rules are found, the rule base simply returns “none”.

The second part of the new engine is a collection of *flexible task performers* (or instances). For example, the skip construct only involves one task and it can be completed by one task performer. The replace construct is accomplished by multiple task performer instances: one for task  $A$  till “ready”, another for task  $B$  from “ready” to “done”, still another from “done” of task  $B$  back to task  $A$ , and the last for task  $A$  from “done” to the end. Similarly, the add construct is also done by three task performers, and the retract construct is done by 2.

The transition of control between task performers is done by “extended events” that also include a new event type form task-name@place, enactment, and an operation (construct, and tasks). When the scheduler reads such an event, the event contents are passed to a new flexible task performer instance. The task performer executes the portion of the execution within this task, and generates another event to continue execution.

When a task performer initially starts it checks if the enactment is currently having execution modification. If so, it will perform the operation without considering any other modification constructs. If the DEZ scheduler (see below) receives a rule from the rule base, it launches a performer for a portion of the task according to the defined semantics, i.e., it starts from one internal place, ends at another, and generates an event. Also, the *Enactments* table needs to be marked with the current construct name.

Finally, a *Dynamic Task (DEZ) Scheduler* handles both EZ-Flow events and also new events from flexible task performers. If an EZ-Flow event comes, the scheduler will

launch a task according to the workflow schema exactly the same as in EZ-Flow. For an event of new type, i.e., corresponding to some execution modification rule, the scheduler will also spawn a flexible performer passing along all the necessary information.

The syntax of rule-based language for formulating dynamic modifications allows definition of specific conditions on core artifacts (and the enactments) when modifications may be or must be applied. It is mostly inspired by SQL, each modification operation is specified in a rule form. The following shows an execution modification rule for Case 2 in Section 2.

```

First-Timer : MUST ADD Prequalification BEFORE Preliminary Decision ON PAF
WHERE projectType="affordable"
  AND developerName NOT IN
  SELECT developerName
  FROM PAF P
WHERE P.artifactId <> SELF.artifactId
  AND P.projectType="affordable"

```

In this case, the rule named *First-Timer* corresponds to the new policy in CAPA that a building contractor who has never applied for affordable housing preselling project before should be qualified before *Preliminary Decision*. As shown in the example, the rule applies to the workflow with the core artifact PAF. The specific change is to add a task *Prequalification* before the task *Preliminary Decision*, when the WHERE-condition holds. A WHERE-condition is expressed as a formula in a nested SQL query. Specially, a keyword SELF is used to reflect the core artifact. Note that the condition is checked when a PAF artifact is retrieved into the place “started” during the execution of the task *Preliminary Decision*. Also, the word MUST indicates that this rule *must* be applied whenever the WHERE-condition is true (workflow engine’s decision).

We now incorporate modification rules into workflow schema to obtain “dynamically modifiable workflow”:

**Definition.** A *dynamically modifiable workflow (DEZ-Flow) schema* is a pair  $Z = (W, \mathcal{P})$ , where  $W$  is a workflow schema, and  $\mathcal{P}$  is a set of modification rules (with distinct names).

The semantics of dynamically modifiable workflow schema  $(W, \mathcal{P})$  can be formulated based on the discussions in Section 3 and the semantics of the operators. One important restriction is that at any time only one modification operation is allowed to be allied to one enactment. In other words, when one modification is ongoing, no other modification is allowed. However, one enactment can be modified sequentially.

Technically, we introduce a schema change operation. Let  $Z = (W, \mathcal{P})$  be a dynamic workflow schema,  $\mathcal{P}^+, \mathcal{P}^-$  be two sets of rules. The operation *modify*  $(Z, \mathcal{P}^+, \mathcal{P}^-)$  results in the dynamic schema  $Z' = (W, (\mathcal{P} \cup \mathcal{P}^+) - \mathcal{P}^-)$ . The following can be established by an induction:

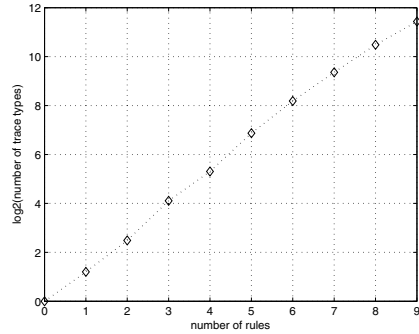
**Proposition 4** Let  $Z = (W, \mathcal{P})$  be a dynamic schema, and  $\mathcal{P}_1^+, \dots, \mathcal{P}_n^+, \mathcal{P}_1^-, \dots, \mathcal{P}_n^-$  two sequences of rule sets. Then,

$$\text{modify}(\dots(\text{modify}(Z, \mathcal{P}_1^+, \mathcal{P}_1^-), \dots, \mathcal{P}_n^+, \mathcal{P}_n^-) = \text{modify}(Z, \cup_{i=1}^n \mathcal{P}_i^+, \cup_{i=1}^n \mathcal{P}_i^-).$$

In [14], a set of correctness criteria were discussed, most of which state structural properties on schemas. In our framework, a dynamic schema combines procedural and declarative components. Understanding whether and how these and other correctness criteria would apply to dynamic schemas remains to be seen.

RULE NAME	CONSTRUCTS
Affordable-Fee	replace
Pre-Qualification	add
Affordable-Certificate	replace
Resettlement-Qualification	add
Greenchannel-PD	skip
Greenchannel-SR	skip
Greenchannel-FA	skip
Reapply-PD	retract
Reapply-SR	retract
Reapply-FA	retract

(a) Modification Rules



(b) Trace Types of CAPA

Fig. 10. Trace Types of CAPA with Different Number of Rules

## 5 Experimental Evaluation and a Case Study

In this section, we present results on flexibility and performance evaluation of DEZ-Flow and a case study to apply DEZ-Flow to the running system at REA.

Flexibility measures a workflow’s ability to adapt to complex situations through the use of different traces of execution. A *trace type* is a sequence of tasks (names) of a complete enactment (a complete lifecycle of a core artifact). We count the number of trace types permitted by a workflow to reflect its flexibility. The more flexible a workflow is, the more trace types it allows. In the experiment, the CAPA workflow in Section 2 is used along with 10 rules listed in Fig. 10(a). These rules are acquired from the actual business changes in the Hangzhou REA.

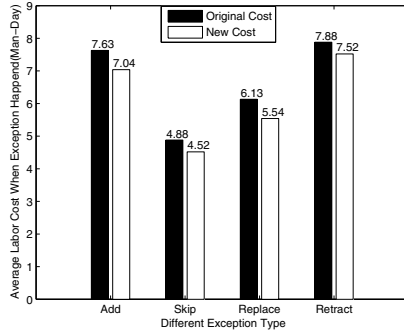
To calculate trace types of runs of a workflow, we count the number of different types of sequences delivered from tasks execution in a business process instance. To avoid infinite length trace types when a schema is cyclic, we limit the length of a meaningful run of a dynamic workflow to no more than twice the number of tasks in the original workflow. We count trace types for the original CAPA workflow and for CAPA dynamic schemas with different number of rules randomly selected from Fig. 10(a).

Fig. 10(b) shows the experiment results. Not surprisingly, the more execution modification rules are applied, the more trace types a CAPA workflow would allow, and the number increases almost exponentially. The results match the intuitive understanding of the rule impact by the staff members at REA.

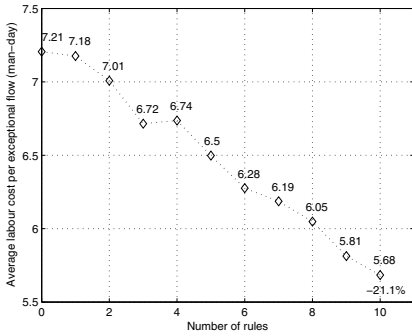
We now focus on performance of DEZ-Flow using CAPA as the original workflow schema. We set up an experiment in MatLab, where each task in a workflow is assigned a labor cost in *man-days* (top part of Fig. 11(a)). The costs are estimated actual costs in the REA business in Hangzhou. The total labor cost of a business process execution is the sum of the costs from the sequence of execution trace. When an unexpected event happens during workflow execution due to business policy changes, tasks are added or removed from the execution trace to reflect the change, and in addition, extra labor costs are required to handle the exceptions (e.g., manager consultation or interaction with IT staff). In actual situations, staff may have to work longer hours to handle these special cases manually. It can sometimes take more than a day when if the case

Task Name	Labor Cost (in man-days)
Receiving App-Form	0.5
Preliminary Decision	1.5
Secondary Review	1
Payment Processing	0.5
Final Approval	0.5
Preparing Certificate	1
Delivery Certificate	0.5
Affordable Payment Processing	0.5
Preparing Affordable Certificate	1
Affordable Pre-qualification	0.5
Resettlement Pre-qualification	0.5

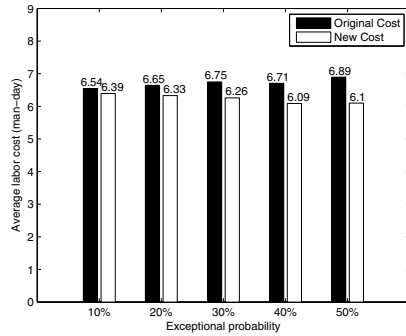
(a) Labor Cost of Tasks



(b) Trace Types of CAPA



(c) Labor Cost with Different Size of Rule Set



(d) Labor Saved with Different Exceptional Rates

**Fig. 11.** Labor Cost and Their Reduction with DEZ-Flow

involve the department head or staff from other departments. The additional labor cost for these situations is included in the additional tasks involved in the changes (bottom part of Fig. 11(a)). By deploying DEZ-Flow modification rules, business changes can be recognized and handled by applying skip, add, replace, retract actions. Thus execution trace is modified appropriately just as the original manual way, but the extra labor cost is minimized.

We classify all exceptional cases based on the main constructs (add, skip, replace, and retract) of the execution modification rules. Fig. 11(b) compares the original labor costs (Original Cost) and the cost by applying rules (New Cost). To simulate the actual working environment, we set up several rounds of runs of CAPA with different percentage of exceptional cases. Each round contains 200 runs, which simulates all CAPA cases dealt with by the Hangzhou REA in one year. The 10 types of exceptions may occurred in these runs with different probabilities.

Fig. 11(c) and (d) show the results of scalability evaluation. As shown in Fig. 11(c), when exception occurs, by adding more rules to the workflow the labor cost will linearly decrease. And the reduction is more than 20% if we have 10 rules in CAPA. Fig. 11(d) shows that as the exception probability grows, the average labor saved by using the DEZ-Flow approach also increases linearly. Fig. 11(c) and (d) only show the average



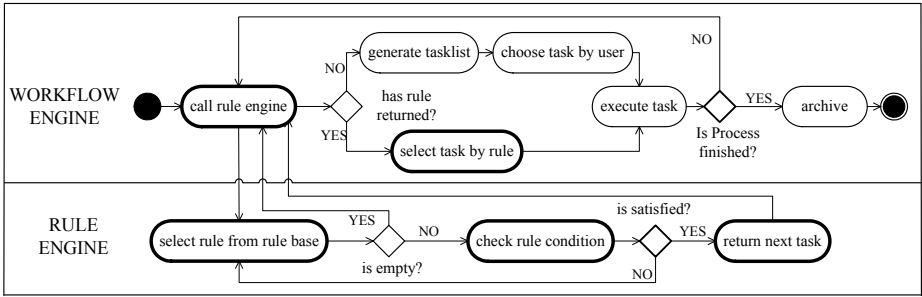


Fig. 12. Modify the Process Engine to Be a Rule-aware

cost for one single business process. In the Hangzhou REA, there are about 300,000 business cases every year and more than 35% of them are long duration cases similar to CAPA with costs 4-8 man-days per case. Therefore with 30% exception, if 10 rules are handled using the DEZ-Flow approach, about 38 man-year can be saved for the REA in Hangzhou. This would be a significant saving for a single bureau.

To validate the applicability of DEZ-Flow, we conducted a case study in the real business of REA in Hangzhou. With the help of the Hangzhou REA, we modified the current process engine in the bureau by introducing a rule-adaptation facility. As depicted in Fig. 12, it is realized by inserting the primitives in rule-handling task, in addition to the EZ-Flow facilities. Technically, business operators can insert or delete rules at any time if necessary. Before the invocation of any task, the task is evaluated first against rules. If any rules are applicable, the modified engine (rounded rectangles with thicker lines) will execute the rules, and decide the next task to be executed. Otherwise, the flow is executed as in the original plan. The main point here is that the next task is determined by the results of rule execution, rather than the task scheduled in the original workflow. As a result, the execution of a business process can be adjusted dynamically at the instance level in a just-in-time manner.

## 6 Related Work and Our Contributions

Process flexibility in business process modeling and management has been studied over a decade. Process flexibility is required (1) at design time when developer needs to have the choice of using procedural (structural) language or declarative language to model the process partially; (2) when execution decisions have to be made at run time; (3) when detailed specifications of some tasks cannot be fully defined at design time; (4) when business rules or policy change.

Substantial work has been done in the related areas in supporting process flexibility: (1) declarative workflow, (2) configurable workflow, (3) business process evolution.

Work in the area of declarative workflow can be classified as partially or fully declarative. Worklets is a partially declarative approach as a separate case in the workflow engine to execute a task [1]. A worklet is associated with a task, and worklets can form repertoire of tasks. A worklet is activated or substituted for another task at runtime. In [11][17] a fully declarative workflow system is developed. In [18], authors presented

the DECLARE framework to analyze the process support in terms of verification and performance. It uses arbitrary constraint templates as the declarative language to construct a workflow. The flexibility it supports are “defer”, “change”, and “deviate”. It is not clear how performance is analyzed in comparison with imperative approaches. [12] proposed an object-centric modeling approach for supporting flexible business process modeling. CPN is used for the syntax and semantics of the proposed FlexConnect model. It can support flexibility in terms of creation, delegation, and nesting flexibility.

Work in the area of configurable workflow for variability support and configuration in workflow mainly focused on (1) providing variant points on the process which can be activated, blocked, hidden; (2) verifying configurability of the model [4].

In the area of business process evolution, work has been done in identifying change patterns and change support [16], extensibility support in business process life cycle management [3], and variants support in [5]. In [16], a set of change patterns are defined and the corresponding change operations for the patterns are developed. Authors in [3] address the issues of customizing reference processes in which some extension points are defined. Reference [5] proposes an approach for storing change operations that need to be applied for a process variant to be derive from a base process.

The above work has provided insights in process change handling and verification techniques that can be used as a foundation for the proposed work. What we focus here, however, is a framework for handling runtime changes and providing just-in-time solution. Our main contributions are:

- An artifact-centric workflow model that explicitly captures the runtime status using snapshots. The snapshots hold the key information that the workflow scheduler and task performer can rely on to determine the suitable execution at the time.
- A hybrid language for workflow specification. The base workflow schema is procedural while the flexible points leading to deviations are specified in a declarative fashion. For the identified points where changes can occur, we can specify rules to stay flexible.
- Separating the workflow schema from the plug-in rules to cater for ad hoc changes. When a change occurs, only the rules need to be changed, or new rules added. The base workflow schema remains unchanged, while the changes only affect the running instances at the point when a rule becomes effective. Since the base schema is untouched and the rules can be added and discarded, the approach provides a holistic and less intrusive way of introducing changes to a running workflow.

## 7 Conclusions

Supporting workflow flexibility is crucial but challenging. We believe new workflow models and mechanisms need to be provided to tackle the challenge, especially for runtime ad hoc execution modifications. In this paper we proposed an approach for just-in-time modification at runtime based on an artifact-centric workflow model. The workflow model DEZ-Flow, in particular the scheduler and flexible task performer are specified. Effectiveness and flexibility of DEZ-Flow are analyzed for a real business case in a Real Estate Administration in China.

## References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
2. Athena, P.: Flower user manual. Technical report, Pallas Athena BV, Apeldoorn, The Netherlands (2002)
3. Balko, S., ter Hofstede, A.H.M., Barros, A.P., La Rosa, M.: Controlled flexibility and life-cycle management of business processes through extensibility. In: EMISA 2009, pp. 97–110 (2009)
4. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., La Rosa, M.: Configurable workflow models. *Int. J. Cooperative Inf. Syst.* 17(2), 177–221 (2008)
5. Hallerbach, A., Bauer, T., Reichert, M.: Managing business process variants in the process lifecycle. In: Proc. ICEIS (2008)
6. 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-stagemilestone approach to specifying business entity lifecycles. In: Proc. Workshop on Web Services and Formal Methods (WS-FM). Springer, Heidelberg (2010)
7. Kucukoguz, E., Su, J.: On lifecycle constraints of artifact-centric workflows. In: Proc. Workshop on Web Services and Formal Methods, WSFM (2010)
8. Liu, G., Liu, X., Qin, H., Su, J., Yan, Z., Zhang, L.: Automated realization of business workflow specification. In: Proc. Int. Workshop on SOA, Globalization, People, and Work (2009)
9. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. *IBM Systems Journal* 42(3) (2003)
10. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for looselystructured processes. In: EDOC 2007, pp. 287–300 (2007)
11. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
12. Redding, G., Dumas, M.: A flexible object-centric approach for business process modeling. *SOCA* 4, 191–201 (2010)
13. Reichert, M., Dadam, P.: Adept<sub>flex</sub>-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.* 10(2), 93–129 (1998)
14. Rinderle, S.B., Reichert, M.U., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. *Data & Knowledge Engineering* 50(1), 9–34 (2004)
15. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process flexibility: A survey of contemporary approaches. In: CIAO! / EOMAS 2008, pp. 16–30 (2008)
16. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
17. van der Aalst, W.M.P., Pesic, M.: Decserflow: towards a truly declarative service flow language. In: International Conference on Web Service and Formal Methods (2006)
18. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. *CSRD* 23, 99–113 (2009)
19. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features enhancing flexibility in process-aware information systems. *Data Knowl. Eng.* 66(3), 438–466 (2008)
20. Weber, B., Sadiq, S.W., Reichert, M.: Beyond rigidity - dynamic process lifecycle support. *Computer Science - R&D* 23(2), 47–65 (2009)

# Event Cube: Another Perspective on Business Processes

J.T.S. Ribeiro and A.J.M.M. Weijters

School of Industrial Engineering, Eindhoven University of Technology  
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands  
{j.t.s.ribeiro,a.j.m.m.weijters}@tue.nl

**Abstract.** In this paper the so-called *Event Cube* is introduced, a multi-dimensional data structure that can hold information about all business dimensions. Like the data cubes of online analytic processing (OLAP) systems, the Event Cube can be used to improve the business analysis quality by providing immediate results under different levels of abstraction. An exploratory analysis of the application of process mining on multidimensional process data is the focus of this paper. The feasibility and potential of this approach is demonstrated through some practical examples.

**Keywords:** process mining, process discovery, OLAP, data mining.

## 1 Introduction

Business process intelligence (BPI) techniques such as process mining can be applied to get strategic insight into the business processes. Process discovery, conformance checking and performance analysis are possible applications for knowledge discovery on process data [1].

Typically represented in event logs, business process data describe the execution of the different process events along the time. This means that operational data are associated with process events, which turns the static nature of the business data into dynamic. This is a great advantage for business process analysis once that the business behavior can be tracked. Applying process mining techniques on the sequences of events that are used to describe the behavior of process instances, it is possible to discover the business as it is being executed. However, so far, these techniques are typically designed to focus on specific process dimensions, omitting information potentially relevant for the analysis comprehension. An illustrative example demonstrates this observation. Let's assume that there is an event log where the execution information of a product repair process is registered. Basically, this process is defined by 7 activities in which 12 different resources (divided by 3 categories) attempt to fix 2 different kinds of products. Figure 1 presents the behavior of this process, which is the result of the application of a control-flow mining technique on the event log. In these cases the only dimension taken into account is the *activity*.

Process models, such as the one depicted in Figure 1, have proven to be effective for process discovery but they can only provide an abstract view of the actual business process. Basically, two distinct aspects are considered: the events (nodes)

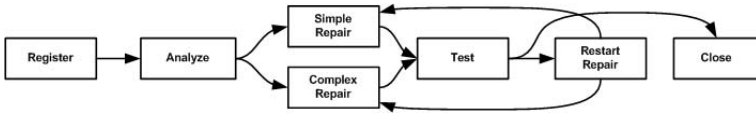


Fig. 1. Repair process

and the workflow (edges). The events identify *what the executed activities are*. On the other hand, the workflow defines *how the events were executed*. Other kinds of queries such as *who performed the events*, *when the events were executed*, *which objects* (e.g., products) are associated with the events (or the process instances), and *why a specific process behavior happens* are not directly addressed in such a basic approach. Nevertheless, some filtering operations or other process mining techniques may achieve that, though in a non-integrated way.

Specially designed to support on-the-fly hypothesis-driven exploration of data, OLAP systems are commonly used as reporting tools in almost every application for business intelligence. Exploiting the data by combining the different dimensions with some measures of interest, it is possible to adjust on-the-fly the analysis' level of abstraction in an interactive way [2]. Relying on the multidimensional data model, OLAP tools organize the data in such a way that it is possible to have multiple perspectives in the same analysis. Considering the event log's attributes as dimensions, process models can be built in such a way that events and the workflow can be constrained by specific process information. Traditionally, only a single constraint is considered: the *activity* as events constraint. In a multidimensional approach, process models can be constrained by the dimensions the analyst considers relevant. An example to show the potential of this approach is given in Figure 2. Making use of event-based attributes such as the *product type* as extra events constraint (besides the *activity*), it is possible to analyze different process behaviors in a single model. An alternative perspective in which the *product type* is used as workflow constraint is presented in Figure 3. The edges represented by solid lines refer the *Product X*, while the dotted lines refer the *Product Y*. Both models are examples of a conjunction of *what*, *which* and *how* queries.

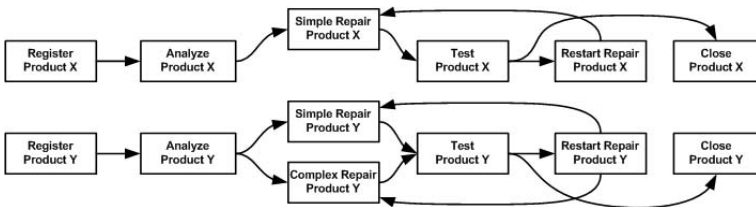
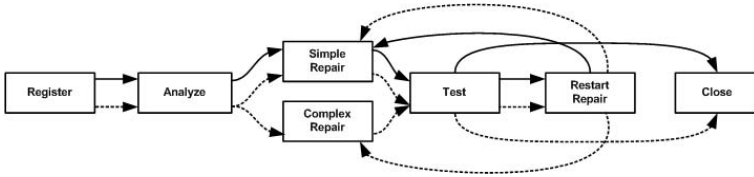


Fig. 2. Repair process by product type (as events constraint)

In this paper the Event Cube is presented, a multidimensional approach for process discovery and analysis. Due to its multidimensional nature, the Event Cube may cover a broader range of queries than any other process mining technique. This means that different aspects of the business process can be exploited in an integrated multidimensional analysis. Rather than providing implementation



**Fig. 3.** Repair process by product type (as workflow constraint)

details, this paper consists of an exploratory study of the potential and feasibility of the Event Cube approach, which is implemented in the ProM framework [12]. A further discussion about the concept implementation and evaluation is planned to be presented in a following paper.

The remainder of this paper is organized as follows. In Section 2 the multidimensional data model as well as OLAP concepts are introduced. Section 3 identifies the differences between traditional and multidimensional process models. Section 4 describes the Event Cube and its components. The results of a preliminary experimental study on the Event cube are presented in Section 5. Related work and conclusions are discussed in sections 6 and 7.

## 2 Multidimensional Data Model

Mainly used in OLAP, the multidimensional model represents data by means of a multidimensional *fact*-based structure that supports complex queries in real time. A fact describes a business operation (e.g., sale or purchase), which can be quantified by one or more *measures* of interest (e.g., the total amount of the sale) and characterized by multiple *dimensions* of analysis (e.g., the time, location, product and customer). Typically numerical, measures can be aggregated for different levels of abstraction. Each dimension consists of a set of discrete values called *dimension values* or members. Eventually, in the same dimension, there may be dimension values that represent different concept levels. For these cases, a *hierarchy* defines the order the different dimension values should be exploited, from a lower to a higher concept level. The typical example of a hierarchy for the *time* dimension based on the attributes *day*, *month*, and *year* is “*day* < *month* < *year*”.

Also designated as hypercube, the *data cube* provides a multidimensional view of data (i.e., facts) through the materialization of given measures for every combination of the cube’s dimension values. Each combination defines a different perspective and is represented by a *multidimensional cell*. A cell consists of a pair with a set of dimension values that identifies univocally the cell, and a set of aggregated values representing the measures. The set of cells that share the same dimensions forms a *cuboid*. The complete set of cuboids forms the data cube through a *lattice of cuboids*.

A cell is materialized with respect to one of its measures when all the values of the cell’s representative facts – for that measure – are aggregated according to a given aggregation function. Additionally, it is said that a cuboid is materialized when all of its cells are materialized. The same principle applies to the data cube and its cuboids.

The multidimensional analysis of a data cube consists of the exploitation of its cuboids. Moving through the lattice of cuboids, the analyst is able to adjust the analysis perspective by selecting the cube dimensions. There are five typical OLAP operators that can be used for querying multidimensional data.

**Drill-Down** descends one level in the lattice by adding one dimension (or hierarchy level) to the current perspective (i.e., decreases the level of abstraction).

**Roll-Up** ascends one level in the lattice by removing one dimension (or hierarchy level) from the current perspective (i.e., increases the level of abstraction).

**Slice and Dice** restricts the perspective by using filtering conditions.

**Pivot** permutes the analysis' axes. The perspective remains the same but the information is given in a different layout.

**Top-k Selection** restricts the perspective to the top-k cells of a given measure.

Figure 4 exemplifies the result of drilling-down the Figure 1's one-dimensional model on the dimension *time period*. Adding the *when* query to the *what* and *how* queries, it is possible to visualize that there may exist bottlenecks in the repair process. This conclusion can be drawn because there is a high amount of incoming edges from different time periods in both simple and complex repairs. Eventually, adding more dimensions to the model, it is possible to determine the causes of specific process behaviors. Note that the different edge colors (gray and black) are used simply to distinguish the interdependencies between time periods.

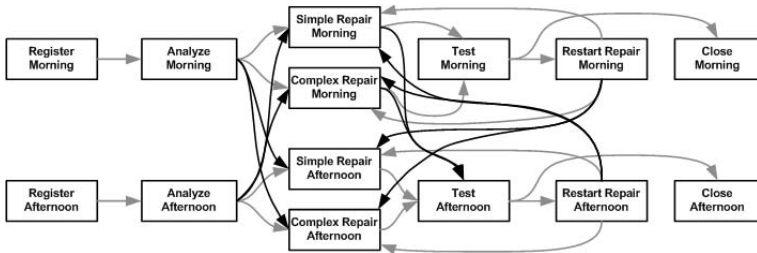


Fig. 4. Repair process by time period

Figure 5 provides an example where the slice and dice operation is applied. By selecting the activities *Test*, *Restart Repair* and *Close*, and considering the instance-based attribute *repair status* as workflow constraint, it is possible to analyze some of the behavior of the repair process. In order to understand this example, remark that the activity *Test* tests whether the – last – repair was successful or not, updating the repair status with a “OK” or “Not OK” tag. The history of test results (for the same process instance) is kept in this attribute. So, multiple tags provide insight into unsuccessful repair attempts. More than that, this process perspective clearly shows that a repair case is closed if the repair was successful (i.e., there is the tag “OK” in the repair status) or at the end of the third unsuccessful repair attempt. So, this case is a good example of a *why* query.

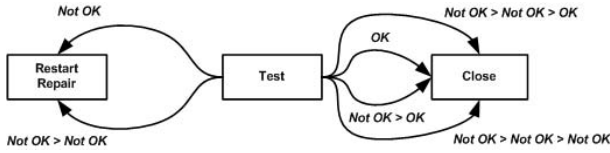


Fig. 5. Partial repair process by repair status

### 3 Process Models

A – traditional – process model can be defined as the set of activities and their dependency relationships (workflow) during a process. Figure 1 is an example of a traditional model in which the activities and the workflow are represented by nodes and edges. Typically, these models can be derived from an event log that contains information about the activities such as the activity designation, the execution date, or the resource that executed the activity. Hence, event-related information such as resources or time is often abstracted from the model. The relationships define how the activities were performed. Depending on the representation language, complex relationships such as splits and joins may be explicitly considered. Examples of these languages can be found in [10].

**Multidimensional Process Models** can be defined as the set of *event occurrences* and their relationships (workflow) during a process. It is considered as – event – occurrence a possible combination of dimension values from distinct dimensions (e.g.,  $\{Test, Product X\}$ ). Additionally, unlike traditional models, the workflow in multidimensional process models may also be constrained by multiple dimension values. The result of this extension is that process models are no longer restricted to *what* and *how* queries. In theory, depending on the available data, any type of query may be answered using these models.

Figures 2 and 3 are simple examples of multidimensional process models. In the first case, there is no workflow constraint and the event occurrences are defined by two dimensions: *activity* and *product type*. In the other case, the *product type* is used as workflow constraint and the event occurrences are defined only by the *activity*. Remark that event occurrences may not be directly associated with *what* queries as it always happens with traditional process models. A good example of these cases is presented in Figure 6. Defining the one-dimensional event occurrences as *resource category*, it is possible to analyze the *handover of work* in the process by executing both *who* and *how* types of queries.

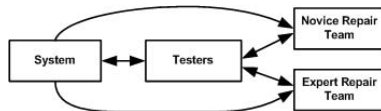


Fig. 6. Handover of work in the repair process



Depending on the cardinality of each dimension, the number of event occurrences may be significant enough to turn the process model unreadable. Known as *curse of dimensionality*, this issue is even more evident if the relationships between occurrences are complex and unpredictable. One possible solution to minimize the impact of the dimensionality is the usage of simplified model representations such as the ones that can be used to represent causal nets (C-Nets) [131]. Simply using two different kinds of elements to represent C-Nets (nodes and edges), it is possible to provide a clear picture of the process without omitting information. Complex relationships between activities (i.e., the splits and joins patterns) are provided separately. A similar approach can be applied for representing multidimensional process models.

## 4 Event Cube

As indicated before, an Event Cube is defined as a data cube of events and can be used to extend the current process mining techniques with multidimensional capabilities. By materializing a selection of dimensions of interest, an Event Cube can be built to accommodate all the necessary measurements to perform process mining. This means that, all the information regarding the different perspectives (and levels of abstraction) is computed and maintained in the cube, facilitating thus the execution of all types of queries.

The basic implementation of an Event Cube is depicted in Figure 7 and can be characterized as follows. Relying on an index instead of directly on the event log, the lattice of cuboids is firstly built according to a given set of dimensions. Representing a different perspective, each of these cuboids holds the event occurrences that define a given perspective. These occurrences are represented as multidimensional cells and can be characterized by multiple measures. The Event Cube materialization process is finished when all the measures of all the multidimensional cells are computed. Then, the materialized measures can be either directly analyzed (e.g., using a pivot table) or used for process mining (e.g., deriving multidimensional process models).

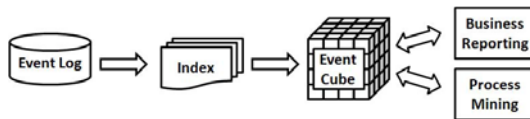


Fig. 7. The Event Cube approach architecture

**Indexation** consists of the application of information retrieval techniques on event logs. Commonly used in document retrieval systems, *inverted indices* are capable to index multidimensional datasets in such a way that any dimension value can be directly accessed. Basically, these indices rely on the locations in the dataset where the dimension values appear. By applying the inverted indexing concept on event logs, it is possible to directly retrieve all the process events according to given constraints. Since event logs organize the process data as sets

of traces (each trace is a set of events), the pair of identifiers (trace ID, event ID) can be used to index the different event dimensions. The intersection of sets of identifiers determines the locations of the events (trace and trace position) that are characterized by specific dimension values. Further details about inverted indices can be found in [5].

**Materialization** consists of the computation of the business process-related measures. For reporting purposes, *final measures* (e.g., the *Throughput Time*) can be either retrieved directly from the event log or derived using some given function. For process mining purposes (e.g., process discovery), *auxiliary measures* can be maintained in order to facilitate the execution of the process mining techniques. An example of these process mining-related measures is the Flexible Heuristics Miner’s dependency measures [13] that can be straightforwardly used to derive multidimensional process models. Basically, this control-flow mining algorithm derives the input and output activities of a given activity by using simple frequency-based measurements and applying thresholds. Considering multidimensional cells (i.e., event occurrences) instead of activities, the algorithm can be normally executed on the cuboids’ cells in order to build the multidimensional process models for the different perspectives.

**Analysis** consists of the exploitation of the Event Cube information. Applying the typical OLAP operators on the Event Cube, it is possible to adjust the analysis’ process perspective by selecting the cube dimensions. This means that the analyst is able to explore the measures or derive process models under different abstraction levels. One of the major claims in this paper is that process models can be built using multiple dimensions. In the traditional process models (e.g., C-Nets) only one dimension is considered: the *activity*. Nodes represent the executed activities and edges refer the workflow. Figure 1 shows an example of these models. So, it can be concluded that traditional process models are defined by one-dimensional nodes and zero-dimensional edges. Remark that these models can be represented by an one-dimensional cuboid on the dimension *activity*. The number of dimensions of multidimensional process models is defined by the cuboid dimensionality. So, all of the cuboid’s dimensions need to be selected either as event occurrences or workflow constraints. This implies that two multidimensional models on the same cuboid (i.e., representing the same perspective) may adopt very distinct representations. Figures 2 and 3 present two different representations of the same multidimensional model. In the first case the occurrences are formed by the dimensions *activity* and *product type* and there are no workflow constraints. In the second case the occurrences are formed exclusively by the dimension *activity*, while the workflow is constrained by product type.

## 5 Experiments

A preliminary implementation of the Event Cube approach was done in Java as a ProM 6 plugin [12]. Basically, the implementation consists of an inverted

index, an Event Cube data structure, a multidimensional version of the Flexible Heuristics Miner (FHM), and a graphical user interface for OLAP-based process analysis. In order to evaluate the scalability of the approach, several Event Cube instances (with the FHM measurements) as well as multidimensional process models were computed using different event logs. Considering the computing time as the main performance indicator, this evaluation study aims at the analysis of the impact of three main variables: (i) the size of the event log, (ii) the Event Cube's dimensionality, and (iii) the dimensions' cardinality.

All the event logs used in these experiments were generated from the repair process introduced in Section II. Varying the number of resources as well as products, 4 event logs with 1000 process instances are used to evaluate the impact of the dimensions' cardinality in the Event Cube's materialization. In order to keep the process consistency, the existing resources and products are simply multiplied (cloned). Figure 8c presents the results of this evaluation. The line represents the materialization time (primary axis) for the different cardinality setups, while the bars refer the number of materialized cells (secondary axis). In the event log identifiers (x-Axis), *A* refers the activities, *P* the products and *R* the resources.

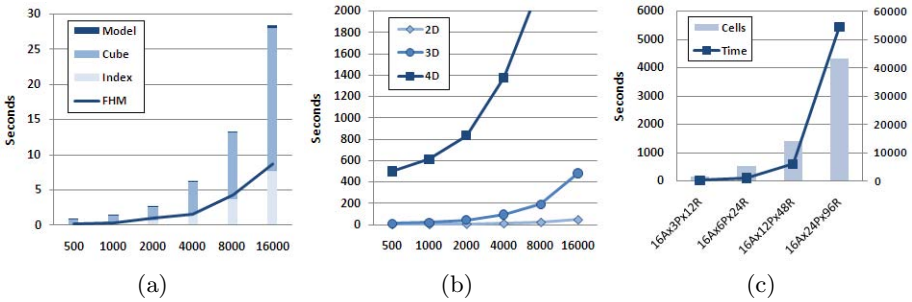


Fig. 8. Performance results

Using again the same characteristics of the repair process, 6 event logs with different number of process instances are used to evaluate the impact of the event log size in the Event Cube's materialization – as well as the multidimensional model computation. With respect to the number of process instances (x-Axis), Figure 8a compares the computing time evolution of building the inverted index, materializing the Event Cube, and computing a one-dimensional model equivalent to a traditional one. Additionally, the computing time of the original FHM (i.e., deriving the C-Net directly from the event log) is also presented for comparison. Figure 8b presents the evolution of the computing time for building multidimensional Event Cubes and multidimensional models (y-Axis) with respect to the number of process instances (x-Axis). All the experiments were run on Intel Core 2 Quad Q9650 3.0GHz with 4Gb on Microsoft Windows XP OS.

An interesting observation in these experiments is the curse of dimensionality issue. Figure 8c confirms that the dimensions' cardinalities have direct impact on the number of cells and, consequently, on the performance. From a different point

of view, Figure 8b shows that cube's dimensionality is a bigger issue than the event log size. As expected, like in traditional OLAP applications, the cube materialization is the most demanding process. Nonetheless, this is not considered a critical issue once that the Event Cube materialization should be a single-run process (i.e., cube instances can be reused at different points in time). The results also confirm that deriving a multidimensional model from an Event Cube is almost immediate. Considering only the time for deriving the process model, the Event Cube clearly outstands the traditional FHM approach (Figure 8a). Taking into account that the experiments were conducted on a preliminary implementation (i.e., there is still room for improvements such as the use of multithreading and efficient materialization strategies), it can be concluded that the experiment results are promising. However, although the results suggest the feasibility of the approach, demonstrating it requires additional experiments.

## 6 Related Work

Traditionally associated to decision support systems (DSS), OLAP systems provide a different view of the data by applying the multidimensional paradigm [2]. OLAP techniques organize the data under multiple combinations of dimensions and, typically, numerical measures. A lot of research have been done to deal with OLAP technical issues such as the materialization process. An extensive overview of these works can be found in [5]. The application of OLAP on non-numerical data is increasingly being explored. Temporal series, graphs, and complex event sequences are possible applications [6,3,7].

Process mining techniques provide strategic insight into the business processes. Process discovery, conformance checking and performance analysis are possible applications [11,4,10]. However, the state-of-the-art process mining techniques do not support yet multidimensional analysis. Nevertheless, some research has been done in multidimensional process analysis [9,8]. Workflow ART is a proposed framework to explore the three main business process dimensions *action* (or activity), *resource* and *time*. Although several predefined business measures can be analyzed, this approach is not flexible as an OLAP-based technique. A fully multidimensional approach for business process analysis can be found in [8]. Relying in the multidimensional data model, this approach maps the different aspects of the business process into a data cube. However, only numerical information is considered.

## 7 Conclusions

This paper is about an exploratory analysis of the application of process mining on multidimensional process data. Rather than the concept implementation details, it is intended to demonstrate the potential and feasibility of the Event Cube approach. Several examples are given to provide some insight into the different types of business queries the proposed approach can execute. The Event Cube (i.e., a data cube of events) is defined to support these multidimensional

analyses, which can either be done directly on the cube or using the cube as basis for process mining techniques such as the Flexible Heuristics Miner. Implementing the main OLAP operators, the proposed framework provides to the business analyst the opportunity to drill-down the process information without complex queries and long response times. A preliminary experiment analysis suggests that the Event Cube approach is feasible. However, further experiments using a revised implementation and real datasets are still necessary.

**Acknowledgements.** This work is being carried out as part of the project “Merging of Incoherent Field Feedback Data into Prioritized Design Information (DataFusion)”, sponsored by the Dutch Ministry of Economic Affairs, Agriculture and Innovation under the IOP IPCR program.

## References

1. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards Robust Conformance Checking. In: Proceedings of the 6th Workshop on Business Process Intelligence, BPI 2010 (2010)
2. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Rec. 26, 65–74 (1997)
3. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S.: Graph OLAP: a multi-dimensional framework for graph data analysis. Knowl. Inf. Syst. 21, 41–63 (2009)
4. Gunther, C.W.: Process Mining in Flexible Environments. PhD thesis, Eindhoven University of Technology, Eindhoven (2009)
5. Han, J., Kamber, M.: Data mining: concepts and techniques. The Morgan Kaufmann series in data management systems. Elsevier (2006)
6. Li, X., Han, J.: Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007, pp. 447–458. VLDB Endowment (2007)
7. Liu, M., Rundensteiner, E., Greenfield, K., Gupta, C., Wang, S., Ari, I., Mehta, A.: E-Cube: Multi-dimensional event sequence processing using concept and pattern hierarchies. In: International Conference on Data Engineering, pp. 1097–1100 (2010)
8. Mansmann, S., Neumuth, T., Scholl, M.H.: Multidimensional data modeling for business process analysis. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 23–38. Springer, Heidelberg (2007)
9. Monakova, G., Leymann, F.: Workflow ART. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 376–393. Springer, Heidelberg (2010)
10. Rozinat, A.: Process Mining: Conformance and Extension. PhD thesis, Eindhoven University of Technology, Eindhoven (2010)
11. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. 47, 237–267 (2003)
12. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM Framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
13. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011. IEEE, Paris (2011)

# Building eCommerce Systems from Shared Micro-schemas

Stefania Leone and Moira C. Norrie

Institute for Information Systems, ETH Zurich  
CH-8092 Zurich, Switzerland  
{leone,norrie}@inf.ethz.ch

**Abstract.** We present an approach that supports the design of eCommerce systems in a modular way based on the composition of micro-schemas that are shared and reused within a community. We have designed and developed a platform that supports platform users in composing their individual eCommerce systems through a process of selecting appropriate micro-schemas from the micro-schema repository, composing, extending and adapting them at the level of the application model.

**Keywords:** database design, model reuse.

## 1 Introduction

eCommerce platforms, such as osCommerce<sup>1</sup> and Magento<sup>2</sup> are widely used by small companies for their online businesses. Such platforms support the complete range of tasks and processes associated with eCommerce, ranging from the management of products, customers and orders to visitor statistics and advertisements. However, while such platforms provide fully-fledged online store functionality, their business model might not match that of a particular company. For example, a company might sell virtual products, such as digital coupons or digital photos, that are not physically shipped and thus need a shipment mechanism for digital products. Another company might sell products with a very particular structure and composition that is not supported by the platform. While these platforms generally do cater for extensibility and the community is free to develop and share extensions, their monolithic core is rather static. Extensions and adaptations on the level of the data model result in code extensions or adaptation of the corresponding application logic and user interface components, which may involve significant development effort as well as altering and manipulating the system core. Furthermore, new releases of the core require adapted and extended core versions to be manually updated – file by file.

An ideal eCommerce platform would support a company in the composition of an online store in a flexible and lightweight way by means of reusable, standard building blocks that optimally accommodate their specific requirements. A user

---

<sup>1</sup> <http://www.oscommerce.com/>

<sup>2</sup> <http://www.magentocommerce.com/>

should be able to select from a collection of variants of standard building blocks which can be adapted and extended if required. Users could then share their customised versions with the community.

In this paper, we present an approach that supports the creation of such systems through the reuse and composition of shared micro-schemas that can be combined, adapted and extended. We present an architecture and eCommerce platform that supports the plug-n-play composition of online stores from standard micro-schemas as well as from ones created and adapted by the community.

We start in Sect. 2 with a discussion of the background of this work. We then present our approach and architecture in Sect. 3, explaining how it allows users to construct a system in a modular fashion based on the reuse and composition of micro-schemas. Details of how micro-schemas can be composed are given in Sect. 4. In Sect. 5, we present the eCommerce platform and a discovery and inspection system for share micro-schemas is described in Sect. 6. Information on the implementation is given in Sect. 7. A discussion of open issues is given in Sect. 8 followed by concluding remarks in Sect. 9.

## 2 Background

While enterprise information systems and the databases that they use are designed and developed by expert teams, small businesses and individual users typically do not have the resources and expertise to create solutions tailored to their requirements and have to rely on off-the-shelf solutions. Researchers have therefore investigated ways of allowing non-expert users to design their own information systems through processes of reuse, specialisation and composition.

A common approach is to adopt a service-oriented model that allows users to build applications through the orchestration of reusable, self-contained services [1]. In the realm of the web, so-called mashup tools have become popular as a means of developing applications through the composition of web services and a number of these offer endusers graphical tools to do the composition process [2,3] as an alternative to programmatic interfaces. Web services interact at the message level and may span multiple applications and organisations. They are shared and reused by publishing them to a repository where consumers can search for and create the necessary bindings to use them.

Services define and manage their own data for the service that they provide. However, users may have their own data management requirements and want to design their information spaces accordingly. Sharing and reuse at the database design level is a valuable means of supporting non-expert users in this task. One approach is to provide domain-specific solutions based on general domain models. In Galaxy [4], they assume that communities working in the same domain agree on general core schemas and allow community members to further extend and customise these schemas according to their specific needs. They call this a core/corona design pattern, where coronas are individual, but related, extensions. Developers design their schema by extending the core with new entities and by reusing existing entities, which can be searched and browsed using a

visual schema search engine [5]. In [6], they propose an alternative approach that allows a general domain model to be decomposed into so-called concept schemas representing specific entities and all of their related entities, as well as generalisation-, aggregation- or instance-of hierarchies, which can be refined.

Others have focused on conceptual model repositories for specific domains such as medicine [7] and ecology [8]. A general repository for the reuse of ER models, with reusable elements extracted from groups of similar conceptual models in terms of generic entities and meta-entities, resulting in a kind of design pattern for a specific application domain has also been proposed [9]. An approach to sharing conceptual models based on so-called reference models that are collaboratively created by the community is presented in [10].

While model repositories support reuse, there is no support for the actual design process in terms of a platform where the system design can be carried out. This has been addressed by a second class of approaches which support modular database design based on ideas of component-based systems. In [11], a methodology for modular database design is based on the notion of modules that encapsulate a set of relations together with operations and integrity constraints. Users can create *subsumption modules* that reuse one or more existing modules as well as optionally creating additional relations, operations and integrity constraints. Similarly, in [12], they propose the use of composable sub-schemas and other meta-structures to support the modelling and management of large and complex database schemas. Their approach is based on the observation that large schemas are often organised in connected components with similar structuring. Such components are database schemas that have an import and an export interface for connecting them to other components through explicit connector types. These solutions focus on the composition of schema components, but do not deal with support for the sharing and reuse of these components.

Community-driven development has reformed the areas of software development and content authoring, turning consumers into producers and allowing endusers to profit from the experience and efforts of their peers. Such systems typically evolve around an extensible kernel developed in a traditional setting with the community developing the periphery [13]. The kernel provides the basic mechanisms required to support sharing and reuse of extensions, with the periphery representing a continuously evolving space of alternative, overlapping and possibly even conflicting solutions. While platforms that follow these design principles offer a flexible approach to system development based on extensions of the core, these extensions tend to be kept separate and there is no composition at the level of the data management. We aim at overcoming this gap and applying the ideas of community-driven development to the domain of eCommerce to offer a flexible eCommerce system design process, where the system composition takes place at the level of the data model. We aim for an approach where, instead of having a fully-fledged eCommerce core, an eCommerce solution provider offers a platform together with an extensible kernel and system building blocks from which individual eCommerce systems can be composed based on a user's individual requirements. Importantly, in accordance with crowdsourcing



principles [13], the aim is not to have consistent or complete domain models, but rather to provide maximum flexibility where conflicting requirements may be accommodated. Users should be able to freely select and import such building blocks which can then be composed, adapted and extended as required. They should be able to register for notifications about changes to a building block that they have imported, including the production of new revisions or even its removal, however they would be under no obligation to act on these.

In the following sections, we first present our approach and architecture before going on to detail the eCommerce platform and its implementation.

### 3 Approach

We will start with an example that illustrates how we envision system construction based on the composition of micro-schemas, followed by an overview of the platform architecture to illustrate how our approach works in practice.

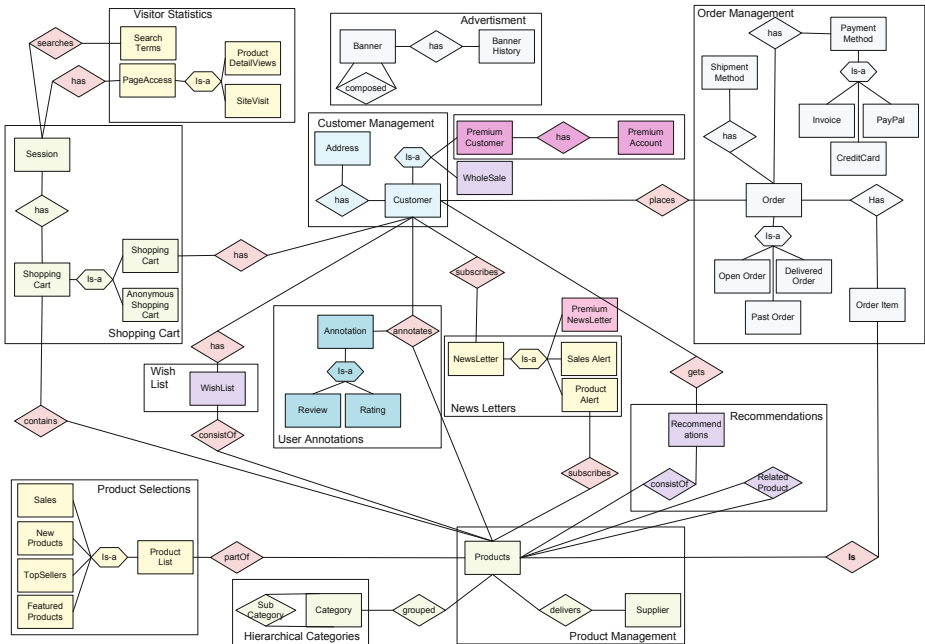


Fig. 1. Example of a composed online store

Figure 1 gives an overview of an ER model for an online store inspired by the osCommerce application model. We have componentised the application model into micro-schemas which are represented by rectangular boxes around schema elements that each address a specific information management requirement. In the lower part of the figure, micro-schemas for product management, hierarchical

product categorisation and featured product groups are represented. The management of customers is supported by a customer management micro-schema shown in the upper centre of the figure, that has been extended with a micro-schema for premium customer management. The shopping cart functionality is provided by the shopping cart micro-schema on the left, while the order management micro-schema on the right supports the ordering, shipment and payment process. Furthermore, the micro-schemas in the centre of the figure provide data structures to manage product ratings and reviews, customer wish lists, newsletter subscriptions as well as product recommendations. Finally, in the upper left corner, there are micro-schemas for the management of advertisements as well as visitor statistics. Note that the complexity of micro-schemas may range from single entities, such as in the case of the wishlist micro-schema, to rather complex application models such as the order management micro-schema.

As illustrated in Fig 1, micro-schemas are related to other micro-schemas to build a more complex schemas. For micro-schema composition, we rely on schema elements inherent to the modelling language. This is illustrated in Fig. 2. On the left of the figure, the customer schema is extended with a micro-schema for managing premium customers through an *isA* relationship that reflects the fact that premium customer is a specialisation of customer. On the right of the figure, the customer micro-schema is associated to an order micro-schema via a *places* association that allows customers to place orders.

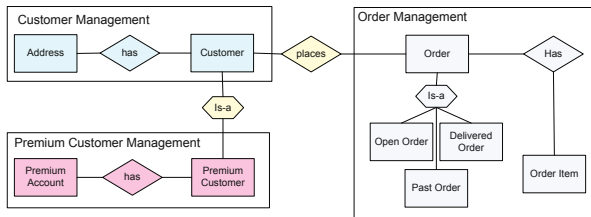


Fig. 2. Composition Example

The system composition is based on an eCommerce platform that supports these operations. Figure 3 gives an overview of the platform architecture. The top right corner shows the eCommerce platform which would be hosted by the platform provider and offer customers an eCommerce software solution. Customers can download the kernel system from the platform to obtain their local installation which offers the functionality of composing and combining micro-schemas to construct a customised eCommerce platform.

A platform user can design their eCommerce system through a process of adapting, extending and composing micro-schemas available in the repository of the eCommerce platform. The micro-schema repository manages the collection of micro-schemas that are available for that specific platform and orchestrates their sharing and reuse. Shared micro-schemas are registered with the micro-schema repository with a unique name and a detailed description. The platform provider offers a number of standard micro-schemas that provide general eCommerce

functionality and can be reused, composed, extended and adapted by the users to build their own eCommerce system. Furthermore, the user community is free to produce their own micro-schemas or variants of standard micro-schemas and to share them over the micro-schema repository. For example, a user could design a micro-schema that extends the eCommerce system with the management of customer feedback and complaints and share it with the community.

Our system supports the management of different versions of the same micro-schema as well as revisions of these. A user might for example extend a standard micro-schema and share it as a variant of the original one, thus supporting collaborative micro-schema design. Also, they may provide revisions of already shared micro-schemas.

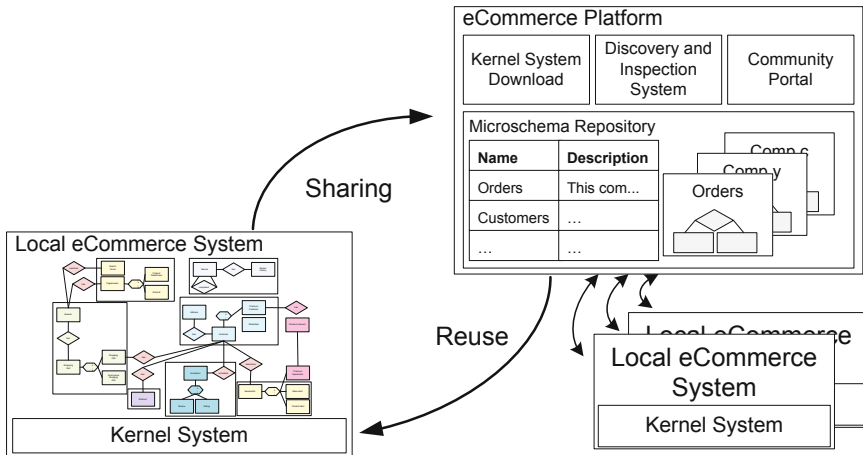


Fig. 3. Architecture

The micro-schema repository can be accessed by users using a discovery and inspection system that supports the users in finding appropriate micro-schemas to be used for their local design.

## 4 Micro-schema Composition

A micro-schema caters for a specific information management task by means of data structures and application logic and acts as the unit of reuse and composition in system design. A micro-schema is defined by means of an ER diagram that defines entities, relationships and *isA* relationships. An entity defines structure and behaviour by means of attributes and methods. This stems from the fact that we have realised our approach based on an object-oriented model where we represent entities as native constructs of the system. Entities can be associated through relationships and cardinality constraints may be defined over the relationship. An *isA* relationship is a directed relationship that defines one entity to be a specialisation of another.

As shown in the last section, systems are developed by composing, adapting and extending micro-schemas to form more complex schemas. It has been shown in previous work on peer-to-peer databases [14] that models with features and constructs such as specialisation and associations naturally support composition. Specialisation allows for intuitive composition of micro schemas through inheritance, while associations act as a mechanism that allows for the natural composition of schemas as well as the decomposition of a more complex schema into micro schemas that can be shared and reused.

We support these two types of composition mechanisms: composition by specialisation and composition by association, as illustrated in Fig. 4.

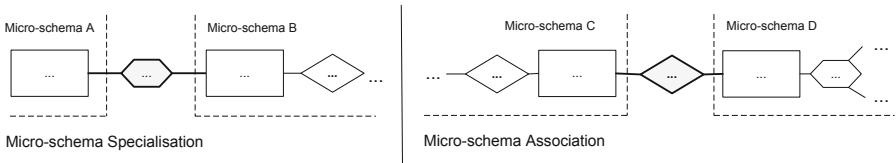


Fig. 4. Composition Mechanisms

Composition by specialisation, shown on the left, specifies by means of an *is-a* relationship that an entity of a micro-schema A is a specialisation of an entity of micro-schema B. Composition by association, illustrated on the right, supports schema composition by associating an entity of a micro-schema C to an entity of a micro-schema D by means of a relationship, where the relationship is refined through cardinality constraints.

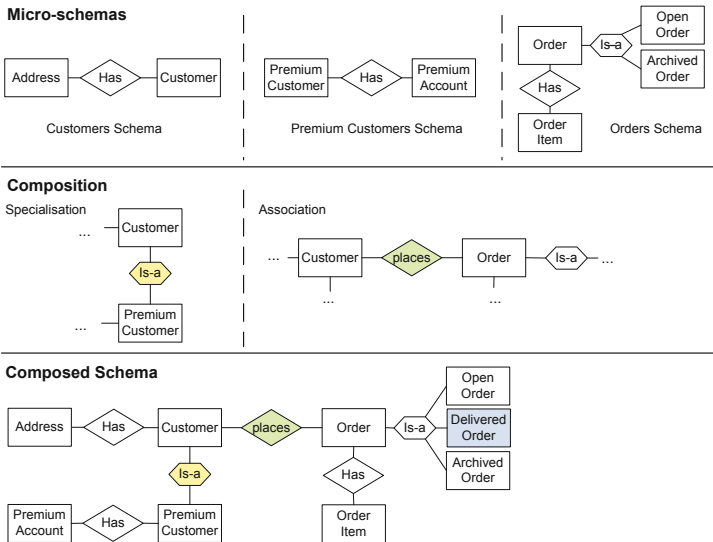


Fig. 5. Example

We will now explain the two composition mechanisms in more detail, based on the example in Fig. 5, where we illustrate the composition of three micro-schemas that form part of an eCommerce platform. In the upper part of the figure, three micro-schemas are illustrated, one for the management of customers, one for the management of premium customers and a third for the management of orders. These micro-schemas will be composed, as shown in the centre part of the figure to form a more complex schema, illustrated in the lower part of the figure. We will now focus on the composition process.

Composition by specialisation is shown in the middle left of Fig. 5, where the `PremiumCustomer` entity of the premium customers micro-schema has been defined to be a specialisation of the `Customer` entity defined as part of the customers micro-schema. The composition is defined by a specialisation specification, illustrated in Fig. 6. In (a), the two entities `Customer` and `PremiumCustomer` are represented by means of UML class diagrams showing their attributes and methods. The specialisation relationship between these two entities is defined by means of the specialisation specification defined by the user who composes the two schemas. (b) illustrates the specialisation specification. A specialisation generally consists of two steps:

- definition of `isA` relationship
- attribute mappings

In the example, the specification first defines `PremiumCustomer` to be a specialisation of `Customer` using an `isA` relationship. Then, mappings between the attributes `Customer.firstName` and `PremiumCustomer.foreName` as well as between the attributes `Customer.lastName` and `PremiumCustomer.name` have been defined. Applying this composition specification to the model in (a), results in a transformed and composed model shown in (c), where the entity `PremiumCustomer` has become a sub-entity of `Customer`. The attribute mappings resulted in the generalisation of the two attributes `forename` and `name` of the `PremiumCustomer` entity.

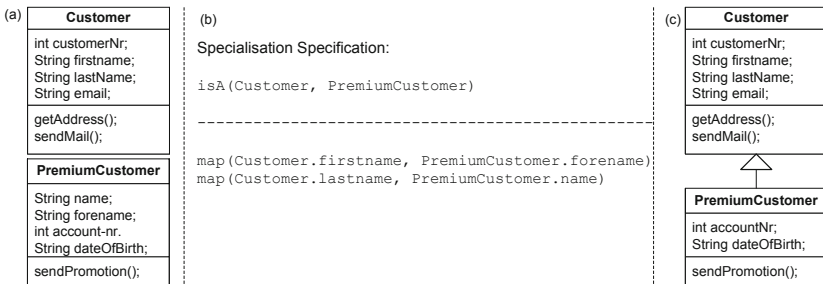


Fig. 6. Composition by Specialisation

As a consequence of these transformation rules, operations defined on the sub-entity can operate on the mapped attributes of the super-entity. For example, the method `sendPromotion`, which makes use of the `forename` attribute of

the `PremiumCustomer` entity will be redirected to use the `firstName` attribute defined by the `Customer` entity.

Composition by association allows micro-schemas to be associated through a relationship between two entities. To given an example, in the middle right of Fig. 5, we associate the `Customer` entity of the customers micro-schema to the `Order` entity of the orders micro-schema via a `places` association that allows customers to place orders. Figure 7 gives more detail about the composition by association. In (a), the two entities `Customer` and `Order` are illustrated. Through the association specification shown in (b), these entities are associated by a `places` relation to form the model shown in (c), where customers can place orders.

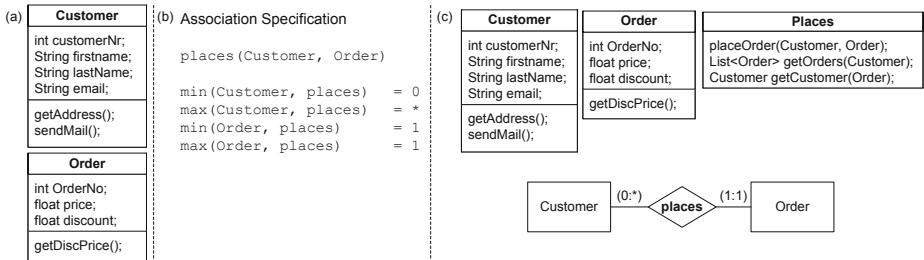


Fig. 7. Composition by Association

The association specification defines the relation `places` from `Customer` to `Order` as well as the constraints in terms of minimum and maximum cardinality constraints. In this example, a customer may place *zero* or *n* orders, while an order is placed by exactly one customer. From the specification of the relation and the constraints, a `Places` class is generated that offers a number of operations that allow two instances to be related as well as to navigate from an instance to a related one. In our specific example, the `Places` class offers methods to place an order and to retrieve the set of orders for a given customer as well as a customer of a given order. Note that the cardinality constraints have an impact on the operations and their return values.

## 5 eCommerce Platform

We have designed and implemented an eCommerce platform based on the approach and architecture presented in the previous sections. The platform provides the functionality to configure an eCommerce system based on the composition of micro-schemas to form a customised and individual eCommerce solution. Micro-schemas are provided by the platform provider and can be used, extended and adapted by the platform users. The platform is extensible in that users are free to develop their own, customised micro-schemas, which can in turn be shared with the platform community. We will first illustrate the design process, before giving a more detailed overview of the platform architecture.

Figure 8 summarises the eCommerce system design process. As shown in (1), a platform user first downloads the eCommerce kernel and installs it on their local server. The kernel exposes the functionality to compose and configure an eCommerce system from shared micro-schemas. In the design phase, shown in (2), the eCommerce system is designed through a process of browsing and inspecting the micro-schema repository situated on the eCommerce platform, selecting and importing suitable micro-schemas into the local system, and, finally, composing, adapting and extending them to form an individual eCommerce system.

For the task of finding suitable micro-schemas, a user can rely on the discovery and inspection system built on top of the repository for browsing and inspecting micro-schemas, which basically exposes the **search** and **browse** operations through a graphical user interface. Once a suitable micro-schema has been found, the **reuse** operation can be used to import a micro-schema into a local eCommerce system for further composition.

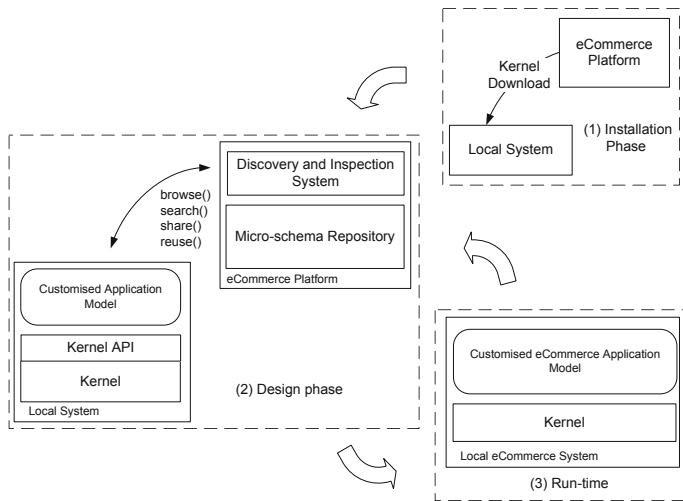
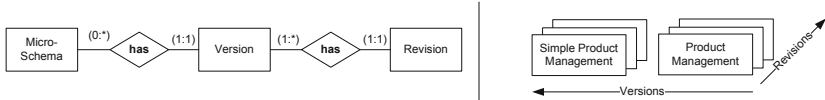


Fig. 8. Design Process

After importing all the required micro-schemas, the user starts to compose, extend and adapt the micro-schemas to form a customised eCommerce application model using the kernel API. The design process may consist of a number of such iterations, where the user browses the repository, imports appropriate micro-schemas and composes them. This results in a modular and incremental eCommerce system design process.

Local users can also share a micro-schema with the community by calling the **share** operation, passing a name, description and the schema definition. Note that a user may also share an adapted or extended version, as well as revisions of an already shared micro-schema to support the collaborative micro-schema creation. Figure 9 gives an overview of how micro-schemas are managed along with their versions and revisions. The metamodel on the left indicates that a

micro-schema may have a number of different versions, and each version may have a number of revisions. On the right, we illustrate the scenario for a product management micro-schema that has two versions, a simple and a more complex one, and each version has a number of revisions. Note that versions are identified by tags, while revisions are identified by revision numbers and the first version of a micro-schema is the default version. New versions have to be tagged by the user when offered for sharing.

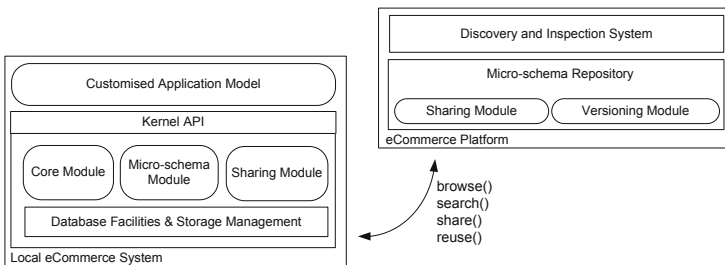


**Fig. 9.** Micro-Schema Versioning

Once the design process has been completed, the local eCommerce system is deployed and used in production, as shown in (3). Note that we support an agile design process, where an eCommerce system can be extended with additional micro-schemas to offer new functionality at a later point in time, as indicated by the arrow between step (3) and (2).

Figure 10 gives a detailed overview of the platform architecture that supports this system design process. The architecture consists of the eCommerce platform, shown on the right, and a number of local eCommerce systems that have installed the eCommerce kernel and make use of the micro-schema repository for their system design. The local eCommerce system on the left consists of a kernel installation and a customised eCommerce application model. The kernel exposes the functionality to compose and configure an eCommerce system from shared micro-schemas through the Kernel API and consists of three modules which are built on top of standard database facilities and storage management. The core module encapsulates data definition functionality to design an application model in terms of standard data definition operators to create entities, relationships, specialisations and constraints as well as data manipulation functionality for basic data creation, retrieval, manipulation and deletion operations.

The micro-schema module supports the schema composition operations and orchestrates the sharing and reuse of micro-schemas with the micro-schema repository, illustrated on the right as part of the eCommerce platform.



**Fig. 10.** Platform Architecture



For micro-schema sharing, the system relies on a sharing module that allows objects to be shared between peer databases without having to deal with the underlying connection and data transfer operations as described in [15]. When a micro-schema is shared, it is registered with and copied to the micro-schema repository, using the sharing module to transfer a copy of the shared schema. Similarly, the sharing module is used during an import operation to transfer copies of the schema to the local eCommerce system. We currently offer *detached* and *synchronised* modes of sharing. The detached mode creates an independent copy of the imported micro-schema, while the synchronised mode allows users to be notified when new revisions of an imported micro-schema are published in the micro-schema repository and offers them the option to update their local copy accordingly. We note, however, that the detached mode is the default, and it is not the intention that imported micro-schemas should automatically evolve to reflect changes in shared micro-schemas since users should be free to use shared micro-schemas in any way they wish and the decision on this is taken at the time of importation. Thus, even if a micro-schema is later removed from the micro-schema repository, this does not effect those users who have already imported it. We accept, however, that users might be interested in extended models or new features that appear in later revisions and therefore offer a notification scheme together with a user-controlled synchronisation tool. Furthermore, adapted and extended versions of a micro-schema can be registered as such with the directory. For the management of versions and revisions, we rely on a versioning module described in [16], that offers the functionality to manage versions and variants of an object.

## 6 Discovery and Inspection System

While we have shown how users can compose their eCommerce system from shared micro-schemas, we have yet to describe how users can find micro-schemas suited to their specific requirements. To support them in this task, we have designed a discovery and inspection system.

The process of finding an appropriate micro-schema consists of two steps: discovering and inspecting. First, a user has to find appropriate micro-schemas by searching and browsing the repository. This is then followed by an in-depth inspection of suitable micro-schemas to help the users understand the purpose of the micro-schema and decide whether it meets their requirements.

The screenshot in Fig. 11 shows the search interface, where users can search for micro-schemas by keyword or by browsing the tag cloud. The tag cloud is generated from the terms used in the descriptions of the shared micro-schemas. When searching for an appropriate micro-schema, the user can either enter a keyword to initiate the search process, or can select a tag in the tag cloud, which triggers a search over the repository. The search result can further be refined by using a combination of keyword search and tag cloud browsing.

As an alternative, the user may directly select a micro-schema from one of the various collections. In the current example, three new micro-schemas are displayed in the *New* collection on the left, one for the management of wish-lists, one that supports rich product classification and one for visitor statistics.

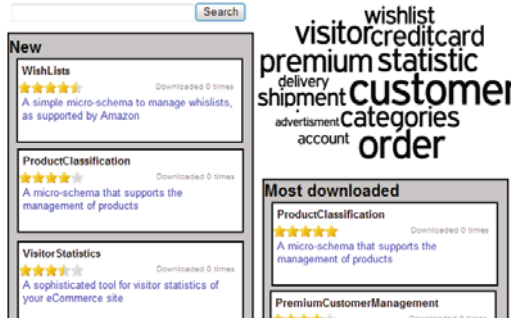


Fig. 11. Search Interface

On the right, the collection of most downloaded micro-schemas is shown. Micro-schemas are summarised by rectangular boxes with the micro-schema name on top and a description provided by its developer as well as the community rating and the number of downloads.

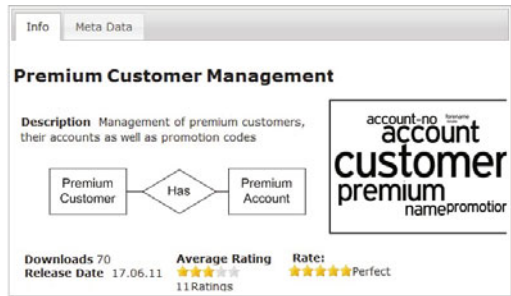


Fig. 12. Detail View

Once a suitable micro-schema has been found, the user may select and inspect it more in detail. The screenshot in Fig. 12 illustrates the user interface for micro-schema inspection. The micro-schema inspection view consists of a textual description, a data model, a tag cloud and additional information such as community ratings, release date and number of downloads. The micro-schema shown supports the management of premium customers where premium customers have a premium customer account. The developer of this micro-schema has provided a textual description as well as a data model. The tag cloud is generated from the micro-schema specification with tags being created from entity and relationship names as well as from entity attribute names and methods. The size of the tags corresponds to the number of occurrences of a name in the micro-schema.

## 7 Implementation

We have implemented the platform and kernel based on object database technology. Specifically, we extended OMS Avon [17], a Java-based object database

that implements the OM model [18], with functionality to support the modular design of systems based on the composition of micro-schemas.

OMS Avon can be extended by means of metamodel extension modules [17] that integrate new data management concepts into the system metamodel, thereby allowing these concepts to be natively integrated within the database as metadata concepts. This is enabled through the fact that OMS Avon treats metadata and data uniformly as data objects and, therefore, metadata are represented as data objects that can be created, retrieved, manipulated and deleted like regular data objects. Extension modules consist of new metamodel concepts, corresponding CRUD classes, and an optional database language extension. The CRUD classes implement the basic data management operations of creating, reading, updating and deleting instances of the metamodel concepts and provide additional methods that implement higher-level operations. Note that the core model of the database is also implemented as a module and its core concepts are accessible through corresponding CRUD classes.

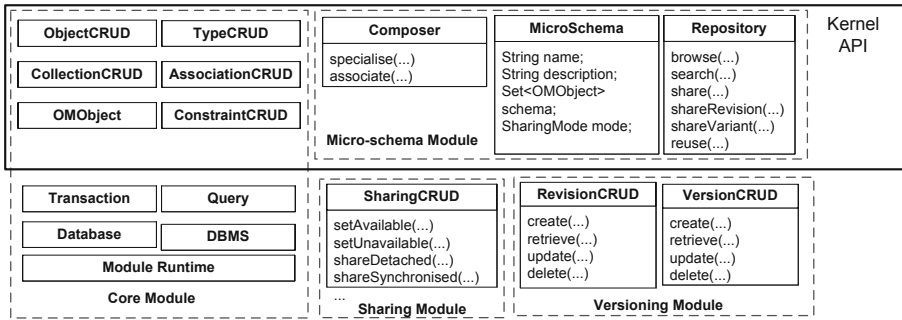


Fig. 13. UML diagram

We used the extension mechanism to realise the concept of micro-schemas, their composition, sharing and versioning. Figure 13 gives a system overview, which illustrates both the kernel implementation as well as the platform implementation. The kernel implementation consists of three modules, namely the core module, the micro-schema module and the sharing module, while the platform implementation in addition uses a versioning module. The core module provides data definition and data manipulation functionality and is used to define, extend and adapt micro-schemas. Note that while we represent the application model using ER notation where entities are represented as a single construct, the implementation of the core model clearly distinguishes between the notions of entity types and entity sets, using object types to represent entity types and collections of objects to represent entity sets. Consequently, our metamodel is defined by the concepts of type, collection, association and constraint, where constraints include sub-type relationships as well as sub-collection relationships.

Metamodel instances are represented with a single Java class `OMObject` according to the type object pattern [19]. The core CRUD classes are used to handle instances of that class as instances of the metamodel concepts. For example, an

object representing a collection is handled using the `CollectionCRUD` offering facilities such as creating and deleting a collection as well as adding and removing members. Similarly, objects that represent associations are created, retrieved, updated and deleted using the `AssociationCRUD`. Note that domain objects are also represented as `OMObject` instances and handled using the `ObjectCRUD`. The core module also provides access to database facilities such as `Database`, `DBMS` and `Transaction` as well as the module management which allows extension modules to be loaded into the database.

Micro-schema composition, sharing and reuse is encapsulated in the micro-schema module that, together with the CRUD classes of the core module, forms the kernel API used to design an eCommerce system. The `Composer` class supports micro-schema composition. The `specialise` method is used on the typing level and creates an isA relationship between two object types, while the `associate` method associates two collections of objects. These two methods differ from the regular creation of specialisation and association through the CRUD classes of the core API in that they implement the functionality described in Sect. 4 where existing metadata objects are related as part of the composition process. The `specialise` method takes two type objects and a set of attribute mappings as input parameters to perform the specialisation. The `associate` method takes two collection objects to be associated and the cardinality constraints as input parameters and generates a set of methods to navigate the association based on the cardinality constraints.

The composition operators have been realised as follows. In the case of specialisation, attribute access to mapped attributes is redirected according to the mapping specification. In the example in Figure 14, we show a specialisation scenario of two type objects represented by means of two Java classes for the sake of illustration. The `Customer` type is illustrated next to the `PremiumCustomer` type. The method `sendPromotionCode` defined on a `PremiumCustomer` accesses the attribute `forename` via its getter method. This method, however, has been altered and invokes the getter of the mapped attribute in the superclass, which is `getFirstname`, as shown in the `getForename` method body. Upon composition, the types and their methods are manipulated to obtain the required behaviour.

The composition by association is realised by using the `AssociationCRUD` for creating the association object and, based on the cardinality constraints, appropriate methods for navigation and creation are generated as part of the association object.

The `Repository` class manages the access to the micro-schema repository and orchestrates the sharing and reuse operations by abstracting from the underlying distribution. It offers the possibility to search and browse the repository as well as to import shared micro-schemas using the `reuse` method, where the name of the micro-schema along with version and revision information is passed as input parameters. Users are also supported in registering new micro-schemas to be shared as well as sharing revisions and versions of previously imported micro-schemas, using the appropriate `share` method. In our implementation, micro-schemas are represented as a set of metadata objects. To share them through the repository,

<pre>public class Customer {     private String firstName;     private String lastName;     ....     public void setFirstName(String firstName){         this.firstName = firstName;     }     public String getFirstName(){         return this.firstName;     }     .... }</pre>	<pre>public class PremiumCustomer extends Customer {     private int accountNr.     ...     //setters and getters ...     String getForeName(){         return super.getFirstName()     }     ...     public String sendPromotion(){         ...         String name = this.getForeName();         ...         ...     } }</pre>
--	--

**Fig. 14.** Specialisation Implementation

they are wrapped with a `MicroSchema` object that builds a repository entry consisting of a micro-schema name, a description, the set of metadata objects that define the schema and the sharing mode, which is either `detached` or `synchronised`.

Note that we have two implementations of these classes, one for the kernel and one for the platform. The kernel class acts as a proxy and forwards `search` and `browse` method invocations to the the server method implementations, where the repository resides. Micro-schema sharing and reuse is supported by a sharing module described in [15]. The module is represented by the `SharingCRUD` and supports collection-based sharing between two peers based on a push approach, where members of a collection residing in one database can be pushed to a collection residing in another database. The sharing module supports different sharing modes: the method `shareDetached` simply copies objects to the remote database, while the method `shareSynchronised` notifies target databases about new revisions, and, if desired, propagates the updates. When a local user shares a micro-schema, the `Repository` invokes the appropriate share method, which handles the copy process and registration of the shared micro-schema to the micro-schema repository. To import a micro-schema from the micro-schema repository, the local user calls the `reuse` method. Since the sharing module only supports pushed-based sharing, the import of metadata into a local system is realised by means of a separate `request` object that encapsulates the import queries. The `request` object triggers the repository to push the metadata objects to be reused to the local eCommerce system.

As mentioned previously, the platform makes use of a versioning module presented in [16] to support the creation and sharing of versions and revisions.

## 8 Discussion

Our approach is in line with research trends in enduser development support. Lieberman et al. [20] have argued that, while most current work addresses the problem of designing systems that are easy to use, the challenge of the future lies in providing systems that are easy to develop. Our work is centred around such a system that supports users in the design of eCommerce systems based on micro-schemas shared with the community.

While we have presented this work in the domain of eCommerce systems, we would like to highlight the generality of the approach and its applicability to information system design in general, where users could be supported in modularly designing their information systems through the composition of domain-specific micro-schemas. Given that our platform kernel is general and basically offers micro-schema composition functionality, a company would simply have to offer a micro-schema repository with domain-specific micro-schemas that could be used for composing domain-specific information systems. For example, companies such as Wordpress that offer web publishing solutions could take over this idea and offer support for modular web information system design and composition at the level of the data model. Also, we have shown in previous work, how such an approach could be applied to the domain of personal information management [21], where non-expert users could design and compose their information space through a graphical user-interface.

The success of such a platform stands and falls with its usability and user acceptance and it is of utmost importance that users are supported in finding appropriate micro-schemas in an easy and intuitive way. The discovery and inspection system is a first approach towards supporting users in this task.

There are also issues of security and trust that deserve attention. It is important to have mechanisms that can ensure, for example, that a user's local system cannot be accessed by other parties unless explicitly shared, and that imported micro-schemas do not perform any malicious actions. Company-deployed platforms as presented in this work are capable of controlling the sharing process, for example by means of rules or in-depth examination of shared artefacts. Also, we have integrated community-driven mechanisms for micro-schema quality control by means of ratings.

Finally, we plan to complement the notion of micro-schemas with user interface definitions, building on the approach presented in [21], to further facilitate the design process.

## 9 Conclusion

We have presented an approach to modular information system development based sharing and reuse. As proof of concept, we have implemented a platform that supports the development of eCommerce solutions based on a set of micro-schemas that represent basic building blocks that can be selectively customised and composed. While the design is based on the ER model, the implementation is based on an object database that supports association and specialisation constructs as a natural basis for schema composition and allows behaviour as well as data to be reused in applications.

## References

1. Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR (2005)
2. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., Gandhi, P.: *Intel Mash Maker: Join the Web*. *ACM SIGMOD Rec.* 36(4) (2007)

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. Mork, P., Seligman, L., Rosenthal, A., Morse, M., Wolf, C., Hoyt, J., Smith, K.: Galaxy: Encouraging Data Sharing among Sources with Schema Variants. In: ICDE (2009)
5. Chen, K., Madhavan, J., Halevy, A.: Exploring Schema Repositories with Schemr. In: SIGMOD 2009 (2009)
6. Delcambre, L.M.L., Langston, J.: Reusing (Shrink Wrap) Schemas by Modifying Concept Schemas. In: ICDE 1996 (1996)
7. Welzer, T., Rozman, I., Družovec, M., Horvat, R.V., Takač, I., Brumen, B.: Database Reusability in Intelligent Medical Systems. *J. Med. Syst.* 25(2) (2001)
8. Cushing, J.B., Nadkarni, N., Finch, M., Fiala, A., Murphy-Hill, E., Delcambre, L., Maier, D.: Component-based End-User Database Design for Ecologists. *J. Intell. Inf. Syst.* 29(1) (2007)
9. Castano, S., De Antonellis, V., Zonta, B.: Classifying and Reusing Conceptual Schemas. In: Pernul, G., Tjoa, A.M. (eds.) ER 1992. LNCS, vol. 645, pp. 121–138. Springer, Heidelberg (1992)
10. Koch, S., Strecker, S., Frank, U.: Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach. In: OSS 2006 (2006)
11. Casanova, M.A., Furtado, A.L., Tucherman, L.: A Software Tool for Modular Database Design. *ACM Trans. Database Syst.* 16(2) (1991)
12. Thalheim, B.: Component Development and Construction for Database Design. *Data Knowl. Eng.* 54(1) (2005)
13. Kazman, R., Chen, H.-M.: The Metropolis Model a New Logic for Development of Crowdsourced Systems. *Commun. ACM* 52(7) (2009)
14. Norrie, M.C., Palinginis, A.: A Modelling Approach to the Realisation of Modular Information Spaces. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 245–261. Springer, Heidelberg (2002)
15. de Spindler, A., Grossniklaus, M., Lins, C., Norrie, M.C.: Information Sharing Modalities for Mobile Ad-Hoc Networks. In: COOPIS 2009 (2009)
16. Grossniklaus, M., Norrie, M.C.: An Object-Oriented Version Model for Context-Aware Data Management. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 398–409. Springer, Heidelberg (2007)
17. Grossniklaus, M., Leone, S., de Spindler, A., Norrie, M.C.: Dynamic Metamodel Extension Modules to Support Adaptive Data Management. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 363–377. Springer, Heidelberg (2010)
18. Norrie, M.C., Grossniklaus, M., Decurtins, C., de Spindler, A., Vancea, A., Leone, S.: Semantic Data Management for db4o. In: ICOODB 2009 (2009)
19. Martin, R.C., Riehle, D., Buschmann, F. (eds.): *Pattern Languages of Program Design 3*. Addison-Wesley Professional (1997)
20. Lieberman, H., Paternò, F., Wulf, V.: *End User Development*. Human-Computer Interaction Series. Springer, Heidelberg (2006)
21. Leone, S., Geel, M., Norrie, M.C.: Managing Personal Information through Information Components (chapter 1). In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 1–14. Springer, Heidelberg (2011)

# $A^2$ -VM : A Cooperative Java VM with Support for Resource-Awareness and Cluster-Wide Thread Scheduling

José Simão<sup>1,2</sup>, João Lemos<sup>1</sup>, and Luís Veiga<sup>1</sup>

<sup>1</sup> Instituto Superior Técnico, Technical University of Lisbon / INESC-ID Lisboa

<sup>2</sup> Instituto Superior de Engenharia de Lisboa

jsimao@cc.isel.ipl.pt, luis.veiga@inesc-id.pt

**Abstract.** In today's scenarios of large scale computing and service providing, the deployment of distributed infrastructures, namely computer clusters, is a very active research area. In recent years, the use of Grids, Utility and Cloud Computing, shows that these are approaches with growing interest and applicability, as well as scientific and also commercial impact.

This work presents the design and implementation issues of a cooperative VM for a distributed execution environment that is resource-aware and policy-driven. Nodes cooperate to achieve efficient management of the available local and global resources. We propose  $A^2$ -VM, a cooperative cluster-enabled virtual execution environment for Java, to be deployed on Grid sites and Cloud data-centers that usually comprise a number of federated clusters. This cooperative VM has the ability to monitor base mechanisms (e.g. thread scheduling, garbage collection, memory or network consumptions) to assess application's performance and reconfigure these mechanisms in run-time according to previously defined resource allocation policies.

We have designed this new cluster runtime by extending the Jikes Research Virtual Machine to incorporate resource awareness (namely resource consumption and restrictions), and extending the TerraCotta DSO with a distributed thread scheduling mechanism driven by policies that take into account resource utilization. In this paper we also discuss the cost of activating such mechanisms, focusing on the overhead of measuring/metering resource usage and performing policy evaluation.

## 1 Introduction

In today's scenarios of large scale computing and service providing, the deployment of distributed infrastructures, namely computer clusters, is a very active research area. In recent years, the use of Grids, Utility and Cloud Computing, shows that these are approaches with growing interest and applicability, as well as scientific and commercial impact.

Managed languages (e.g., Java, C#) are becoming increasingly relevant in the development of large scale solutions, leveraging the benefits of a virtual execution environment (VEE) to provide secure, manageable and componentized



solutions. Relevant examples include work done in various areas such as web application hosting, data processing, enterprise services, supply-chain platforms, implementation of functionality in service-oriented architectures, and even in e-Science fields (e.g., with more and more usage of Java in the context of physics simulation, economics/statistics, network simulation, chemistry, computational biology and bio-informatics [15,14,17], there being already many available Java-based APIs such as Neobio)<sup>1</sup>

To extend the benefits of a local VEE, while allowing scale-out regarding performance and memory requirements, many solutions have been proposed to federate Java virtual machines [22,23], aiming to provide a single system image where the managed application can benefit from the global resources of the cluster.

A VEE cluster-enabled environment can execute applications with very different resource requirements. This leads to the use of selected algorithms for runtime and system services, aiming to maximize the performance of the applications running on the cluster. However, for other applications, for example the ones owned by restricted users, it can be necessary to impose limits on their resource consumption. These two non functional requirements can only be fulfilled if the cluster can monitor and control the resources it uses both at the VEE and distributed level, and whether the several local VEE, each running on its node, are able to cooperate to manage resources overall.

Existing approaches to resource-awareness in VMs, cluster-enabled runtimes, and adaptability are still not adequate for this intended scenario (more details in Section 5) as they have not been combined into a single infrastructure for unmodified applications. Existing resource-aware VMs do not target popular platforms, and cluster-enabled runtimes have support neither for global thread scheduling nor for checkpointing/restore mechanisms. Furthermore, lower-level mechanisms and VM parameters cannot be governed by declarative policies.

In this paper, we report on the design and implementation of *A<sup>2</sup>-VM* (*Autonomous and Adaptive Virtual Machine*), a distributed execution environment where nodes cooperate to make an efficient management of the available local and global resources. We propose a cluster-enabled VEE with the ability to monitor base mechanisms (e.g. thread scheduling, garbage collection, memory or network consumptions) at different nodes in order to assess an application's performance and resource usage, and reconfigure these mechanisms in run-time, in one or more nodes, according to previously defined resource allocation policies (or *quality-of-execution* specifications). These policies regulate how resources should be used by the application in the cluster, leading to the adaptation of components at different levels of the cluster in order to enforce it.

We propose a layered approach to resource monitoring, management and restriction enforcement. While restrictions to resources are effectively enforced at the level of each of the individual cooperating VMs, overall performance assessment and policy-driven resource management are carried out by node and cluster-wise manager components.

---

<sup>1</sup> <http://www.bioinformatics.org/neobio/>

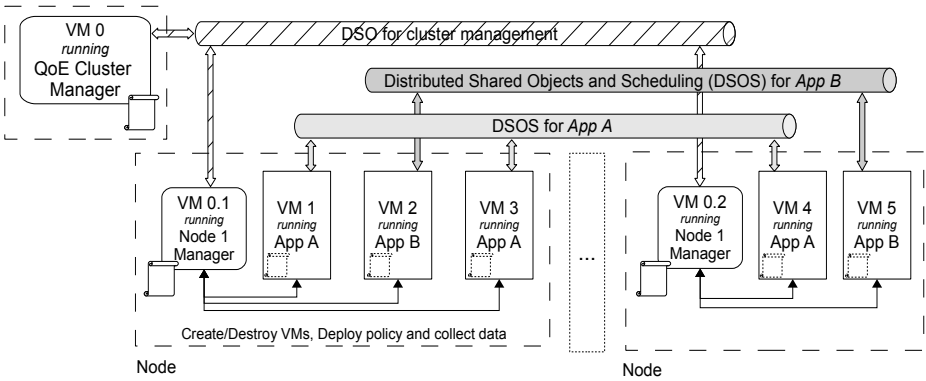


Fig. 1. Architecture of the  $A^2$ -VM

The rest of this paper is organized as follows. Section 2 describes the overall architecture of our proposal for a resource-aware and policy-driven cooperative VM, introducing the main components of  $A^2$ -VM, and details on the specific internal mechanisms and functionality offered. Section 3 describes the main aspects of the current development and implementation of  $A^2$ -VM. In Section 4, we assess and evaluate  $A^2$ -VM, regarding: i) the impact of resource awareness and policy support (measuring overhead of resource monitoring and policy engine performance), and ii) the cluster-wide cooperative thread scheduling offered (overheads and performance improvements). In Section 5, we discuss our research in light of other systems described in the literature, framing them with our contribution. Section 6 closes the paper with conclusions and future work.

## 2 Architecture

The overall architecture of  $A^2$ -VM (*Autonomous and Adaptive Virtual Machine*) is presented in Figure 1. We consider a *cluster* as a typical aggregation of a number of *nodes* which are usually machines with one or more multi-core CPUs, with several GB of RAM, interconnected by a regular LAN network link (100 Mbit, 1 Gbit transfer rate). We assume there may be several applications, possibly from different users, running on the cluster at a given time, i.e., the cluster is not necessarily dedicated to a single application. The cluster has one top-level coordinator, the *QoE Manager* that monitors the *Quality-of-Execution* of applications.<sup>2</sup>

Each *node* is capable of executing several instances of a Java VM, with each VM holding part of the data and executing part of the threads of an application. As these VMs may compete for the resources of the underlying cluster

<sup>2</sup> We opt for this notion instead of hard *service-level agreements* usually employed in commercial application hosting scenarios, because we intend to target several types of applications in shared infrastructures, without necessarily strict contractual requirements.

node, there must be a *node manager* in each node, in charge of VM deployment, lifecycle management, resource monitoring and resource management/restriction. Finally, in order for the node and cluster manager to be able to obtain monitoring data and get their policies and decisions carried out, the Java VMs must be resource-aware, essentially, report on resource usage and enforce limits on resource consumption. Cooperation among VMs is carried out via the *QoE Manager*, that receives information regarding resource consumption in each VM, by each application, and instructs VMs to allow or restrict further resource usage.

In summary, the responsibilities of each of these entities are the following:

- Cluster QoE Manager
  - collect global data of cluster applications (i.e. partitioned across VMs and nodes)
  - deploy/regulate nodes based on user’s QoE
- Node QoE Manager
  - report information about node load
  - deploy new policies on VMs
  - create or destroy new instances
  - collect VM’s resource usage data
- (resource-aware) VM
  - enforce resource usage limits
  - give internal information about resource usage

A<sup>2</sup>-VM is thus comprised of several components, or building blocks. Each one gives a contribution to support applications with a global distributed image of a virtual machine runtime, where resource consumption and allocation is driven by high-level policies, system-wide, application or user related. From a bottom-up point of view, the first building block above the operating system in each node is a process-level managed language virtual machine, enhanced with mechanisms and services that are not available in regular VMs. These include the accounting of resource consumption.

The second building block aggregates individual VMs, as the ones mentioned above, to form within the cluster a distributed shared object space assigned to a specific application. It gives running applications support for single system image semantics, across the cluster, with regard to the object address space. Techniques like bytecode enhancement/instrumentation or rewriting must be used, so that unmodified applications can operate in a partitioned global address space, where some objects exist only as local copies and others are shared in a global heap.

The third building block turns A<sup>2</sup>-VM into a cluster-aware cooperative virtual machine. This abstraction layer is responsible for the global thread scheduling in the cluster, starting new work items in local or remote nodes, depending on a cluster wide policy and the assessment of available resources. This layer is the A<sup>2</sup>-VM boundary that the cluster-enabled applications interface with (note that for the applications, the cluster looks like a single, yet much *larger*, virtual machine). Similarly to the previous block, application classes are further

instrumented/enhanced (although the two sets of instrumentation can be applied in a single phase), in order to guarantee correct behavior in the cluster. Finally, it exposes the underlying mechanisms to the adaptability policy engine, and accepts external commands that will regulate how the VM's internal mechanisms should behave.

The resource-aware VM, the distributed shared object layer, and the cluster level scheduler are all sources of relevant monitoring information to the policy engine of  $A^2-VM$ . This data can be used as input to declarative policies in order to determine a certain rule outcome, i.e. what action to perform when a resource is exhausted or has reached its limit, regarding a user or application.

Currently, thread scheduling tries first to collocate threads of the sample application to the same VM, until the specified CPU load, wait time, and memory usage thresholds are reached. After that, subsequent threads are allocated elsewhere within the same node or across the cluster, balancing the load. Application performance is monitored by a combination of black-box and grey-box approaches. Black-box consists in the monitoring of the parameters mentioned above that allows us to determine, roughly, whether an application is experiencing poor performance due to resource shortage or contention. Grey-box approach consists in monitoring advancement of file cursors (for sequential reading and writing), and data transferred in order to estimate current progress against previous executions of the same application.

On top of this distributed runtime are the applications, consuming resources on each node and using the services provided by the resource-aware VM that is executing on each one.  $A^2-VM$  targets mainly applications with a long execution time and that may spawn several threads to parallelize their work, as usual in e-Science fields such as those mentioned before.

The following sections will describe the depicted architecture, explaining the specific contributions of each component.

## 2.1 Resource Awareness and Control

The Resource Aware virtual machine is the underlying component of the proposed infrastructure. It has two main characteristics: *i*) resource usage monitoring, and *ii*) resource usage restriction or limitation. Furthermore, there are checkpointing, restore and migration mechanisms, used for more coarse-grained load-balancing across the cluster, that are out of the scope of this paper [13].

Current virtual machines (VM) for managed languages can report about several aspects of their internal components, like used memory, number of threads, classes loaded [19,18]. However they do not enforce limits on the resources consumed by their single node applications. In a cluster of collaborating virtual machines, because there is a limited amount of resources to be shared among several instances, some resources must be constrained in favor of an application or group of applications.

Extending a managed language VM to be aware of the existing resources must be done without compromising the usability (mainly portability) of application code. The VM must continue to run existent applications as they are. This

component is an extended Java virtual machine with the capacity to extract high and low level VM parameters, e.g., heap memory, network and system threads usage. Along with the capacity to obtain these parameters, they can also be constrained to reflect a cluster policy. The monitoring system is extensible in the number and type of resources to consider.

## 2.2 Cluster-Wide Cooperative VM

To enable effective distribution of load among different nodes of the cluster, our system relies on a cluster level load balancer capable of spawning new threads (or work tasks) on any cluster node based on a cluster wide policy. When an application asks for a new thread to be created (e.g., by invoking the `start` method on a `Thread` object), the request can be either denied or granted based on the resource allocation decided for the cluster. If it is granted, the load balancer will create the new thread in the most appropriate node to fulfill the cluster policy. For example, if the application has a high priority order compared to other applications of the cluster, then the thread could be created in a lesser loaded node (preferably, one with a VM already assigned to the application's DSO; if needed and allowed, a new VM on any lesser loaded node) . The decision on what node new threads are created is left to the policy engine to decide with current information. Nevertheless, the resource-aware VM has an important role in this process by making it possible to impose a hard limit on resources, e.g., the number of running threads of the application at a specific node, or globally.

A<sup>2</sup>-VM aims to accommodate applications developed by users with different levels of expertise regarding the development of multi-threaded applications and cluster architectures, giving a performance versus transparency trade-off. To this end, the thread scheduler has two operation modes: i) Identity and ii) Full SSI. **Identity** mode should be used if we have the byte-code of a multi-threaded Java application that is *explicitly* synchronized (e.g., using Java monitors), or there is access to the source code and synchronization code can be added with ease. **Full SSI** mode should be used if we have the byte-code of a multi-threaded Java application that is not explicitly synchronized (mainly applications comprised of cooperating threads that make use of *volatile* object fields that the Java VM specification assures to be updated in a single memory write operation, while Terracotta does not honour this) and there is no access to the source code. For instance, in DaCapo 2009 benchmarks, 6 out of 14 applications do indeed use such *volatile* fields, and rely on the VM to uphold this semantics, hence the relevance of the Full SSI mode to ensure compatibility, transparency and correct functionality when deploying such applications with A<sup>2</sup>-VM on a cluster.

## 2.3 Adaptability and the Policy Engine

The policy engine is responsible for loading and enforcing the policies provided by administrators and possibly users regarding resource management. It achieves this by, globally, sending the necessary commands to the resource-aware VMs in order for them to modify some runtime parameters, or the type of algorithm

```

<?xml version="1.0" encoding="UTF-8" ?>
<RAMConfiguration>
  <ResourceAttributes name="NumberOfThreads" initialLimit="15" />
  <ResourceAttributes name="CpuUsage" initialLimit="75%" />
  ...
  <Rule target="NumberOfThreads">
    <!-- Determines how accumulation is done -->
    <OnConsume> <Counter/> </OnConsume>
    <!-- Determines what happens if limit is reached -->
    <OnLimit> <ResourceException/> </OnLimit>
    <!-- Determines what happens if consumption is successful -->
    <OnAfterConsumption>
      <UseCluster threshold="AllCpus"/>
    </OnAfterConsumption>
  </Rule>
  <Rule target="CpuUsage">
    <OnConsume> <HistoryAverage window="5"/> </OnConsume>
    <OnLimit> <Suspend miliseconds="500"/> </OnLimit>
  </Rule>
  ...
</RAMConfiguration>

```

**Fig. 2.** Declarative policy

used to accomplish a cluster related task, as well as instructing them to spawn threads or activate checkpointing/restore and migration mechanisms. A special focus of this component of  $A^2$ -VM is also on the improvement of applications' performance, and what can be adapted in the underlying resource-aware VMs in order to achieve it.

It operates autonomously or in reaction to a given resource outage in the VMs. Autonomous behavior is governed by maintaining knowledge about the applications' previous execution, and adjusting the VMs and cluster parameters to achieve better performance for that specific application. Reactive operation is driven by declarative policies that determine the response to a resource outage. This response may result in a local adaptation (e.g. restrain the resources of another VM in the same node, or change the GC algorithm to consume less memory but eventually taking more time to execute).

Figure 2 presents a declarative policy to be used by VM instances represented in Figure 1 (i.e.  $VM_{1..5}$ ). It defines limits for CPU usage, and the number of threads and sockets the application is allowed to use. CPU usage and threads are monitored and managed by specific rules but using a similar, reusable approach: i) CPU usage is monitored with a sliding window in order to filter irrelevant peaks, while ii) the number of active threads is also monitored with a sliding window in order to trigger rescheduling only when the limit is consistently exceeded.

### 3 Implementation

Some of the building blocks of  $A^2$ -VM's architecture are partially available in the research community but do not operate in an ensemble. Nevertheless, although some essential functionalities needed in our architecture are missing,

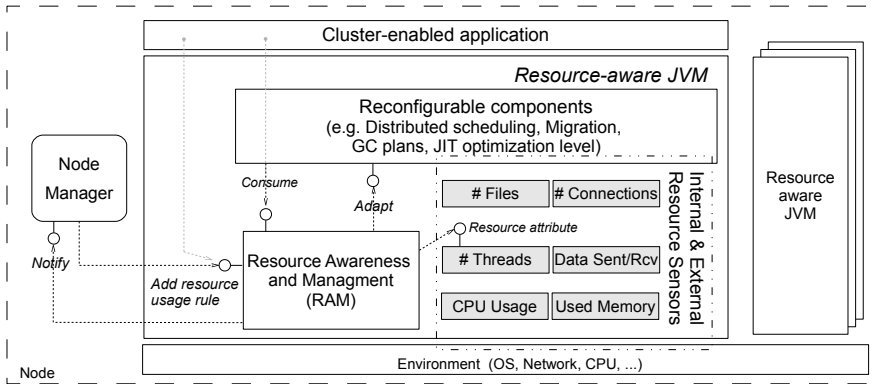


Fig. 3. Resource-aware JVM

the available components constitute a good starting point we can leverage and extend with our own work.

Our first implementation effort is centered on developing a managed language virtual machine with the capacity to monitor and restraint the use of resources based on a dynamic policy, defined declaratively outside the VM. Some work has been done in the past aiming to introduce resource-awareness in such high level virtual machines (details in Section 5). Nevertheless, to the best of our knowledge, none of them is publicly available or usable with popular software, operating systems and hardware architectures. Based on this observation, we have chosen to extend the JikesRVM [1] to be resource-aware. Thus, in the next subsection we will describe different aspects of our current work on JikesRVM. Later on, we describe the main implementation aspects of our system regarding the spawning and scheduling of threads in other nodes.

### 3.1 Extending JikesRVM with Resource-Awareness

Figure 3 depicts further details on the architecture of the resource-aware VM we developed for A<sup>2</sup>-VM. The resource-ware VM has a specific module for each type of manageable resource (e.g., files, threads, CPU usage, connections, bandwidth, and memory). Each of the module exports to the RAM (Resource Awareness Management) module an *attribute* that abstracts the specifics of the resource. This way, when the RAM decides to limit, reduce or block the usage of a resource by the application, it can instruct the respective attribute without worrying about the details of applying limitation to that specific resource (e.g., disallowing file open, or take a thread out of scheduling). The RAM consumes profile information from the main VM and A<sup>2</sup>-VM mechanisms (GC and JIT level, and distributed scheduling and migration, respectively). These mechanisms can be adapted and reconfigured by command of the RAM.

Being RAM the *engine* that enables awareness and adaptation, all its decisions are carried out according to the evaluation of rules in the policies loaded by the

```

public class ThreadsCreationNode implements Notification {
    long _threshold;
    public ThreadsCreationNode(long threshold) { _threshold = threshold; }
    public void postConsume(
        ResourceDomain domain,
        long previousUsage, long currentUsage) {
        if (currentUsage >= _threshold)
            Scheduler.getInstance().changeAllocationToCluster();
    }
}

```

**Fig. 4.** A sample notification handling to change thread allocation to the cluster

node manager. The node manager is also notified by the RAM, in each VM, about the application's performance and outcome of RAM's decisions.

The management of a given resource implies the capacity to monitor its current state and to be able to directly or indirectly control its use and usage. The resources that can be monitored in a virtual machine can be either specific of the runtime (e.g. number of threads, number of objects, amount of memory) or be strongly dependent in the operating system (e.g. CPU usage). To unify the management of such disparate types of resources, we carried out the implementation of JSR 284 - The Resource Management API [12] in the context of JikesRVM, previously not implemented in the context of any widely usable virtual machine.

The relevant elements to resource management as prescribed by JSR 284 are: *resources*, *consumers* and *resource management policies*. Resources are represented by their attributes. For example resources can be classified as *Bounded* or *Unbounded*. *Unbounded* resources are those that have no intrinsic limit (or if it exists, it is large enough to be essentially ignored) on the consumption of the resource (e.g. number of threads). The limits on the consumption of unbounded resources are only those imposed by application-level resource usage policies. Resources can also be *Bounded* if it is possible to reserve a priori a given number of units of a resource to an application. A *Consumer* represents an executing entity which can be a thread or the whole VM. Each consumer is bound to a resource through a *Resource Domain*. *Resource domains* impose a common resource management policy to all *consumers* registered. This policy is programmable through callback functions to the executing application. Although *consumers* can be bound to different *Resource Domains*, they cannot be associated to the same *resource* through different *Domains*. When a resource is about to be consumed, the resource-aware VM, implementing JSR 284, delegates this decision, via a callback, that can be handled by RAM, and either allowed, delayed or denied (with an exception thrown).

Figure 4 shows a notification, `ThreadsCreationNode`, which can be used to configure an  $A^2$ -VM instance. This callback would be called on each local thread allocation (in JikesRVM, Java threads are backed by a native class, `RVMThread` that is shown, and that interacts with the host OS threads). It determines that if the number of threads created in the local node reaches a certain threshold new threads will be created elsewhere in the cluster. The exact node where they will be placed is left to be determined by the distributed scheduler own policy.



```

public class HistoryAverage implements Constraint {
    ...
    long[] _samplesHistory;
    public HistoryAverage(int wndSize, long maxConsumption) { ... }
    public long preConsume(ResourceDomain domain,
        long currentUsage, long proposedUsage) {
        long average = 0;
        if (_nSamples == _samplesHistory.length) {
            average = _currentSum / _nSamples;
            _currentSum -= _samplesHistory[_idx];
        }
        else { _nSamples += 1; }
        _currentSum += proposedUsage;
        _samplesHistory[_idx] = proposedUsage;
        _idx = (_idx + 1) % _samplesHistory.length;
        return average > _maxConsumption ? 0 : proposedUsage;
    }
}

```

Fig. 5. Regulate consumption based on past *wndSize* observations

Figure 5 shows a *constraint*, `HistoryAverage`, which can be used to regulate a CPU usage policy. Consider a scenario where the running application cannot use the CPU above a threshold for a given time window, because the remaining CPU available is reserved for another application (e.g., as part of the quality-of-execution awarded to it). In this case, when the CPU usage monitor evaluates this rule, it would suspend all threads (i.e. return 0 for the allowed usage) if the intended usage is above the average of the last `wndSize` observations. A practical case would be to suspend the application if the CPU usage is above 75% for more than 5 observations.

### 3.2 Cluster-Wide Cooperative Thread Scheduling

In A<sup>2</sup>-VM, to achieve distributed thread scheduling, we need to be able to spawn threads in a node different from where the thread's start method is invoked.

Our mechanism to distribute threads among the cluster is built by leveraging and extending the Terracotta [6] Distributed Shared Objects. This middleware uses the client/server terminology and calls the application JVMs that are clustered together Terracotta *clients* or *Terracotta cluster nodes*. These clients run the same application code in each JVM and are clustered together by injecting cluster-aware bytecode into the application Java code at runtime, as the classes are loaded by each JVM. This bytecode injection mechanism is what makes Terracotta transparent to the application. Part of the cluster-aware bytecode injected causes each JVM to connect to the Terracotta server instances. In a cluster, a Terracotta server instance handles the storage and retrieval of object data in the shared clustered virtual heap. The server instance can also store this heap data on disk, making it persistent just as if it were part of a database. Multiple terracotta server instances can exist as a cohesive array.

In a single JVM, objects in the heap are addressed through references. In the Terracotta clustered virtual heap objects are addressed in a similar way, through references to clustered objects which we refer to as distributed shared objects or

managed objects in the Terracotta cluster. To the application, these objects are just like regular objects on the heap of the local JVMs, the Terracotta clients. However Terracotta knows that clustered objects need to be handled differently than regular objects. When changes are made to a clustered object, Terracotta keeps track of those changes and sends them to all Terracotta server instances. Server instances, in turn, make sure those changes are visible to all the other JVMs in the cluster as necessary. This way, clustered objects are always up-to-date whenever they are accessed, just as they are in a single JVM. Consistency is assured by using the synchronization present in the Java application (with monitors), which turns into Terracotta transaction boundaries. Piggybacked on these operations, Terracotta injects code to update and fetch data from remote nodes at the beginning and end of these transactions.

Therefore we need to perform additional byte-code enhancement on application classes as a previous step to the byte-code enhancing performed by the Terracotta cluster middleware before applications are run. To do this we used the ASM framework [7]. Creation of threads in remote nodes is a result of invoking JSR 284 in order to attempt to consume a thread resource at that node. The most intricate aspects deal with the issue of enforcing thread transparency (regarding its actual running node) and identity across the cluster, as we explain next.

The instrumentation replaces Java type opcodes that have the Java Thread type as argument with equal opcodes with our custom type ClusterThread. It also replaces the `getField` and `getStatic` opcodes type with ClusterThread instead of Thread. As the ClusterThread class extends the original Java Thread class, type compatibility is guaranteed. For the method calls, some of the methods belonging to the Thread class are final, and therefore cannot be overridden. To circumvent this, we aliased the final methods and replaced Thread method calls with the aliased method. For example, if we have an `invokevirtual` opcode that invokes the final “join” method of the Thread class, we invoke the “clusterJoin” method instead.

In Identity mode, the instrumentation process adds the `AutoLockWrite` Terracotta’s annotation, in order to take advantage of the local synchronization (Java monitors) to add a Terracotta transaction in every method. In Full SSI mode, we apply ‘getters and setters’ instrumentation, in order to add synchronization at its lowest level, on field access and array writes. We transform individual get and set operations into invocations to synchronized methods, automatically generated, that perform the equivalent (now synchronized) get and set operation.

Therefore, for adding getters, we implemented an ASM class adapter transformation that adds a getter for each non-static field. Each getter has the Java `synchronized` method modifier and is annotated with the `AutoLockRead` annotation to allow for concurrent reads of the field, but still in the context of a Terracotta transaction. For generating setters, we implemented a similar class adapter, with the corresponding `AutoLockWrite` annotation. We also developed equivalent instrumentations for static fields.

To use the getters and setters generated, we developed a method adapter that replaces direct field accesses with method calls. As such, the method adapter replaces the `getField` and `putField` instructions with `invokevirtual` instructions that will invoke the generated corresponding getters and setters. Equivalent `getStatic` and `putStatic` instructions will be replaced by `invokestatic` instructions that will invoke the corresponding static getters and setters. In array access, writes using array store instructions also need synchronization at some point, if the array is in shared object space. Considering this scenario, we developed a new class with static methods that consumes exactly the same arguments and performs the array store inside a synchronized block. Our method adapter will then replace the array store instruction by an invocation of the method corresponding to the data type.

## 4 Evaluation

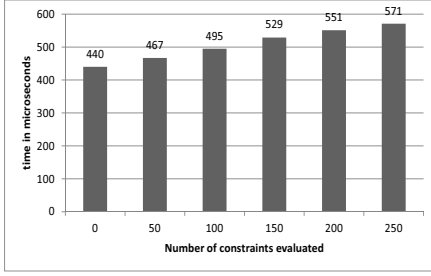
In this section we are going to describe the methodology used for evaluating the A<sup>2</sup>-VM prototype, and its results. We used up to three machines in a cluster, with Intel(R) Core(TM)2 Quad processors (with four cores each) and 8GB of RAM. Each machine was running Linux Ubuntu 9.04, with Java version 1.6.0\_16 and JikesRVM base code version 3.1.1, Terracotta Open Source edition, version 3.3.0, and multi-threaded Java applications that have the potential to scale well with multiple processors, taking advantage of the extra resources available in terms of computational power and memory.

### 4.1 Policy Evaluation and Resource Monitoring

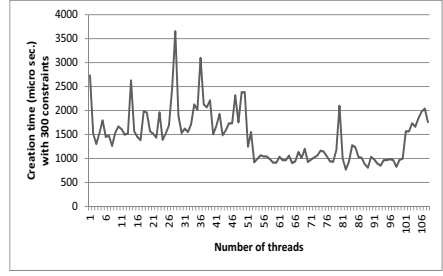
The first part of our performance evaluation regards the resource-aware VM and its impact on rules' evaluation during regular VM operations. Therefore we conducted a series of tests, measuring different aspects of a running application, starting from: i) the overhead introduced in the consumption of a specific resource, to ii) the overhead of our JSR 284 implementation, and to iii) policy evaluation in a complete benchmark scenario. All these evaluations are made locally in a single modified JikesRVM (version 3.1.1), compiled with the `production` profile.

In Figure 6a we can observe the evolution of the overhead introduced to thread creation, by measuring average thread creation and start time, as the policy engine has increasingly larger numbers of rules to evaluate, up to 250 (simulating a highly complex policy). The graph shows that this overhead, while increasing, does not hinder scalability as it is very small, ranging around 500 microseconds.

In Figure 6b we evaluate whether resource monitoring and policy evaluation (with 200 constraints) introduce any kind of performance degradation as more and more threads are created, resources consumed. Figure 6b clearly shows (omitting Garbage Collection spikes) that thread creation time does not degrade during application execution, being around 1 millisecond; although subject to some variation, it presents no lasting degradation.



(a) Thread creation time with increasing number of constraints to evaluate



(b) Thread creation time during execution with 200 constraints (GC spikes omitted)

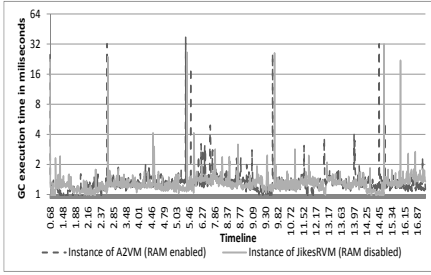
**Fig. 6.** Policy evaluation cost

The previous results were obtained monitoring only a single resource, i.e. number of application threads. For other counted resources, e.g. number of bytes sent and received, similar results are expected. Although the allocation of new objects can also be seen as a counted resource, e.g. number of bytes allocated in heap, it is more efficient to evaluate it differently. The cost of checking for constraints regarding object allocation was thus transferred to the garbage collection process, leaving the very frequent allocation operation free of additional verifications.

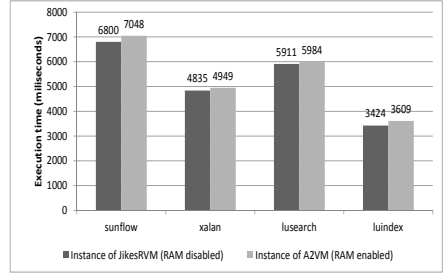
Figure 7a presents the duration of each GC cycle during the execution of Da-Capo’s benchmark [5]. `lusearch`, with and without evaluating constraints on heap consumption (i.e. RAM enabled and disabled). The `lusearch` benchmark was configured with a `small` data set, one thread for each available processor (i.e. four threads) and the convergence option active, resulting in some extra warm up runs before the final evaluation. Because of the generational garbage collection algorithm used in our modified JikesRVM, we can observe many small collection cycles, interleaved with some full heap transversal and defragmentation operations. The two runs share approximately the same average execution time and a similar average deviation:  $1.38 \pm 0.31ms$  and  $1.38 \pm 0.27ms$ , where the former value is when the RAM module is enabled and the last when RAM is disabled. With these results we conclude that performance of object allocation and garbage collection is not diminished with the extra work introduced.

To conclude the evaluation of the RAM module, we stressed an instance of our resource-aware VM with four macro benchmarks, as presented in Figure 7b. These four benchmarks are multi-threaded applications, which allows us to do a macro evaluation of the proposed modifications. During the execution of these benchmarks there were three resources being monitored (and eventually constrained): the number of threads, the total allocated memory and the CPU usage. The constraints used in evaluation did not restrain the usage of resources so that the benchmarks could properly assess the impact of monitoring different resources simultaneously in real applications (as opposed to the specific

<sup>3</sup> The version 9.12 used in the evaluation of  $A^2$ -VM’s RAM module is available at <http://www.dacapobench.org/>



(a) GC execution time during Dacapo's LuSearch benchmark



(b) Four Dacapo's multi threaded benchmarks with RAM enabled and disabled

**Fig. 7.** Macro evaluation of an instance of A<sup>2</sup>-VM

benchmarks presented previously in Figures 6(a) and 6(b). The results show only a negligible overhead: 3% in average.

## 4.2 Cooperative Scheduling

For micro-benchmarking purposes, we developed two sample applications (Fibonacci, Matrix by vector multiplication).

The Fibonacci application computes Fibonacci numbers using Binet's Fibonacci number formula. It takes the maximum number of Fibonacci to compute, along with the number of threads, and splits the workload by having each thread compute a number of Fibonacci numbers corresponding to the maximum given divided by the number of threads. For the execution time measurements, we configured our application to compute the first 1200 numbers of the Fibonacci sequence, with a number of threads directly proportional to the number of threads available. Also, we tested our application using only the Terracotta middleware, to have a general idea of how the usage of the original Terracotta platform impacts the performance (this is the price to pay for the memory scalability and elasticity it provides).

We considered two different scenarios for the tests: **Terracotta Inst. only** and **Terracotta Inst + Sharing**. The former tested the application with only the Terracotta bytecode instrumentations activated, while the latter also shared the same data structures shared in the Identity and Full SSI modes. Finally, we tested our application in a standard local JVM, for comparison purposes with our distributed solution. The results are presented in Figure 8 (note that results for 2 and 4 threads refer to execution on a single quad-core node). As we can observe in the graph, the overhead introduced is not much, as we only share a relatively small array in each thread for storing the Fibonacci numbers, along with some auxiliary variables. By adding our middleware, we introduce an extra overhead which is not very significant, even when running it in Full SSI mode and as such, it is possible to obtain smaller execution times by adding more nodes to the Terracotta cluster.

We also developed a multi-threaded application that multiplies a matrix by a vector, splitting the matrix rows across the threads. For the execution time measurements, we tested our application by multiplying a matrix of 32768 rows by 32768 columns and a vector of 32768 positions. As with previous applications, we ran the matrix by vector multiplication with no more than one thread per processor and measured the time taken by each mode with two, four, eight and twelve processors. We also tested our application in a standard local JVM, for comparison purposes with our distributed solution. The results for Identity and Full SSI mode are presented in Figure 9. Recall that distributed scheduling is only used for threads above 4; and that local execution without Terracotta (although not scalable w.r.t. memory and CPU) naturally beats local execution with Terracotta instrumentation in the limited scenario of a single-node.

As we can observe in the graph, the Terracotta bytecode instrumentations adds a small overhead, even when we do not share any data in the DSO. By adding the same data structures that are shared in both Identity and Full SSI modes, the execution times of the application in Terracotta for two and four threads are very similar to the ones presented by Identity mode, for the same number of threads. Therefore, we can obtain better execution times by using the extra processors. The Full SSI mode adds a very significant overhead (albeit only necessary for applications that are not explicitly synchronized, probably a minority), having execution times much greater than any of its counterparts, as every write in an array of results needs to be propagated to the Terracotta Server.

## 5 Related Work

Monitoring low level aspects of a computer system regarding the execution of a given application must be done with low impact in the overall application's performance. For runtime mechanisms, Price et al. [20] describes a method for modifying the garbage collector to measure the amount of live memory reachable from each group of threads. Their implementation is also based on an older version of JikesRVM but the algorithms proposed could be applied to our system and further extended (i.e. the work presented in [20] does not support all tracing collectors). They give some usage scenarios for the information accounted, but leave as an open issue the building of a policy driven framework.

Some system exchange low level precision and additional overhead for the sake of portability. Binder's profiling framework [4] statically instruments the core runtime libraries, and dynamically instruments the rest of the code. The instrumented code periodically calls pure Java agents to process and collect profiling information.

Some high level virtual machines have been augmented or designed from scratch to integrate resource accounting [11,21,3,10]. MVM [10] is based on the Hotspot virtual machine. It supports isolated computations, akin to address spaces, to be made in the same instance of the VM. This abstraction is called *isolate*. Another distinguishing characteristic is the capacity to impose constraints

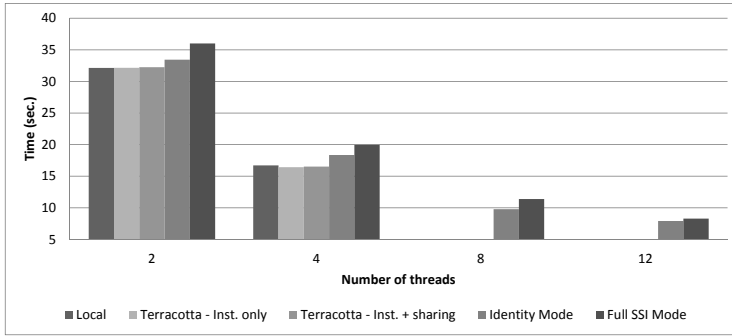


Fig. 8. Fibonacci - Execution times

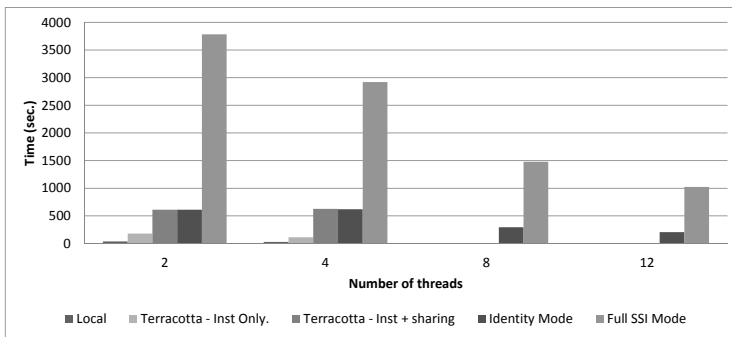


Fig. 9. Matrix\*Vector - Execution times

regarding consumption of *isolates*. MVM resource management work is related to the Java Specification Request 284 [12]. Our work builds on this JSR and uses a widely accessible VM. MVM only runs on Solaris on top of SPARC's hardware. The work in [21] and [3] enables precise memory and CPU accounting. Nevertheless they do not provide an integrated interface to determine the resource consumption policy, which may involve VM, system or class library resources.

Cluster-aware high language virtual machines have been a topic of interest for some time. They generically address three main problems: the resource monitoring problem, the migration of workload and a global address reference space. The architecture presented in [9] federates the multi-task virtual machine [10], forming a cluster where there are local and global resources that can be monitored and constrained. However, Czajkowski's work lacks the capacity to relocate workload across the cluster. Regarding policies, their's are only defined programmatically and cannot be changed without recompiling the programs/libraries responsible by clustering mechanisms (e.g. load balancer).

The Jessica VM thread migration schemes have recently been improved to take into account the dependency between threads [16]. To preserve locality of objects, a stack-based mechanism is proposed to profile the set of objects which are tightly

coupled with a migrant thread. The mechanisms and algorithms presented in this work can be explored in our system to determine the node where to spawn new threads. Moreover, by leveraging the support for a distributed shared object space in  $A^2-VM$ , thread migration needs not know in advance, with so fine-grained detail, which objects are more tightly coupled with a thread, as they can be fetched later on when accessed again.

In [22], Zhang et al. present VCluster, a thread migration middleware addressing both tightly coupled shared-memory multiprocessors and loosely coupled distributed memory multiprocessors. Their work focus on thread inter-communication and migration mechanisms. To use the VCluster middleware, the programmer must explicitly define what is the high-level thread state, relevant to be migrated to other node. In our work, the application source code does not need to be modified.

Grid systems have also been designed to take into account each node's own resources and task requirements. The work in [8] employees a multi-layer (CPU, node and site) set of reconfiguration strategies to dynamically adapt grid users' jobs to changes in the available CPU resources at each node. This adaptation is focused solely on scheduling the task to a different node, but once the task is scheduled, no further adaptation is possible. The task, and all its comprised threads, are run until completion on the same node. Our research also aims to dynamically adapt the runtime parameters and/or algorithms activated at the virtual machine in each node. Furthermore, resource monitoring is carried out during task execution and its threads can be spawned on less loaded nodes in the cluster.

## 6 Conclusion

In this document we described the architecture, implementation issues, and evaluation of  $A^2-VM$ , a research effort to design a cooperative Java virtual machine, to be deployed on clusters, able to manage resources autonomously and adaptively. It aims at offering the semantics of distributed execution environment transparently across clusters, each executing an instance of an extended resource-aware VM for the managed language Java.

Semantically, this execution environment provides a partitioned global address space where an application uses resources in several nodes, where objects are shared, and threads are spawned and scheduled globally. Regarding its operation,  $A^2-VM$  resorts to a policy-driven adaptability engine that drives resource management, global scheduling of threads, and determines the activation of other coarse-grained mechanisms (e.g., checkpointing and migration among VMs).

In summary, the goal of such an infrastructure is to provide more flexibility, control, scalability and efficiency to applications running in clusters.

**Acknowledgement.** This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds.



## References

1. Alpern, B., Augart, S., Blackburn, S.M., Butrico, M., Cocchi, A., Cheng, P., Dolby, J., Fink, S., Grove, D., Hind, M., McKinley, K.S., Mergen, M., Moss, J.E.B., Ngo, T., Sarkar, V.: The jikes research virtual machine project: building an open-source research community. *IBM Syst. J.* 44, 399–417 (2005)
2. Aridor, Y., Factor, M., Teperman, A.: cJVM: a single system image of a JVM on a cluster. In: *Proceedings of the International Conference on Parallel Processing*, pp. 4–11 (1999)
3. Back, G., Hsieh, W.C., Lepreau, J.: Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. In: *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pp. 333–346 (2000)
4. Binder, W., Hulaas, J., Moret, P., Villazón, A.: Platform-independent profiling in a virtual execution environment. *Softw. Pract. Exper.* 39, 47–79 (2009)
5. Blackburn, S.M., Garner, R., Hoffmann, C., Khang, A.M., McKinley, K.S., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A., Jump, M., Lee, H., Moss, J.E.B., Moss, B., Phansalkar, A., Stefanović, D., Van Drunen, T., von Dincklage, D., Wiedermann, B.: The DaCapo benchmarks: Java benchmarking development and analysis. In: *OOPSLA 2006: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 169–190. ACM, New York (2006)
6. Boner, J., Kuleshov, E.: Clustering the Java Virtual Machine using Aspect-Oriented Programming. In: *AOSD 2007: Industry Track of the 6th international conference on Aspect-Oriented Software Development. Conference on Aspect Oriented Software Development* (March 2007)
7. Bruneton, E., Lenglet, R., Coupaye, T.: ASM: a Code Manipulation Tool to Implement Adaptable Systems. In: *Proceedings of the ASF (ACM SIGOPS France) Journées Composants 2002: Systèmes à composants adaptables et extensibles (Adaptable and Extensible Component Systems)* (November 2002)
8. Chen, P.-C., Chang, J.-B., Liang, T.-Y., Shieh, C.-K.: A progressive multi-layer resource reconfiguration framework for time-shared grid systems. *Future Gener. Comput. Syst.* 25, 662–673 (2009)
9. Czajkowski, G., Wegiel, M., Daynes, L., Palacz, K., Jordan, M., Skinner, G., Bryce, C.: Resource management for clusters of virtual machines. In: *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid, CC-GRID 2005*, vol. 01, pp. 382–389. IEEE Computer Society, Washington, DC, USA (2005)
10. Czajkowski, G., Hahn, S., Skinner, G., Soper, P., Bryce, C.: A resource management interface for the Java platform. *Softw. Pract. Exper.* 35, 123–157 (2005)
11. Czajkowski, G., von Eicken, T.: JRes: a resource accounting interface for Java. In: *Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA 1998*, pp. 21–35. ACM, New York (1998)
12. Czajkowski, G., et al.: Java specification request 284 - resource consumption management api (2009)
13. Garrochinho, T., Veiga, L.: CRM-OO-VM: a checkpointing-enabled Java VM for efficient and reliable e-science applications in grids. In: *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science, MGC 2010*, pp. 1:1–1:7. ACM, New York (2010)

14. Gront, D., Kolinski, A.: Utility library for structural bioinformatics. *Bioinformatics* 24(4), 584–585 (2008)
15. Holland, R.C.G., Down, T.A., Pocock, M.R., Prlic, A., Huen, D., James, K., Foisy, S., Dräger, A., Yates, A., Heuer, M., Schreiber, M.J.: Biojava: an open-source framework for bioinformatics. *Bioinformatics* 24(18), 2096–2097 (2008)
16. Lam, K.T., Luo, Y., Wang, C.-L.: Adaptive sampling-based profiling techniques for optimizing the distributed JVM runtime. In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS 2010)*, pp. 1–11 (April 2010)
17. López-Arévalo, I., Bañares-Alcántara, R., Aldea, A., Rodríguez-Martínez, A.: A hierarchical approach for the redesign of chemical processes. *Knowl. Inf. Syst.* 12(2), 169–201 (2007)
18. Microsoft. CLR Profiler for the .NET framework 2.0 (2007),  
<http://www.microsoft.com/download/en/details.aspx?id=13382>
19. Oracle. Java virtual machine tool interface (JVMTI),  
<http://download.oracle.com/javase/6/docs/technotes/guides/jvmti/>
20. Price, D.W., Rudys, A., Wallach, D.S.: Garbage collector memory accounting in language-based systems. In: *Proceedings of the IEEE Symposium on Security and Privacy SP 2003*, pp. 263–274. IEEE Computer Society, Washington, DC, USA (2003)
21. Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., Saavedra, R.: State capture and resource control for java: the design and implementation of the aroma virtual machine. In: *Proceedings of the 2001 Symposium on Java™ Virtual Machine Research and Technology Symposium, JVM 2001*, vol. 1, pp. 11–11. USENIX Association, Berkeley (2001)
22. Zhang, H., Lee, J., Guha, R.: Vcluster: a thread-based java middleware for smp and heterogeneous clusters with thread migration support. *Softw. Pract. Exper.* 38, 1049–1071 (2008)
23. Zhu, W., Wang, C.-L., Lau, F.C.M.: Jessica2: A distributed java virtual machine with transparent thread migration support. In: *IEEE International Conference on Cluster Computing*, p. 381 (2002)

# Peer-Based Relay Scheme of Collaborative Filtering for Research Literature

Youliang Zhong<sup>1</sup>, Weiliang Zhao<sup>1</sup>, Jian Yang<sup>1</sup>, and Lai Xu<sup>2</sup>

<sup>1</sup> Department of Computing, Macquarie University  
North Ryde, NSW 2109, Australia

{youliang.zhong, weiliang.zhao, jian.yang}@mq.edu.au

<sup>2</sup> Software Systems Research Centre, Bournemouth University  
Fern Barrow, Poole, Dorset BH12 5BB, UK  
lxu@bournemouth.ac.uk

**Abstract.** Much work has been done in both industry and academia on filtering for research literature, however most existing studies have their limitations in coping with the inherent characteristics of research literature search, i.e., most articles attract very few readers among all the researchers, and the recommendations are often circulated through members of particular communities. In this paper we propose a peer-based relay scheme of collaborative filtering for, but not limited to research literature. In the scheme, a recommendation request is relayed through a social structure dynamically formed by *co-peers with common interests*, and the recommendation results are adjusted and propagated by the *co-peers*. A hybrid filtering approach is deployed in the scheme.

**Keywords:** Recommendation relay scheme, Collaborative filtering, Social networks, Research literature.

## 1 Introduction

With the development of Web 2.0 technology, collaborative filtering for research literature has attracted great interests in both industry and academia. The current researches in the field have their limitations in addressing the special issues of recommending research articles. Unlike mass media products such as movies and TV programs that are watched by millions of people, a research article is often read by much less people. An empirical study showed that the average citations of a scientific paper was 11.9 in 1997, and the number was decreasing with time [13]. It is also observed that researchers seldom put research articles and associated ratings in public social sites although they do normally maintain collections of articles by means of various tools.

When a researcher wants to expand her collections in a particular research area, she will ask her colleagues to give recommendations of a specific topic, and her colleagues may continue with the requests to their associates and then send the results back to the researcher. This is a relay process for getting personalized recommendations through social networks. Now the challenge is how we can

follow a similar way in collaborative filtering methods. Two critical issues have to be addressed: (1) how shall a user choose the peers to get their recommendations, based on similar taste or rating style? (2) how can the involved peers aggregate the recommendations from others who have various opinions on same or similar items?

In this paper, we propose a novel filtering scheme, which is based on a peer structure dynamically formed by "peers with common interests" in a social network. All the peers relay recommendation requests and responses, and each peer aggregates and filters recommended items. The peer structure, the hybrid filtering algorithms, and the relay process will be discussed in the paper. The main contributions of the paper are as follows:

- **Relay based collaborative filtering.** This paper proposes a collaborative filtering scheme by explicitly utilizing social relationships, which is based on a dynamically established social structure, and the recommendation process is carried out in a relay fashion over the structure. Such a relay approach is coincident with the human behaviour of making recommendations in social or professional life.
- **Peer-based recommendation.** In the proposed scheme, every user works on its own and collaborates with its peers to expend recommendations. This is in contrast with most existing recommendation approaches that collect raw data from all users and make recommendations at a central place. Furthermore, the recommendation results are adjusted by direct and indirect co-peers during a recommendation process.

The rest of the paper is organized as follows. We firstly present a motivating example and notations in section 2, and subsequently elaborate the peer-based relay scheme in section 3. Experiments are illustrated in section 4, and related work is discussed in section 5, followed by a concluding remark in section 6.

## 2 Preliminaries

### 2.1 Motivating Example

Fig 1 illustrates how recommendations are produced in a peer-based social network by a relay mechanism, in which each user maintains a list of items and associated ratings. For instance, user *Alex* possesses a list of item "*a, c, e*", with ratings "*8, 7, 3*". *Alex* has commonly rated item *c* with *John*, and item *e* with *Peter*, however there is no overlap with *Eddy*. We call an item like *c* or *e* as a co-rated-item (CRI), and friends who have CRIs as co-peers.

Suppose user *Alex* wants to expand his collection, he sends a request to his co-peers, and the co-peers may forward the request to their co-peer and so forth. Therefore, an initial requestor and all its successive co-peers form a Co-Peer Graph, and every user in the graph will be an active recommendation engine. Finally, after getting recommendations from its co-peers, *Alex* aggregates and filters the recommendations to create a final recommendation list for himself.

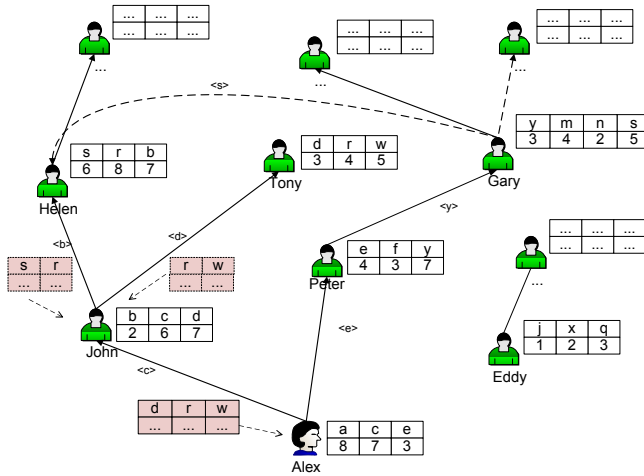


Fig. 1. Peer-structured recommendation process

## 2.2 Notations of Co-peer Graph

A co-peer graph is defined as a labeled directed acyclic graph  $CPG(V, A)$ , where  $V$  is a set of vertices and  $A$  a set of directed arcs over  $V$ , such that,

- 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  $R(v) = \{v(t_i) \mid i = 1..n\}$  is the set of associated ratings over  $T(v)$ .
- For a pair of vertices  $u, v \in V$ , the set intersection of  $T(u) \cap T(v)$  is denoted as  $C_u^v$ , referred to co-rated-items (CRI) between  $v$  and  $u$ . If  $C_u^v \neq \emptyset$ , then  $u$  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  $Icp(v) = \{u\}$ , while 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, for those outbound co-peers of  $v$  that commonly rate on a particular item  $t_i (t_i \notin T(v))$  are 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 Collaborative Relay Scheme

### 3.1 Collaborative Prediction Algorithms

Considering a pair of co-peers  $u$  and  $v$ , with co-rated-items  $C_u^v$ , and  $u$  is the inbound co-peer of  $v$ . For an item  $t_i \in P_u^v$ , where  $P_u^v = T(v) \setminus T(u)$  indicating the set of "potential item" from  $v$  for  $u$ , we want to predict the rating for the item  $t_i$  from  $v$  for  $u$ , 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 of  $t_i$  made by  $v$ , and  $b$  an adjustment constant. According to Minimum Mean Square Error, we have  $E = \sum_{t_j \in C_u^v} (v(t_j) + b - u(t_j))^2$ , and need to get a suitable  $b$  for a minimized  $E$ . Put these together, we have

$$\begin{aligned}
 r_u^v(t_i) &= v(t_i) + b, \text{ where } t_i \in P_u^v, \quad b = \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} (u(t_j) - v(t_j)), \\
 \text{so, } r_u^v(t_i) &= \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} (v(t_i) - v(t_j)) + \bar{R}_v(u), \\
 \text{where } \bar{R}_v(u) &= \frac{1}{|C_u^v|} \sum_{t_j \in C_u^v} u(t_j).
 \end{aligned} \tag{1}$$

From the standing point of an inbound co-peer, user  $u$  may have multiple outbound co-peers which have made predictions for the user on a same potential item say  $t_i$ , the user then needs to aggregate all the coming predicted ratings of  $t_i$ , described as follows (formula 2).

$$\begin{aligned}
 r_u(t_i) &= \frac{1}{|Ocp_i(u)|} \sum_{v \in Ocp_i(u)} r_u^v(t_i). \\
 T(u) &= T^e(u) \cup T^r(u), \text{ where } T^r(u) = \cup_{v \in Ocp(u)} P_u^v, \\
 R(u) &= R^e(u) \cup \{r_u(t_j) \mid t_j \in T^r(u)\}.
 \end{aligned} \tag{2}$$

In the above formula,  $r_u(t_i)$  is the aggregated rating of  $t_i$  for  $u$ , and  $Ocp_i(u)$  the set of outbound co-peers who commonly rate on item  $t_i$ .  $T^e(u)$  and  $R^e(u)$  stand for existing items and ratings possessed by user  $u$ , and  $T^r(u)$  and  $R^r(u)$  the recommended items and predicted ratings.

### 3.2 Content-Based Relevance Probability Formulas

Owing to the distinct feature of research literature, it is very natural to filter the articles by relevance measurement between articles and a user’s possessed items or/and profile [1]. In our scheme, we adopt the concepts of "local- and document-wide relevance" proposed by [17], and use an equivalent version of [18] as follows (formula 3), where  $W(u)$  represents the words extracted from the Profiles of a peer user  $u$  and a root user  $v_0$ , and  $W(t)$  the words from the Abstract of an article  $t$ . Function  $f(w, t)$  stands for the occurrences of a word  $w$  in the article  $t$ . The cardinality  $|T(u)|$  is the number of the articles possessed by user  $u$ , and  $|T_w(u)|$  the subset of  $|T(u)|$  of which all articles contain the word  $w$ .

$$\rho(u, t) = \sum_{w \in (W(u) \cap W(t))} f(w, t) \log \frac{|T(u)|}{|T_w(u)|}. \tag{3}$$

Having calculated the prediction ratings and relevance probabilities of potential items, two hybrid methods are designed in the proposed scheme, one is *Weighted* and the other *Mixed*, as classified by [5]. The *Weighted* method takes a form of  $\alpha * rating + \beta * relevance$  where  $\alpha + \beta = 1.0$ , and the *Mixed* one uses a logic *OR/AND* operator, that is, either  $\alpha * rating$  *OR/AND*  $\beta * relevance$  becomes a measurement. Finally, we denote a set of recommended items from  $v$  for  $u$  as  $\tilde{P}_u^v$ , where  $\tilde{P}_u^v \subset P_u^v$ .

### 3.3 Recommendation Relay Process Sequence

Generally, a root node broadcasts a request to its outbound co-peers, the outbound co-peers forward the request further when they have their outbound co-peers. When the request reaches at leaf peers, they stop forwarding, instead make individual predictions using formula 1 and filter potential items for their inbound co-peers. If an inbound co-peer gets a desired number of recommendations from its outbound co-peers, it then aggregates predictions on recommended items using formula 2. After that, it computes individual prediction ratings using formula 1, and sends the predictions backwards to its inbound co-peer. The relay will iteratively occur until a recommendation list is produced.

To effectively relay recommendations over a co-peer graph, several issues need to be addressed. Firstly, all users need to avoid repeatedly working for a same request sending from different co-peers. In other words, we need to have a tree structure for the co-peer graph. To this end, every user keeps a cache of all executed requests so that it will reject the same requests afterwards if they have been processed.

Secondly, without constraints on request-forwarding, co-peers may keep forwarding and the relay process may never end. To control the termination of request-forwarding among co-peers, every recommendation request is issued with constraints, including the maximum number of desired responses, the expected number of layers of co-peers, and the period of timeout.

Through a full cycle of replay process, the root node in a co-peer graph will get a set of recommended items and associated prediction ratings of  $\langle T^r(v_0), R^r(v_0) \rangle$  described in the following formula (4), where  $T^r(v_0)$  represents the set of all recommended items, and  $R^r(v_0)$  the associated predicted ratings, which may be further aggregated and filtered for a final recommendation list.

$$\begin{aligned} T^r(v_0) &= \cup_{u,v \in V}^{v \in \text{Ocp}(u)} \tilde{P}_u^v, \\ R^r(v_0) &= \{r_{v_0}(t_j) \mid t_j \in T^r(v_0)\}. \end{aligned} \quad (4)$$

## 4 Experiments

### 4.1 Simulation System and Evaluation Metrics

A simulation system of the proposed scheme has been developed by using Spring WebFlow, JSF and Spring framework, and a dataset was collected from ISI Web of Knowledge database [8]. The dataset consisted of 10,228 articles covering 20 different fields of computing science and engineering, and each bibliographic item of an article had fields of title, author, publish year, abstract and keywords. 530 users were prepared, with randomly selected profiles from authors' home pages in relevant conferences. Totally 32,567 ratings are generated using a Gaussian distribution, ranging from 0 to 10.

In this paper, we use NMAE (Normalized Mean Absolute Error) [7], riMAE (Mean Absolute Error of recommended items), and Coverage [2] to measure the

personalized degree of recommendation results. In the following formulas, the superscript "e" is used for items or ratings which are originally rated by the user, and  $r_v(\cdot)$  stands for the predicted ratings generated by recommendation algorithms.  $T^r(v)$  stands for the recommended items, and  $T_l^r(v)$  the highly relevant items that have been recommended.

$$NMAE = \frac{1}{|V|} \sum_{v \in V} \frac{M_v}{R_{max}^e(v) - R_{min}^e(v)}, \quad M_v = \frac{1}{|T^e(v)|} \sum_{t_j \in T^e(v)} |r_v(t_j) - v^e(t_j)|. \quad (5)$$

$$riMAE_v = \frac{1}{|T^r(v)|} \sum_{t_j \in T^r(v)} r_v(t_j) - \frac{1}{|T^e(v)|} \sum_{t_i \in T^e(v)} v^e(t_i). \quad (6)$$

$$Coverage = \frac{1}{|V|} \sum_{v \in V} \frac{|T_l^r(v)|}{|T^r(v)|}. \quad (7)$$

### 4.2 Evaluation and Analysis

In the two pictures of Fig. 2, X-axis stands for the users having different sizes of direct and indirect co-peers from 50 to 165, and Y-axis for Coverage (%) and NMAE/riMAE values respectively. As shown, the Coverage(%) of the users with lowest number of co-peers is about 40%, however that of most co-peers rises to 74%. Likewise, while group 1 has both NMAE and riMAE values at over 0.2, other groups get the MAE values around 0.15. This indicated that the personalized degree of recommendations is higher for the users having more co-peers than those with less.

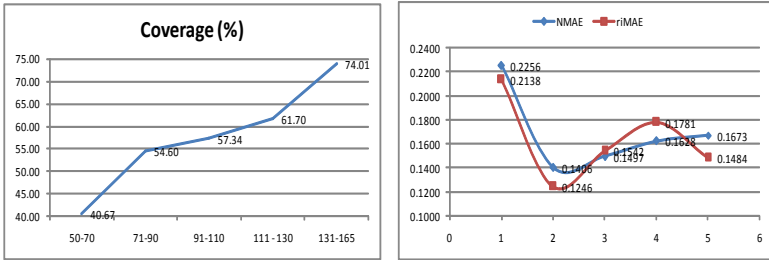


Fig. 2. Effect of co-peer size

Two pictures of Fig. 3 demonstrate how the depth of relay process affect recommendation results, in which X-axis stands for the depths of relay process from 1 to 5, and Y-axis for Coverage (%) and MAE values respectively. As showed, the Coverage increased from about 51% to near 62% when relay process went further. On the other hand, while riMAE values kept at a stable level of 0.16xx, NMAE values rose from 0.14 at depth 1 to near 0.19 at depth 5. That did show NMAE was affected by relay depth, however the NMAE was calculated on the items that were previously rated by a requesting user. It was the riMAE which measured the personalized degree of recommendation items, and it were the riMAE values which were stably staying at a low level of 0.16xx.



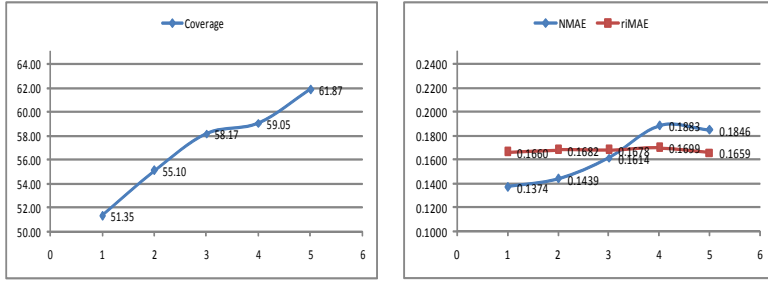


Fig. 3. Impact of relay depth

## 5 Related Work

There are basically two types of filtering methods: content-based filtering (CN) and collaborative filtering (CF). While content-based filtering recommends items to a user based on the similarities between potential items and those rated by the user [2,3], collaborative filtering is based on the ratings assigned by other people with "similar taste" [14,15]. Of the CF methods, *Slope One* is one of the commonly used prediction algorithms [10]. We leverage both Slope One method and peer-based structure in a social network so that the recommendation can be replayed and adjusted by the co-peers in the network.

To cope with peer-based collaborative filtering, researches proposed various distributed or peer-to-peer solutions, especially some of them focused on distributed data sources and distributed computation, such as [16,4]. In contrast to these studies, our peer-based relay scheme has neither a global dataset nor a centralized process, every user maintains its own data set and makes recommendations on its own.

Much work has been done on recommending research articles. The research of [11] was an important study of applying collaborative filtering to research articles by using citation information. Several other studies were proposed using personal behaviour to enrich users' personal profiles [6,9,12,19].

## 6 Conclusions

In this paper we propose a novel scheme of filtering research literature, in which a recommendation process is carried out over a peer-based social structure formed by "co-peers with similar interests", and the recommendation requests and responses are relayed and adjusted by direct and indirect peers. The scheme will be also valuable for other applications where most items attract few users, and the users prefer to circulate recommendations within peer-based groups.

As discussed in the paper, the depth of relay process and the size of co-peers significantly affect recommendation performance. In fact a user may select co-peers according to more other factors, such as the depth of social relationship, the number of potential co-peers and etc. It is also valuable to take qualitative aspects of social networks into account for instance 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. Baeza-Yates, R., Ribeiro-Neto, B., et al.: *Modern information retrieval*, vol. 463. ACM press, New York (1999)
3. 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)
4. Berkovsky, S., Busetta, P., Eytani, Y., Kuflik, T., Ricci, F.: Collaborative Filtering over Distributed Environment. In: *DASUM Workshop, Citeseer* (2005)
5. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* 12(4), 331–370 (2002)
6. Geyer-Schulz, A., Hahsler, M., Neumann, A., Thede, A.: Behavior-Based Recommender Systems as Value-Added Services for Scientific Libraries
7. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval* 4(2), 133–151 (2001)
8. ISI-WoK (2010), <http://wokinfo.com/>
9. Jung, S., Kim, J., Herlocker, J.L.: Applying collaborative filtering for efficient document search. In: *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 640–643. IEEE Computer Society (2004)
10. Lemire, D., Maclachlan, A.: Slope one predictors for online Rating-Based collaborative filtering. *Society for Industrial Mathematics* (2005)
11. McNee, S.M., Albert, I., Cosley, D., Gopalkrishnan, P., Lam, S.K., Rashid, A.M., Konstan, J.A., Riedl, J.: On the recommending of citations for research papers. In: *ACM Conference on Computer Supported Cooperative Work*, New Orleans, Louisiana, USA, pp. 116–125. ACM (2002)
12. Pohl, S., Radlinski, F., Joachims, T.: Recommending related papers based on digital library access records. In: *ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 417–418. ACM (2007)
13. Redner, S.: How popular is your paper? An empirical study of the citation distribution. *The European Physical Journal B* 4(2), 131–134 (1998)
14. 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)
15. 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)
16. Tveit, A.: Peer-to-peer based recommendations for mobile commerce. In: *International Workshop on Mobile Commerce*, Rome, Italy, pp. 26–29. ACM (2001)
17. Wu, H.C., Luk, R.W.P., Wong, K.F., Kwok, K. L.: A retrospective study of a hybrid document-context based retrieval model. *Information Processing and Management: an International Journal* 43(5), 1308–1331 (2007)
18. Wu, H.C., Luk, R.W.P., Wong, K.F., Kwok, K.L.: Interpreting TF-IDF term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)* 26(3), 1–37 (2008)
19. Zhang, Z.K., Zhou, T., Zhang, Y.C.: Personalized recommendation via integrated diffusion on user-item-tag tripartite graphs. *Physica A Statistical Mechanics and its Applications* 389, 179–186 (2010)

# Detecting and Resolving Conflicts of Mutual-Exclusion and Binding Constraints in a Business Process Context

Sigrid Schefer<sup>1</sup>, Mark Strembeck<sup>1</sup>, Jan Mendling<sup>2</sup>, and Anne Baumgrass<sup>1</sup>

<sup>1</sup> Institute for Information Systems, New Media Lab  
Vienna University of Economics and Business (WU Vienna), Austria  
`{firstname.lastname}@wu.ac.at`

<sup>2</sup> Institute for Information Business  
Vienna University of Economics and Business (WU Vienna), Austria  
`jan.mendling@wu.ac.at`

**Abstract.** Mutual exclusion and binding constraints are important means to define which combinations of subjects and roles can be assigned to the tasks that are included in a business process. Due to the combinatorial complexity of potential role-to-subject and task-to-role assignments, there is a strong need to systematically check the consistency of a given set of constraints. In this paper, we discuss the detection of consistency conflicts and provide resolution strategies for the corresponding conflicts.

**Keywords:** business processes, information systems, mutual exclusion, separation of duty, binding of duty.

## 1 Introduction

In recent years, business processes are increasingly designed with security and compliance considerations in mind (see, e.g., [3,16,19]). For example, the definition of process-related security properties is important if a conflict of interest could arise from the simultaneous assignment of decision and control tasks to the same subject. In this context, process-related access control mechanisms are typically used to specify authorization constraints, such as *separation of duty (SOD)* and *binding of duty (BOD)*, to regulate which subject is allowed (or obliged) to execute a particular task (see, e.g., [4,5,14,15,16,17,19]).

In a workflow environment, SOD constraints enforce conflict of interest policies by defining that two or more tasks must be performed by different individuals. Conflict of interest arises as a result of the simultaneous assignment of two mutual exclusive entities (e.g. permissions or tasks) to the same subject. Tasks can be defined as statically mutual exclusive (on the process type level) or dynamically mutual exclusive (on the process instance level). Thus, a *static mutual exclusion (SME)* constraint is global with respect to all process instances in an information system. Therefore, two SME tasks can never be assigned to the same subject or role. On the other hand, two *dynamically mutual exclusive*

(DME) tasks can be assigned to the same subject but must not be executed by the same subject in the same process instance.

In contrast, BOD constraints specify that *bound tasks* must always be performed by the same subject or role (see, e.g., [14,15,16,17]). BOD can be subdivided into subject-based and role-based constraints (see, e.g., [14,15]). A *subject-based BOD constraint* defines that the same individual who performed the first task must also perform the bound task(s). On the other hand, a *role-based BOD constraint* defines that bound tasks must be performed by members of the same role, but not necessarily by the same individual. Throughout the paper, we will use the terms *subject-binding (SB)* and *role-binding (RB)* as synonyms for subject-based BOD constraints and role-based BOD constraints, respectively.

In recent years, role-based access control (RBAC) [7,11] has developed into a de facto standard for access control. A specific problem in the area of process-related RBAC is the immanent complexity of interrelated mutual-exclusion and binding constraints. Thus, when defining process-related mutual-exclusion or binding constraints, design-time and runtime checks need to ensure the consistency of the corresponding RBAC model. In particular, at design-time conflicts may result from inconsistent constraints or assignment relations. At runtime conflicts may result from invalid task-to-subject allocations (see also [14]).

In this paper, we take the conflicts identified in [14] as a starting point. We adapt the algorithms from [14] to detect and name corresponding conflicts, and discuss resolution strategies for these conflicts. In particular, we consider conflicts at the level of design-time constraint definition, design-time assignment relations, and runtime task allocation.

The remainder of this paper is structured as follows. Section 2 gives an overview of process-related RBAC models and the requirements for design-time and runtime consistency of these models. Sections 3, 4, and 5 present algorithms to detect potential conflicts of mutual-exclusion and binding constraints. Furthermore, we provide resolution strategies that exemplarily show how these conflicts can be resolved to ensure the consistency of a process-related RBAC model. Subsequently, Section 6 discusses related work and Section 7 concludes the paper.

## 2 Process-Related RBAC Models

The algorithms and resolution strategies presented in Section 3, 4, and 5 are based on the formal definitions for process-related RBAC models from [14,15]. However, due to the page restrictions we cannot repeat the complete list of definitions in this paper. Therefore, we now give an overview of the definitions we use below – for further details please consult [14,15].

**Definition 1 (Process-related RBAC Model).** *A Process-related RBAC Model  $PRM = (E, Q, D)$  where  $E = S \cup R \cup P_T \cup P_I \cup T_T \cup T_I$  refers to pairwise disjoint sets of the model,  $Q = rh \cup rsa \cup tra \cup es \cup er \cup ar \cup pi \cup ti$  to mappings that establish relationships, and  $D = sb \cup rb \cup sme \cup dme$  to binding and mutual-exclusion constraints.*

An element of  $S$  is called *Subject*. An element of  $R$  is called *Role*. An element of  $P_T$  is called *Process Type*. An element of  $P_I$  is called *Process Instance*. An element of  $T_T$  is called *Task Type*. An element of  $T_I$  is called *Task Instance*.

We allow the definition of subject-binding ( $sb$ ), role-binding ( $rb$ ), static mutual exclusion ( $sme$ ), and dynamic mutual exclusion ( $dme$ ) constraints on task types. Roles can be arranged in a role-hierarchy ( $rh$ ), where more powerful senior-roles inherit the permissions from their junior-roles. The task-to-role assignment relation ( $tra$ ) defines which tasks can be performed by the members of a certain role. Thereby,  $tra$  specifies the permissions of a role. The task-ownership mapping ( $town$ ) allows to determine which tasks are assigned to a particular role – including the tasks inherited from junior-roles. The inverse mapping ( $town^{-1}$ ) returns the set of roles a task is assigned to. The role-to-subject assignment relation ( $rsa$ ) defines which roles are assigned to particular users. The role-ownership mapping ( $rown$ ) returns all roles assigned to a certain subject (including roles that are inherited via a role-hierarchy). The inverse mapping ( $rown^{-1}$ ) allows to determine all subjects assigned to a particular role. Each subject can activate the roles that are assigned to this subject, and the active-role mapping ( $ar$ ) returns the role that is currently activated. For each task instance we have an executing-subject ( $es$ ) and an executing-role ( $er$ ).

Definition 2 provides rules for the static correctness of process-related RBAC models to ensure the design-time consistency of the included elements and relationships.

**Definition 2.** Let  $PRM = (E, Q, D)$  be a *Process-related RBAC Model*.  $PRM$  is said to be *statically correct* if the following requirements hold:

1. *Tasks cannot be mutual exclusive to themselves:*  
 $\forall t_2 \in sme(t_1) : t_1 \neq t_2$  and  $\forall t_2 \in dme(t_1) : t_1 \neq t_2$
2. *Mutuality of mutual exclusion constraints:*  
 $\forall t_2 \in sme(t_1) : t_1 \in sme(t_2)$  and  $\forall t_2 \in dme(t_1) : t_1 \in dme(t_2)$
3. *Tasks cannot be bound to themselves:*  
 $\forall t_2 \in sb(t_1) : t_1 \neq t_2$  and  $\forall t_2 \in rb(t_1) : t_1 \neq t_2$
4. *Mutuality of binding constraints:*  
 $\forall t_2 \in sb(t_1) : t_1 \in sb(t_2)$  and  $\forall t_2 \in rb(t_1) : t_1 \in rb(t_2)$
5. *Tasks are either statically or dynamically mutual exclusive:*  
 $\forall t_2 \in sme(t_1) : t_2 \notin dme(t_1)$
6. *Either SME constraint or binding constraint:*  
 $\forall t_2 \in sme(t_1) : t_2 \notin sb(t_1) \wedge t_2 \notin rb(t_1)$
7. *Either DME constraint or subject-binding constraint:*  
 $\forall t_2 \in dme(t_1) : t_2 \notin sb(t_1)$
8. *Consistency of task-ownership and SME:*  
 $\forall t_2 \in sme(t_1) : town^{-1}(t_2) \cap town^{-1}(t_1) = \emptyset$
9. *Consistency of role-ownership and SME:*  $\forall t_2 \in sme(t_1), r_2 \in town^{-1}(t_2),$   
 $r_1 \in town^{-1}(t_1) : rown^{-1}(r_2) \cap rown^{-1}(r_1) = \emptyset$

Definition 3 provides the rules for dynamic correctness of a process-related RBAC model, i.e. the rules that can only be checked in the context of runtime process instances.

**Definition 3.** Let  $PRM = (E, Q, D)$  be a Process-related RBAC Model and  $P_I$  its set of process instances.  $PRM$  is said to be dynamically correct if the following requirements hold:

1. In the same process instance, the executing subjects of SME tasks must be different:  
 $\forall t_2 \in sme(t_1), p_i \in P_I : \forall t_x \in ti(t_2, p_i), t_y \in ti(t_1, p_i) : es(t_x) \cap es(t_y) = \emptyset$
2. In the same process instance, the executing subjects of DME tasks must be different:  
 $\forall t_2 \in dme(t_1), p_i \in P_I : \forall t_x \in ti(t_2, p_i), t_y \in ti(t_1, p_i) : es(t_x) \cap es(t_y) = \emptyset$
3. In the same process instance, role-bound tasks must have the same executing-role:  $\forall t_2 \in rb(t_1), p_i \in P_I : \forall t_x \in ti(t_2, p_i), t_y \in ti(t_1, p_i) : er(t_x) = er(t_y)$
4. In the same process instance, subject-bound tasks must have the same executing-subject:  $\forall t_2 \in sb(t_1), p_i \in P_I : \forall t_x \in ti(t_2, p_i), t_y \in ti(t_1, p_i) : es(t_x) = es(t_y)$

### 3 Constraint Definition Conflicts

When defining SME, DME, RB, or SB constraints at design-time, a number of conflicts may occur that would lead to inconsistencies in the corresponding process-related RBAC model. Below we first present algorithms to detect these constraint definition conflicts. If a conflict is detected, the algorithms return the name of the respective conflict. In the following subsections, we provide descriptions for each conflict type and present different resolution strategies.

#### 3.1 Algorithms for Detecting Constraint Definition Conflicts

**Algorithm 1.** Check if the definition of a new SME constraint is allowed.

**Name:** *isSMEConstraintAllowed*

**Input:**  $task_1, task_2 \in T_T$

```

1: if  $task_1 == task_2$  then return selfConstraintConflict
2: if  $task_1 \in dme(task_2)$  then return directDMEConflict
3: if  $task_1 \in allRoleBindings(task_2)$  then return RBConflict
4: if  $task_1 \in allSubjectBindings(task_2)$  then return SBCConflict
5: if  $\exists r \in R \mid r \in town^{-1}(task_1) \wedge r \in town^{-1}(task_2)$ 
6:   then return taskOwnershipConflict
7: if  $\exists s \in S \mid r_1 \in rown(s) \wedge r_2 \in rown(s) \wedge$ 
8:    $r_1 \in town^{-1}(task_1) \wedge r_2 \in town^{-1}(task_2)$ 
9:   then return roleOwnershipConflict
10: return true

```

**Algorithm 2.** Check if the definition of a new DME constraint is allowed.

**Name:** *isDMEConstraintAllowed*

**Input:**  $task_1, task_2 \in T_T$

```

1: if  $task_1 == task_2$  then return selfConstraintConflict

```

```

2: if  $task_1 \in sme(task_2)$  then return directSMEConflict
3: if  $task_1 \in allSubjectBindings(task_2)$  then return SBCConflict
4: return true

```

**Algorithm 3.** Check if the definition of a new RB constraint is allowed.

*Name:* *isRBConstraintAllowed*

*Input:*  $task_1, task_2 \in T_T$

```

1: if  $task_1 == task_2$  then return selfConstraintConflict
2: if  $task_1 \in sme(task_2)$  then return directSMEConflict
3: if  $\exists task_x \in sme(task_1) \mid task_x \in allRoleBindings(task_2)$ 
4:   then return transitiveSMEConflict
5: if  $\exists task_x \in sme(task_2) \mid task_x \in allRoleBindings(task_1)$ 
6:   then return transitiveSMEConflict
7: return true

```

**Algorithm 4.** Check if the definition of a new SB constraint is allowed.

*Name:* *isSBConstraintAllowed*

*Input:*  $task_1, task_2 \in T_T$

```

1: if  $task_1 == task_2$  then return selfConstraintConflict
2: if  $task_1 \in dme(task_2)$  then return directDMEConflict
3: if  $task_1 \in sme(task_2)$  then return directSMEConflict
4: if  $\exists task_x \in sme(task_1) \mid task_x \in allSubjectBindings(task_2)$ 
5:   then return transitiveSMEConflict
6: if  $\exists task_x \in dme(task_1) \mid task_x \in allSubjectBindings(task_2)$ 
7:   then return transitiveDMEConflict
8: if  $\exists task_x \in sme(task_2) \mid task_x \in allSubjectBindings(task_1)$ 
9:   then return transitiveSMEConflict
10: if  $\exists task_x \in dme(task_2) \mid task_x \in allSubjectBindings(task_1)$ 
11:   then return transitiveDMEConflict
12: return true

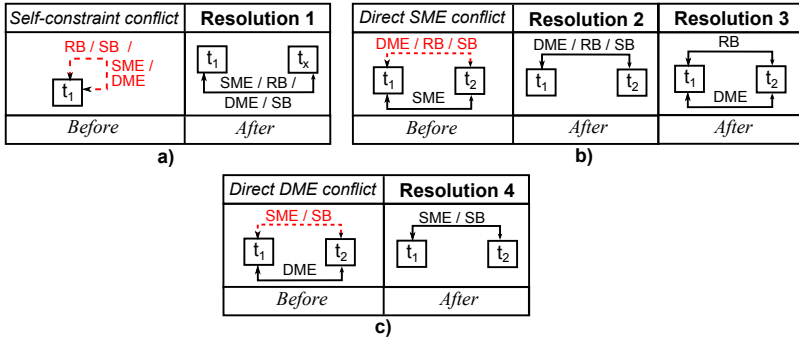
```

### 3.2 Resolving Constraint Definition Conflicts

**Self-constraint Conflict:** A *self-constraint conflict* occurs if we try to define tasks as mutual exclusive or bound to themselves (see Figure 1a and Algorithms 1-4). However, because mutual exclusion as well as binding constraints must be defined on two different task types, such a “self-exclusion” or “self-binding” would violate the consistency requirements defined in Def. 2.1 and Def 2.3.

**Resolution to Self-constraint Conflicts:** In order to prevent inconsistencies resulting from a *self-constraint conflict*, mutual exclusion and binding constraints need always be defined on two different task types (see Resolution 1 and Figure 1a).

**Direct SME Conflict:** A *direct SME conflict* occurs if one tries to define a new DME, RB, or SB constraint on two task types which are already defined as being



**Fig. 1.** Resolving self-constraint (a), SME (b), or DME (c) conflicts

statically mutual exclusive (see Figure 1b). However, as defined in Def. 2.5, two tasks can either be statically or dynamically mutual exclusive (see also 14.15). Furthermore, if two tasks are defined as statically mutual exclusive, it is not possible to define a binding constraint between the same tasks (see Def. 2.6).

**Resolutions to Direct SME Conflicts:** Figure 1b shows two resolutions to prevent *direct SME conflicts*. In particular, this type of conflict can be avoided by removing the conflicting SME constraint before defining the new DME or binding constraint (see Resolution 2). If a direct SME conflict occurs when defining a RB constraint, it can also be resolved by changing the SME into a DME constraint (see Resolution 3), because DME constraints do not conflict with RB constraints (see 14.15).

**Direct DME Conflict:** A *direct DME conflict* occurs if one tries to define a new SME or SB constraint on two task types which are already defined as being dynamically mutual exclusive (see Figure 1c). However, as defined in Def. 2.5, two tasks can either be statically or dynamically mutual exclusive. Moreover, DME and SB constraints conflict (see Def. 2.7, Def. 3.2, and Def. 3.4).

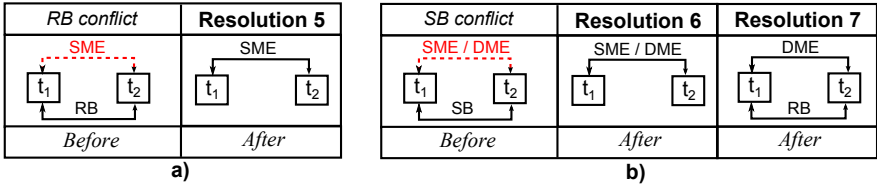
**Resolution to Direct DME Conflicts:** A *direct DME conflict* can be prevented by removing the conflicting DME constraint before defining the new SME or SB constraint (see Resolution 4 and Figure 1c).

**RB Conflict:** A *RB conflict* arises if one tries to define a new SME constraint on two role-bound task types (see Figure 2a). However, because one cannot define a SME constraint and a RB constraint on the same task types (see Def. 2.6), such a configuration would result in a RB conflict.

**Resolution to RB Conflicts:** A *RB conflict* can be prevented by removing the conflicting RB constraint before defining the new SME constraint (see Resolution 5 and Figure 2a).

**SB Conflict:** A *SB conflict* arises if one tries to define a SME or a DME constraint between two subject-bound tasks (see Figure 2b). However, because we cannot define a mutual exclusion constraint and a SB constraint on the same task types (see Def. 2.6 and Def. 2.7), such a configuration would result in a SB conflict.





**Fig. 2.** Resolving RB conflicts (a) or SB conflicts (b)

**Resolutions to SB Conflicts:** A *SB conflict* can be prevented by removing the conflicting SB constraint before defining the new mutual exclusion constraint (see Resolution 6 and Figure 2b). If a SB conflict occurs when defining a DME constraint, it can also be avoided by changing the conflicting SB constraint into a RB constraint (see Resolution 7), because DME and RB do not conflict (see [14,15]).

### Resolution Strategies for Constraint Definition Conflicts

The following resolution strategies define the conflict resolutions described above with respect to the formal definitions of process-related RBAC models (see Section 2 and [14,15]).

**Resolution 1.** *Select two different tasks*

*Input:*  $task_i \in T_T$

- 1: *select*  $task_x \in T \mid task_i \neq task_x \wedge task_x \notin sme(task_i) \wedge task_x \notin dme(task_i) \wedge$
- 2:  $task_x \notin allRoleBindings(task_i) \wedge task_x \notin allSubjectBindings(task_i)$

**Resolution 2.** *Remove SME constraint*

*Input:*  $task_1, task_2 \in T_T$

- 1: *remove*  $task_1$  *from*  $sme(task_2)$  *so that*  $task_1 \notin sme(task_2)$

**Resolution 3.** *Change SME constraint into DME constraint*

*Input:*  $task_1, task_2 \in T_T$

- 1: *remove*  $task_1$  *from*  $sme(task_2)$  *so that*  $task_1 \notin sme(task_2)$
- 2: *and add*  $task_1$  *to*  $dme(task_2)$  *so that*  $task_1 \in dme(task_2)$

**Resolution 4.** *Remove DME constraint*

*Input:*  $task_1, task_2 \in T_T$

- 1: *remove*  $task_1$  *from*  $dme(task_2)$  *so that*  $task_1 \notin dme(task_2)$

**Resolution 5.** *Remove RB constraint*

*Input:*  $task_1, task_2 \in T_T$

- 1: *remove*  $task_1$  *from*  $rb(task_2)$  *so that*  $task_1 \notin rb(task_2)$

**Resolution 6.** Remove SB constraint

Input:  $task_1, task_2 \in T_T$

1: remove  $task_1$  from  $sb(task_2)$  so that  $task_1 \notin sb(task_2)$

**Resolution 7.** Change SB constraint into RB constraint

Input:  $task_1, task_2 \in T_T$

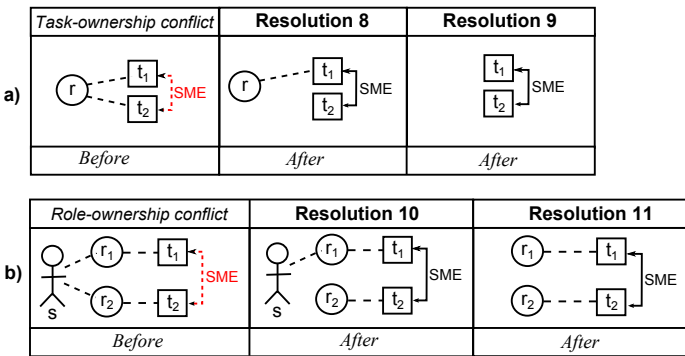
1: remove  $task_1$  from  $sb(task_2)$  so that  $task_1 \notin sb(task_2)$

2: and add  $task_1$  to  $rb(task_2)$  so that  $task_1 \in rb(task_2)$

**3.3 Resolving Ownership Conflicts**

**Task-Ownership Conflict:** A *task-ownership conflict* occurs if one tries to define a SME constraint between two task types that are already assigned to the same role (see Figure 3a). Because two SME tasks must never be assigned to the same role (neither directly nor transitively) such a configuration would result in a task-ownership conflict (see Def. 2(8)).

**Resolutions to Task-Ownership Conflicts:** Figure 3a shows two resolutions to prevent *task-ownership conflicts*. A task-ownership conflict can be avoided by revoking one of the tasks from the corresponding role before defining the new SME constraint (see Resolution 8), or by deleting the conflicting role before defining the new SME constraint (see Resolution 9). Note that Resolution 9 will rarely be applicable in real-world scenarios and is thus only presented for the sake of completeness.



**Fig. 3.** Resolving task-ownership (a) and role-ownership (b) conflicts

**Role-Ownership Conflict:** A *role-ownership conflict* occurs if one tries to define a SME constraint on two task types which are (via the subject’s roles) already assigned to the same subject (see Figure 3b). Because two SME tasks must never be assigned to the same subject (see Def. 2(9)) such a configuration would result in a role-ownership conflict.

**Resolutions to Role-Ownership Conflicts:** A *role-ownership conflict* as shown in Figure 3b can be prevented by revoking one of the conflicting

task-to-role assignments before defining the new SME constraint (see Resolution 8), or by revoking one of the corresponding roles from the subject before defining the new SME constraint (see Resolution 10). Alternatively, it can be avoided by removing role  $r_1$  or  $r_2$  (see Resolution 9) or by removing the subject which owns the conflicting roles (see Resolution 11). Again, Resolutions 9 and 11 will rarely be applicable in real-world scenarios and are only presented for the sake of completeness.

### Resolution Strategies for Ownership Conflicts

The following resolution strategies define the conflict resolutions described above with respect to the formal definitions of process-related RBAC models (see Section 2 and [14,15]).

#### Resolution 8. Remove task-to-role assignment

Input:  $role \in R, task \in T_T$

1: remove role from  $town^{-1}(task)$  so that  $role \notin town^{-1}(task)$

#### Resolution 9. Remove role

Input:  $role \in R$

1: remove role from  $R$  so that  $role \notin R$

#### Resolution 10. Remove role-to-subject assignment

Input:  $subject \in S, role \in R$

1: remove role from  $rown(subject)$  so that  $role \notin rown(subject)$

#### Resolution 11. Remove subject

Input:  $subject \in S$

1: remove subject from  $S$  so that  $subject \notin S$

#### Resolution 12. Remove task

Input:  $task \in T_T$

1: remove task from  $T_T$  so that  $task \notin T_T$

### 3.4 Resolving Transitive Constraint Conflicts

Transitive SME or DME conflicts arise because of the transitivity of binding constraints (see Def. 3.3, Def. 3.4 and [14,15]). Therefore, a conflict may arise when defining a RB or SB constraint on two tasks  $t_1$  and  $t_2$  because of pre-existing mutual exclusion constraints between on one of the tasks  $t_1$  or  $t_2$  and some third task  $t_3$ .

**Transitive SME Conflict:** Figure 4a shows a *transitive SME conflict* that occurs if one tries to define a new role- or subject-binding constraint between two tasks ( $t_1$  and  $t_2$  in Figure 4a) that would result in a transitive binding of a third task ( $t_x$  in Figure 4a) which is already defined as statically mutual exclusive

to one of the other tasks (see SME constraint between  $t_1$  and  $t_x$  in Figure 4a). However, because binding constraints define that two task instances must be executed by the same subject/role (see Def. 3.3 and Def. 3.4), while SME tasks must *not* be executed by the same subject (see Def. 3.1) such a configuration would result in a transitive SME conflict between  $t_1$  and  $t_x$  (see also Def. 2.6).

**Resolutions to Transitive SME Conflicts:** Figure 4a shows conflict resolutions for *transitive SME conflicts*. Such a conflict can be avoided by removing the SME constraint before defining the new binding constraint (see Resolution 2). If the conflict arises when defining a RB constraint, it can also be prevented by changing the SME into a DME constraint before defining the new RB constraint (see Resolution 3). Moreover, the conflict can be resolved by removing the pre-existing binding constraint between  $t_2$  and  $t_x$  before defining the new binding constraint on  $t_1$  and  $t_2$  (see Resolution 5 for removing RB constraints and Resolution 6 for removing SB constraints). Alternatively, a transitive SME conflict can be avoided by removing the task that causes the transitive SME conflict (see Resolution 12). However, Resolution 12 will rarely be applicable in practice.

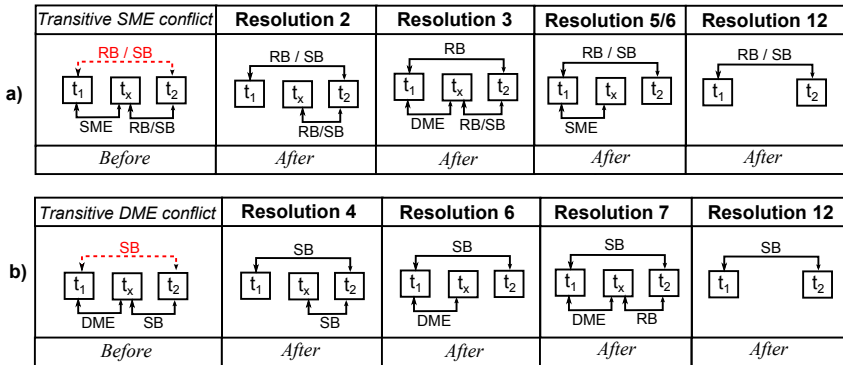


Fig. 4. Resolving transitive SME (a) and DME (b) conflicts

**Transitive DME Conflict:** A *transitive DME conflict* arises because of the transitivity of SB constraints. Figure 4b shows a transitive DME conflict that occurs if one tries to define a new subject-binding between two tasks ( $t_1$  and  $t_2$  in Figure 4b) that would result in a transitive subject-binding of a third task ( $t_x$  in Figure 4b) which is already defined as dynamically mutual exclusive to one of the other tasks (see DME constraint between  $t_1$  and  $t_x$  in Figure 4b). However, SB constraints define that two task instances must be executed by the same subject (see Def. 3.4), while DME constraints define that the corresponding task instance must *not* be executed by the same subject (see Def. 3.2). Therefore, such a configuration would result in a transitive DME conflict between  $t_1$  and  $t_x$  (see also Def. 2.7).

**Resolutions to Transitive DME Conflicts:** Figure 4b shows resolutions for *transitive DME conflicts*. Such a conflict can be prevented by removing the DME constraint before defining the new SB constraint (see Resolution 4), or by removing the pre-existing SB constraint between  $t_2$  and  $t_x$  before defining the new SB constraint (see Resolution 6). It can also be avoided by changing the existing SB constraint into a RB constraint before defining the new SB constraint (see Resolution 7), or by removing the conflicting task  $t_x$  (see Resolution 12).

## 4 Detecting and Resolving Assignment Conflicts

Assignment conflicts arise at design-time when defining new assignment relations between roles, subjects, and tasks. The algorithms defined below check the design-time consistency of a process-related RBAC model when defining a task-to-role, role-to-role, or role-to-subject assignment relation. If an assignment conflict is detected, the algorithms return the name of the respective conflict (see also 14).

### 4.1 Algorithms for Detecting Assignment Conflicts

**Algorithm 5.** Check if it is allowed to assign a particular task type to a particular role (task-to-role assignment).

*Name:*  $isT2RAssignmentAllowed$

*Input:*  $task_x \in T_T, role_y \in R$

- 1: if  $\exists task_y \in town(role_y) \mid task_y \in sme(task_x)$  then return  $taskAssignmentConflict$
- 2: if  $\exists role_z \in allSeniorRoles(role_y) \mid task_z \in town(role_z) \wedge$   
3:  $task_z \in sme(task_x)$  then return  $taskAssignmentConflict$
- 4: if  $\exists s \in S \mid role_y \in rown(s) \wedge role_z \in rown(s) \wedge$   
5:  $task_z \in town(role_z) \wedge task_z \in sme(task_x)$  then return  $roleAssignmentConflict$
- 6: return  $true$

**Algorithm 6.** Check if it is allowed to define a (new) junior-role relation between two roles (role-to-role assignment).

*Name:*  $isR2RAssignmentAllowed$

*Input:*  $junior, senior \in R$

- 1: if  $junior == senior$  then return  $selfInheritanceConflict$
- 2: if  $senior \in rh^*(junior)$  then return  $cyclicInheritanceConflict$
- 3: if  $\exists task_j \in town(junior) \wedge task_s \in town(senior) \mid$   
4:  $task_j \in sme(task_s)$  then return  $taskAssignmentConflict$
- 5: if  $\exists role_x \in allSeniorRoles(senior) \mid task_x \in town(role_x) \wedge$   
6:  $task_j \in town(junior) \wedge task_x \in sme(task_j)$   
7: then return  $taskAssignmentConflict$
- 8: if  $\exists s \in S \mid senior \in rown(s) \wedge role_x \in rown(s) \wedge$

```

9:   taskx ∈ town(rolex) ∧ taskj ∈ town(junior) ∧ taskx ∈ sme(taskj)
10:  then return roleAssignmentConflict
11: return true

```

**Algorithm 7.** Check if it is allowed to assign a particular role to a particular subject.

**Name:** *isR2SAssignmentAllowed*

**Input:**  $role_x \in R, subject \in S$

```

1: if ∃ roley ∈ rown(subject) | tasky ∈ town(roley) ∧
2:   taskx ∈ town(rolex) ∧ tasky ∈ sme(taskx) then return
   roleAssignmentConflict
3: return true

```

## 4.2 Resolving Assignment Conflicts

**Self Inheritance Conflict:** A *self inheritance conflict* may arise when defining a new inheritance relation between roles. In particular, a role cannot be its own junior-role (see Figure 5a and [14,15]).

**Resolution to Self Inheritance Conflicts:** This conflict can be resolved by changing one of the selected roles so that the inheritance relation is defined between two different roles (see Figure 5a and Resolution 13).

**Cyclic Inheritance Conflict:** A *cyclic inheritance conflict* results from the definition of a new inheritance relation in a role-hierarchy (also called role-to-role assignment). In particular, a role-hierarchy must not include a cycle because all roles within such a cyclic inheritance relation would own the same permissions which would again render the respective part of the role-hierarchy redundant (see Figure 5b and [14,15]).

**Resolutions to Cyclic Inheritance Conflicts:** This conflict can be resolved by defining a new inheritance relation between roles which are not already part of the same role-hierarchy (see Resolution 13). In Figure 5b, Resolution 13 is applied by defining a new inheritance relation between  $r_x$  and  $r_y$  while keeping the existing inheritance relation between  $r_y$  and  $r_z$ . Moreover, the existing inheritance relation between  $r_y$  and  $r_z$  can be removed before defining the inverse inheritance relation with  $r_z$  as junior role of  $r_y$  (see Resolution 14).

**Task-Assignment Conflict:** A *task-assignment conflict* may occur if the definition of a new *tra* or junior-role relation would result in the assignment of two SME tasks to the same role (see Def. 28). Figure 6a depicts an example where a role  $r_y$  owns a task  $t_y$  which is defined as SME to another task  $t_x$ . Thus, assigning  $t_x$  to  $r_y$  would result in a task-assignment conflict.

**Resolutions to Task-Assignment Conflicts:** To avoid the *task-assignment conflict* in Figure 6a, the conflicting SME constraint between the two task types can be removed or changed into a DME constraint (see Resolutions 2 and 3). Alternatively, task  $t_y$  can be revoked from  $r_y$ , or the conflicting task  $t_y$  can be deleted (see Resolutions 8 and 12).

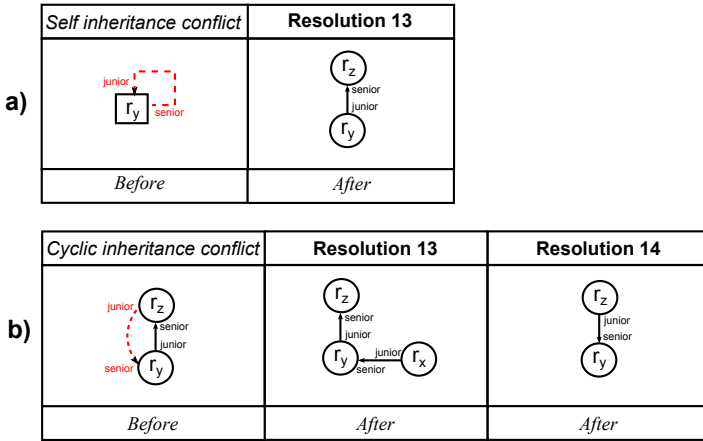


Fig. 5. Resolving self-inheritance (a) and cyclic inheritance (b) conflicts

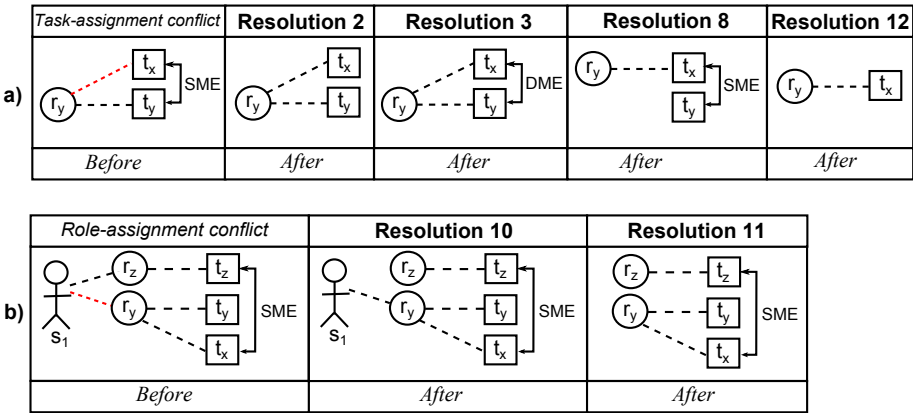


Fig. 6. Resolving task- (a) and role-assignment (b) conflicts

**Role-Assignment Conflict:** A *role-assignment conflict* arises if a new assignment relation would authorize a subject to perform two SME tasks. Figure 6b shows an example, where an assignment of role  $r_y$  to subject  $s_1$  would result in a role-assignment conflict because subject  $s_1$  would then be authorized to perform the two SME tasks  $t_z$  and  $t_x$ . Thus, such an assignment would violate the consistency requirement specified in Def. 2.9. Similarly, when defining a new junior-role or *tra* relation, we need to check for role-assignment conflicts.

**Resolutions to Role-Assignment Conflicts:** To avoid a *role-assignment conflict*, the same resolutions as for task-assignment conflicts can be applied (see Resolutions 2, 3, 8, and 12). In addition, Resolution 10 can be applied by removing the conflicting assignment between  $r_z$  and  $s_1$  (see Figure 6b). Moreover,

the conflict can (theoretically) be resolved by removing the conflicting subject  $s_1$  which is assigned to the two SME tasks (see Resolution [11](#)).

**Resolution Strategies for Assignment Conflicts**

The following resolution strategies define the conflict resolutions described above with respect to the formal definitions of process-related RBAC models (see Section [2](#) and [14,15](#)).

**Resolution 13.** *Select two different roles*

Input:  $role_i \in R$

1: select  $role_x \in R \mid role_i \neq role_x \wedge role_x \notin allSeniorRoles(role_i)$

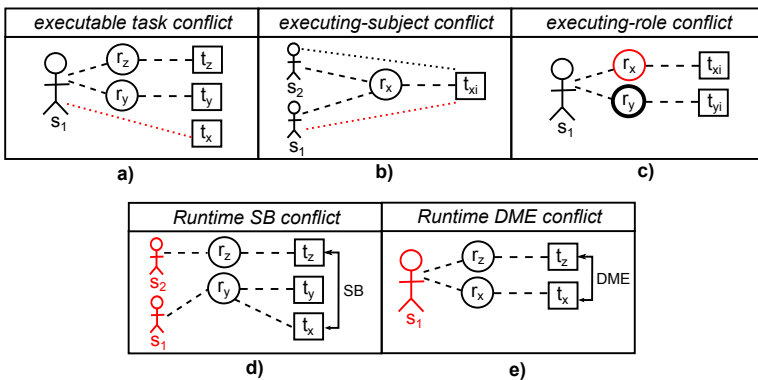
**Resolution 14.** *Remove junior-role relation*

Input:  $role_y, role_z \in R$

1: remove  $role_y$  from  $rh^*(role_z)$  so that  $role_y \notin rh^*(role_z)$

**5 Detecting and Resolving Runtime Conflicts**

Conflicts may also occur when executing process instances. Thus, runtime conflicts arise when actually enforcing constraints. In particular, mutual-exclusion and binding constraints directly impact the allocation of tasks to subjects. Below we discuss five potential conflicts when allocating a particular task instance to a certain subject. These conflicts are illustrated in Figures [7a-e](#), where conflicts arise when we try to allocate subject  $s_1$  to an instance of the a task type  $t_x$  (in Figure [7](#) instances of  $t_x$  are labeled as  $t_{xi}$ ).



**Fig. 7.** Runtime conflicts

Algorithm [8](#) checks the runtime consistency of a process-related RBAC model when allocating a task instance to a particular subject. If one of the runtime conflicts shown in Figures [7a-e](#) is detected, the algorithm returns the name of the respective conflict.



**Algorithm 8.** Check if a particular task instance executed during a specific process instance can be allocated to a particular subject.

**Name:** *isAllocationAllowed*

**Input:**  $subject \in S, task_{type} \in T_T, process_{type} \in P_T,$

$process_{instance} \in pi(process_{type}), task_{instance} \in ti(task_{type}, process_{instance})$

```

1: if  $task_{type} \notin executableTasks(subject)$  then return executableTaskConflict
2: if  $es(task_{instance}) \neq \emptyset$  then return executingSubjectConflict
3: if  $er(task_{instance}) \neq \emptyset \wedge er(task_{instance}) \neq ar(subject)$ 
4:   then return executingRoleConflict
5: if  $\exists type_x \in allSubjectBindings(task_{type}) \mid$ 
6:    $type_x \notin executableTasks(subject)$  then return runtimeSBConflict
7: if  $\exists instance_y \in ti(type_y, process_{instance}) \mid$ 
8:    $type_y \in dme(task_{type}) \wedge es(instance_y) == subject$ 
9:   then return runtimeDMEConflict
10: return true

```

**Executable Task Conflict:** An *executable task conflict* arises if the selected subject is not allowed to execute the task type the corresponding task instance was instantiated from. If subject  $s_1$  is not allowed to execute instances of task  $t_x$  (see Figure 7a), the respective task instance must not be allocated to  $s_1$ .

**Resolutions to Executable Task Conflicts:** An *executable task conflict* can be resolved by allocating an executing subject that actually owns the permission to perform the respective task (see Resolution 15). Alternatively, one may change the *rsa* or the *tra* relations so that  $s_1$  is allowed to execute  $t_x$ .

**Executing-Subject Conflict:** An *executing-subject conflict* arises if the allocation is not possible, because the respective task instance already has been allocated to another subject. For example, in Figure 7b the task instance  $t_{xi}$  already has an executing subject  $s_2$  and thus cannot be allocated to  $s_1$ .

**Resolution to Executing-Subject Conflicts:** An *executing-subject conflict* can only be resolved by first deallocating the executing-subject before the respective task instance can be reallocated to another subject that is allowed to perform the respective task (see Resolution 16 and Algorithm 8).

**Executing-Role Conflict:** An *executing role conflict* visualized in Figure 7c occurs if a task instance already has an executing role, but this executing role is not the active role of the designated executing-subject.

**Resolution to Executing-Role Conflicts:** An *executing-role conflict* can be resolved by changing the active role of the subject to the executing-role of the respective task instance (see Resolution 17).

**Runtime SB Conflict:** Figure 7d shows an example of a *runtime SB conflict* that occurs when we try to allocate  $s_1$  to an instance of  $t_x$ . In particular, we need to check if some task type  $t_z$  exists that has a SB relation to  $t_x$  but cannot be executed by  $s_1$ . Such an allocation violates the consistency requirement specified in Def. 34, because subject-bound tasks must have the same executing

subject. Thus, a subject can only be allocated if it owns the right to perform the corresponding task type as well as all subject-bound tasks.

**Resolutions to Runtime SB Conflicts:** This conflict can be resolved by removing the SB constraint (see Resolution [6](#)). Moreover, the *tra* relation for the subject-bound task or the *rsa* relation for one of the roles owning this task can be changed so that the designated executing-subject is allowed to perform the tasks that are connected via a (transitive) SB constraint. Furthermore, one of the subject-bound tasks can be removed in order to resolve the SB conflict (see Resolution [12](#)), or the executing-subject can be changed (see Resolution [15](#)).

**Runtime DME Conflict:** In the example from Figure [7e](#), a *runtime DME conflict* would occur if we try to allocate  $s_1$  to an instance of  $t_z$  and to an instance of  $t_x$  in the same process instance. This is because a DME constraint defines that in the same process instance the instances of two DME task types must not be performed by the same subject (see Def. [32](#)).

**Resolutions to Runtime DME Conflicts:** A *runtime DME conflict* is prevented by either removing the DME constraint, by removing one of the DME tasks, or by changing the executing-subject (see Resolutions [4](#), [12](#), [16](#) and [15](#)).

### Resolution Strategies for Runtime Conflicts

The following resolution strategies define the conflict resolutions presented above with respect to the formal definitions of process-related RBAC models.

**Resolution 15.** *Select a subject that is allowed to perform the respective task*

*Input:*  $task \in T_T, role \in R$

1: *select*  $subject \in S \mid role \in rown(subject) \wedge task \in town(role)$

**Resolution 16.** *Deallocate a task instance*

*Input:*  $task_i \in T_I$

1: *set*  $es(task_i) = \emptyset$  *and*  $er(task_i) = \emptyset$

**Resolution 17.** *Change the executing-subject's active role to the executing-role of the respective task*

*Input:*  $task_i \in T_I, subject \in S \mid es(task_i) == subject$

1: *if*  $er(task_i) \neq ar(subject)$  *then set*  $ar(subject) = er(task_i)$

## 6 Related Work

Sloman and Moffett [9,10,13](#) were among the first to analyze and categorize conflicts between different types of policies. They also presented informal strategies for resolving these conflicts. In [11](#), Ahn and Sandhu presented the RCL 2000 language for the specification of role-based authorization constraints. They also show how SOD constraints can be expressed in RCL 2000 and discuss different types of conflicts that may result from constraints specified via RCL 2000. Bertino et al. [3](#)

present a language to express SOD constraints as clauses in logic programs. Moreover, they present corresponding algorithms that check the consistency of such constraints. Thereby they ensure that all tasks within a workflow are performed by predefined users/roles only. In [4], Botha and Eloff present an approach called conflicting entities administration paradigm. In particular, they discuss possible conflicts of static and dynamic SOD constraints in a workflow environment and share a number of lessons learned from the implementation of a prototype system. Schaad [12] discusses the detection of conflicts between SOD constraints in a role-based delegation model. Schaad follows a rule-based, declarative approach by using the Prolog language as an executable specification language.

Wang et al. [18] define algorithms for the detection of conflicts between access control policies. Similarly, in [2], an approach for the formalization of policy rules is proposed and algorithms for policy conflict resolutions are derived. Yet, both approaches do not consider conflicts resulting from SOD or BOD constraints. Tan et al. [16] define a model for constrained workflow systems, including SOD and BOD constraints. They discuss different issues concerning the consistency of such constraints and provide a set of formal consistency rules that guarantee the definition of a sound constrained workflow specification. In [6] Ferraiolo et al. present RBAC/Web, a model and implementation for RBAC in Web servers. They also discuss the inheritance and resulting consistency issues of SOD constraints in role-hierarchies. Jaeger et al. [8] present a formal model for constraint conflicts and define properties for resolving these conflicts. They applied metrics for resolving Biba integrity violations in an SELinux example policy.

## 7 Conclusion

In this paper, we discussed resolution strategies for conflicts of process-related mutual-exclusion and binding constraints. Because of the countless configurations that could cause conflicts, we chose to discuss frequently occurring conflict types which group similar conflicts. In the same way, we described corresponding types of resolution strategies. If a certain resolution strategy is actually applicable to a specific real-world conflict can, however, only be decided by the corresponding process modeler or security engineer.

Note that in our approach, conflicts are detected and resolved before causing an inconsistent RBAC configuration. In other words, the formal consistency requirements for static and dynamic correctness of our process-related RBAC models must hold at any time and therefore prevent the definition of inconsistent RBAC models. The application of the algorithms and resolution strategies described in this paper can help process modelers and security engineers to identify resolution options for design-time and runtime conflicts in process-related RBAC models.

## References

1. Ahn, G., Sandhu, R.: Role-based Authorization Constraints Specification. ACM Transactions on Information and System Security (TISSEC) 3(4) (November 2000)

2. Baliosian, J., Serrat, J.: Finite State Transducers for Policy Evaluation and Conflict Resolution. In: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (June 2004)
3. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC)* 2(1) (1999)
4. Botha, R.A., Eloff, J.H.: Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal* 40(3) (2001)
5. Casati, F., Castano, S., Fugini, M.: Managing Workflow Authorization Constraints through Active Database Technology. *Information Systems Frontiers* 3(3) (2001)
6. Ferraiolo, D., Barkley, J., Kuhn, D.: A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet. *ACM Transactions on Information and System Security (TISSEC)* 2(1) (February 1999)
7. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: *Role-Based Access Control*, 2nd edn. Artech House (2007)
8. Jaeger, T., Sailer, R., Zhang, X.: Resolving constraint conflicts. In: Proc. of the Ninth ACM Symposium on Access Control Models and Technologies, SACMAT (2004)
9. Moffett, J.D., Sloman, M.S.: Policy Hierarchies for Distributed Systems Management. *IEEE Journal on Selected Areas in Communications* 11(9) (1993)
10. Moffett, J.D., Sloman, M.S.: Policy Conflict Analysis in Distributed System Management. *Journal of Organizational Computing* 4(1) (1994)
11. Feinstein, H., Sandhu, R., Coyne, E., Youman, C.: Role-based access control models. *IEEE Computer* 29(2) (1996)
12. Schaad, A.: Detecting Conflicts in a Role-Based Delegation Model. In: Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC (December 2001)
13. Sloman, M.S.: Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management* 2(4) (1994)
14. Strembeck, M., Mendling, J.: Generic Algorithms for Consistency Checking of Mutual-Exclusion and Binding Constraints in a Business Process Context. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) *OTM 2010*. LNCS, vol. 6426, pp. 204–221. Springer, Heidelberg (2010)
15. Strembeck, M., Mendling, J.: Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology* 53(5) (2011)
16. Tan, K., Crampton, J., Gunter, C.A.: The Consistency of Task-Based Authorization Constraints in Workflow Systems. In: Proceedings of the 17th IEEE workshop on Computer Security Foundations (June 2004)
17. Wainer, J., Barthelmeß, P., Kumar, A.: W-RBAC - A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems (IJCIS)* 12(4) (2003)
18. Wang, H., Sun, L., Varadharajan, V.: Purpose-based access control policies and conflicting analysis. In: Rannenber, K., Varadharajan, V., Weber, C. (eds.) *SEC 2010*. IFIP AICT, vol. 330, pp. 217–228. Springer, Heidelberg (2010)
19. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proc. of the Eleventh ACM Symposium on Access Control Models and Technologies, SACMAT (June 2006)

# Implementation, Optimization and Performance Tests of Privacy Preserving Mechanisms in Homogeneous Collaborative Association Rules Mining

Marcin Gorawski<sup>1,2</sup> and Zacheusz Siedlecki<sup>1</sup>

<sup>1</sup> Institute of Computer Science,  
Silesian University of Technology,  
Akademicka 16, 44-100 Gliwice, Poland  
{Marcin.Gorawski,Zacheusz.Siedlecki}@polsl.pl

<sup>2</sup> Institute of Computer Science,  
Wroclaw University of Technology,  
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland  
Marcin.Gorawski@pwr.wroc.pl

**Abstract.** This article focuses on optimization and performance tests of association rules multiparty mining algorithms. We present how to improve Secure Set Union performance by using a Common Decrypting Key for a commutative encryption. We have also improved Secure Set Union making it fully secure in a semi-honest model. As an example of the above mentioned mechanisms application, the article presents new algorithms of mining association rules on horizontally partitioned data with preserving data privacy - CDKSU (Secure Union with Common Decrypting Key) and its fully secure version - CDKRSU (Secure Union with Common Decrypting Key and secure duplicate Removing). Those algorithms are compared with KCS scheme since they are all based on FDM. As far as the performance optimization is concerned, the application of Elliptic Curve Cryptography vs Exponential Cryptography is presented as well. The real, working system implementing given algorithms is subjected to performance tests which results are presented and analyzed.

**Keywords:** cryptography, association rule, multiparty data mining, commutative encryption, elliptic curve.

## 1 Introduction

In the modern business, success depends on collaboration and partnership. Collaborative data mining becomes highly important, because of the mutual benefit the partners gain. Collaboration occurs among companies that have conflict of interests or compete against each other on the market. During collaboration multiple parties want to conduct data mining on a data set that consists of all the parties' private data, but neither party wants the other parties or any third

party to learn much about their private data. It makes data privacy mechanisms extremely important. The performance expense of this mechanisms, however, can be prohibitive when applying to large databases. This article focuses on optimization of cryptographic privacy preserving mechanisms in association rules mining algorithms on horizontally partitioned data.

## 1.1 Related Works

There is a great interest in research on privacy-preserving data mining [28,3,31,12]. There have been proposed privacy preserving algorithms for different data mining applications, including clustering [38,24,21,23], association rules mining on randomized data [35,13], association rules mining across multiple databases [37,25], Bayes classification [40,39,29,42], decision trees on randomized data [3], frequent pattern mining [16] and collaborative filtering [9]. Additionally, several privacy preserving solutions have been proposed for simple primitives that are very useful for designing privacy preserving data mining algorithms. These include computing scalar products [8,6,37,43,14,17] and finding common elements [14,3]. The Secure Multiparty Computation paradigm provides cryptographic solutions for protecting privacy in any distributed computation [19,44]. Works most related to ours are [11,10,25]. We have improved Secure Sum scheme [11] and created a new algorithm similar to KCS (Kantarcioglu and Clifton Scheme) [25] (referenced as HPSU also [22,20]) and based on FDM [10]. We have not come across any study dealing with the Elliptic Curve Cryptography (ECC) used in multiparty data mining.

## 1.2 Our Contribution

We have demonstrated that it is possible to increase performance focusing on optimization of cryptographic mechanisms used in the given algorithms. Of all the elements of protection, encrypting has the highest computation expense, but contrary to distorting input data, it does not disturb the results. Previous research in mining association rules on horizontally partitioned data deal with commutative [25,15,22] and homomorphic encryption [45]. They all describe the use of exponential ciphers. It is recommended to substitute exponential ciphers with methods based on Elliptic Curve Cryptography [30,11]. In this article, the application of Elliptic Curve Cryptography (ECC) versus Exponential Cryptography is presented. We introduce the manner of using a Common Decrypting Key (CDK) for commutative encryption in a secure union to improve performance as well. As an example of the usage of the above mentioned methods, we have presented our own algorithm CDKSU (Secure Union with Common Decrypting Key). We have also improved Secure Set Union making it fully secure in a semi honest model by adding a Secure Duplicate Removing (SDR) protocol. We have proposed CDKRSU (Secure Union with Common Decrypting Key and secure duplicate Removing) which is a fully secure version of CDKSU. Those algorithms are compared with KCS [25], since they are similar and all based on FDM [10]. The system implementing given algorithms is subjected to performance tests. The comparison of the tests results shows the performance benefits that stem from the use of CDK and ECC.

## 2 Homogeneous Collaborative Association Rules Mining with Data Privacy Preserving

The goal of association rule mining is to discover meaningful associations among the attributes of a large quantity of data. Briefly, an association rule is an expression  $X \rightarrow Y$  where  $X$  and  $Y$  are sets of items. Given a database  $D$ ,  $X \rightarrow Y$  means that whenever a record contains  $X$  then it also contains  $Y$  with certain confidence. The confidence is the percentage of records containing both  $X$  and  $Y$  with regard to the overall number of records containing  $X$ . The support of  $X$  is the fraction of records supporting an item  $X$  with respect to the database  $D$ .

The majority of association rules mining algorithms is based on the algorithm *Apriori* whose author is R. Agrawal [5]. Multiple algorithms for distributed association rules mining have been proposed in [46,4]. On the basis of these algorithms, multiple association rules mining with preserving data privacy algorithms [25,22,37] have been created. They are essential, if multiple parties want to find association rules in a data set that consists of all the parties' private data, but neither party is willing to disclose its private data to the other parties or any other parties. The introduction to the association rules multiparty mining algorithms with preserving data privacy can be found in [22,47].

There have been two main approaches for privacy preserving in multiparty association rules mining. One is a randomization approach [13,35] in which the privacy of data cannot be always fully preserved when achieving the precision of the results [26,2]. The other is a cryptographic approach mostly using SMC (Secure Multiparty Computation) [44,11].

Most efficient privacy preserving solutions can be often designed for specific distributed computations. We focus on SMC-based association rules mining algorithms on horizontally partitioned data in the cryptographic approach. This task is also known as a heterogeneous collaborative association rule mining [25,22,47]. By using CDK for a commutative cipher, we have improved a Secure Set Union method [11] commonly used in such algorithms. We have designed a new multiparty association rules mining algorithm CDKSU as an example of an algorithm using CDK. We have also proposed CDKRSU algorithm which is a fully secure version of CDKSU using SDR protocol. Bear in mind that it is a fully secure versio in a semi honest model. These two algorithms are based on FDM and use a commutative encryption with CDK.

## 3 Cryptographic Privacy Preserving Mechanisms in a Homogeneous Collaborative Association Rules Mining

### 3.1 Commutative Encryption

**Definition 1.** An encryption algorithm is commutative if given encryption keys  $K_1, \dots, K_n \in K$ , for any  $m$  in domain  $M$ , and for any permutation  $i, j$ , the following two equations hold [11]:

$$E_{K_{i1}} (\dots E_{K_{in}} (m) \dots) = E_{K_{j1}} (\dots E_{K_{jn}} (m) \dots)$$

$\wedge m_1, m_2 \in M$  such that  $m_1 \neq m_2$  and for given  $k, \epsilon < \frac{1}{2^k}$

$$Pr [E_{K_{i1}} (\dots E_{K_{in}} (m_1) \dots) = E_{K_{j1}} (\dots E_{K_{jn}} (m_2) \dots)] < \epsilon$$

### 3.2 Commutative Encryption with Common Decryption Key

We have come up with a new method of defining a Common Decrypting Key CDK (Common Decrypting Key) for a commutative encryption.

**Definition 2.** For a commutative encryption  $E$  you can determine CDK, if for the given decrypting keys  $K_1, \dots, K_n \in K$  it is possible to determine such a common key  $K_d$  that, when using it to decrypt  $D$  for every message  $m$  in domain  $M$  and for every permutation  $i, j$ , the following equation is hold:

$$D_{K_d} (E_{K_{j1}} (\dots E_{K_{jn}} (m) \dots)) = m$$

Key  $K_d$  is called the Common Decrypting Key.

### 3.3 Pohlig-Hellman Exponential Commutative Cipher

We used Pohlig-Hellman encryption scheme with modulus shared among parties [33]. For given finite field  $GF(p)$  where  $p$  is prime,  $\otimes$  operation in it, field element  $m$  and integers  $e, d$  such that  $ed = 1 \pmod{p-1}$ , we know that  $m^{ed} = m^{1+k(p-1)} = m^1 \otimes 1 = m$ . Based on exponentiation in such finite fields there are two operations defined:

- encryption  $m^e = n$ ,
- decryption  $n^d = m$ .

Element  $m$  is the plaintext message and pair  $\{e, d\}$  is the secret key. It satisfies equations given in [31] because for every  $a$  and  $b$  the equation  $(m^a)^b = (m^b)^a$  is held. Thus Pohlig-Hellman cipher is commutative.

Let's assume that we have a pair of keys  $\{e, d\}$  satisfying the conditions from [32]:  $K_1 = \{e_1, d_1\}$  and  $K_2 = \{e_2, d_2\}$ . For each pair like that (in the same field  $GF(p)$ ) the following equation is true:

$$((m^{e_1})^{e_2})^{d_1 d_2} = m$$

This case can include any number of keys. Thus, by multiplying all the decrypting keys we will get CDK.

### 3.4 Elliptic Curves Cryptography

This work presents the use of ECC in association rules multiparty mining algorithms with preserving data privacy. Smooth elliptic curves that were used had



points that had integer coordinates in finite fields  $Z_p$ . When referring to *elliptic curve* further in this work, we will have in mind this type of smooth curve. Introduction to elliptic curves and their usage in cryptography can be found in [7]. In order to use elliptic curves cryptography we have to encode our plaintext data as points on some given elliptic curve  $E$  over a finite field  $Z_p$ . We use a probabilistic algorithm given by Kolbitz at the beginning of the *Elliptic curve cryptosystems* chapter in [27].

**Elliptic Curve Pohlig-Hellman Cipher.** The Elliptic Curve Pohlig-Hellman cipher is described in [41]. It works exactly like the Pohlig-Hellman cipher, except for the fact that the multiplicative group of integers modulo  $p$  is replaced by the additive elliptic curve group.

Let  $p$  be a large prime and let  $E$  be an elliptic curve modulo  $p$  that has order  $N$ . A point on  $E$  is enciphered by adding to itself  $e$  times so the ciphertext point is  $Q = eP$ . The latter is deciphered by multiplying by  $d$ :  $P = dQ$ . The pair  $\{e, d\}$  is the secret key. The  $O$  is point at infinite. The  $e$  and  $d$  must satisfy  $ed = 1 \pmod{N}$ , because  $NP = O$  by Lagrange’s theorem so  $e$  (and  $d$ ) must be chosen relatively prime to  $n$ .

Since elliptic curve group is an abelian group, we can say that  $((P * m) * n) = ((P * n) * m)$ , where  $P$  is the point on curve and  $m, n$  are integers. Such defined encryption is commutative, which means  $C_m(C_n(P)) = C_n(C_m(P))$ . Moreover, since  $((P * m) * n) = P * (n * m)$ , then the defined decryption has a unique quality  $D_m(D_n(P)) = D_{m*n}(P)$ . It allows us to sequentially encrypt the message using many keys, and then decrypt it using a key that is the product of multiplying these keys.

### 3.5 Secure Sum and Product

The secure sum is often given as a simple example of a secure multiparty computation [36]. The secure product works just like the sum except that the addition is replaced by multiplication and subtraction by division. Assuming the presence of three or more parties, the following method securely computes the sum of values from the individual sites.

There is an operation  $\otimes$  and set  $G$ , for which the pair  $(\otimes, G)$  is a group consisting of at least two elements. For a sequence  $x_1, \dots, x_l \in G$  let’s mark the operation  $x_1 \otimes x_2 \otimes x_3, \dots, x_{l-1} \otimes x_l$  as  $\prod_{i=1}^l x_i$ . The algorithm calculates the value  $V = \prod_{i=1}^l x_i$ , where:  $l$  is a number of sites,  $x_i$  is the value of a site  $i$  [11].

First, the chosen site generates a random number  $R$ , uniformly chosen from  $[0..n]$  and to the next site it sends the value of  $R \otimes x_1$ . On the basis of this value, the next site calculates  $R \otimes x_1 \otimes x_2$  and sends it to the next site, etc. The last site sends the given value to the first site, which knows the element  $R'$  (which is the reciprocal of the element  $R$ ). First site calculates  $R' \otimes R \otimes \prod_{i=1}^l x_i = \prod_{i=1}^l x_i$  and gets the expected value  $V$ .

In the secure product the  $\otimes$  is the multiplication. In the secure sum  $\otimes$  is the addition and the calculated result is  $V = \sum_{i=1}^l x_i$ . Additive or multiplicative

groups are often used. Calculations in them are conducted modulo given number (modulus), as described in [11].

### 3.6 Multiparty Calculation of Common Decrypting Key

Using a multiparty secure product calculation method one can determine CDK for Pohlig-Hellman and Elliptic Curve Pohlig-Hellman ciphers. When using this method, if all other sites agree, CDK can be revealed to one of the designated sites, but the secret encrypting keys that belong to these sites won't be revealed.

### 3.7 Calculating the Secure Set Union Using Encryption with CDK

Using a commutative encryption described in the previous chapter, one can securely calculate the union of sets belonging to many parties without revealing the originator of the particular element. Each party that takes part in the process, has its own encrypting key. If every  $l$  site has its own private set  $Z_i$ , then by calculating the set union, we mean calculating  $\bigcup_{i=1}^l Z_i$ .

One special type of site has been designated – *Site D*. It is responsible for decrypting data, initializing the computation of a CDK (let's call it *Master Key*), and it is the only site that knows its value. Because of that, the communication should be conducted in such a way, that *Site D* has no contact with other sites' data encrypted by all the sites, which at the same time do not belong to the result. Computation on fully encrypted data (meaning encrypted by all the sites) will be conducted by the rest of the sites that do not have the access to the *Master Key*.

1. Each party encrypts its own data set and sends it to the next party. When the next party receives data from the previous party, it enciphers it again using its own encrypting key and sends it to another party.
2. This process is repeated until all the data sets are encrypted by all the parties. Then, among the parties which are not *Site D*, the calculation of the union of sets of encrypted values is conducted. This calculation is based on commutative encryption properties.
3. Using multiparty Secure Sum algorithm the *Master Key* is calculated (it can be calculated once for all iterations).
4. The resulting union is sent to *Site D*, which deciphers it and then sends to the other parties.

The use of the CDK reduces the number of expensive decryption operations required to obtain the result. It is worth mentioning, that analogically one can conduct the computation of intersection of the sets  $\bigcap_{i=1}^l z_i$ .

### 3.8 Secure Duplicate Removing

The protocol is used to send data elements from one site to another so that the second site does not receive elements it already has. None of the sites is able

to determine whether there are duplicates between the sites (which have not been sent). The protocol is based on a one-way function. It would be possible to use cryptographic hash functions for that purpose. Such functions additionally compress the data, but also have several properties which increase the computational load, and are unnecessary in this case.

Four sites are involved in the protocol:

- two sites (A and B) whose data elements are to be compared:
  - Site A (source), whose element is to be sent
  - Site B (target), which is to receive the element
- Comparator Site, which compares the element values
- Remover Site, which acts as an intermediary in the item transfer from Site A to Site B and removes any duplicate elements.

The algorithm may be used to transfer an element between two sites as follows:

1. Sites A and B agree between them upon a random data block  $r$ . Site A randomly generates the data block  $r$  and sends it to Site B.
2. Site A assigns an identifier to each of its data elements. These identifiers will identify the messages sent by the Remover Site to Site B.
3. Sites A and B compute concatenation  $k$  of a unique representation of each of their data elements with the data block  $r$ .
4. Sites A and B compute the result  $s$  of a given one-way function (e.g. secure cryptographic hash SHA1) of each  $k$ .
5. Sites A and B agree upon a set of fake hashes  $f$ .
6. Sites A and B send the set of their  $s$ , together with the identifiers assigned as per step 2 above, to the Comparator Site. To hide the number of duplicated elements from the Comparator Site, Sites A and B also send identical fake hashes  $f$  agreed upon previously.
7. If the Comparator Site detects a duplicate, it sends to the Remover Site the identifier of the message (from Site A) which is to be removed.
8. Site A transfers to Site B, through the Remover Site, the data elements, as well as fake elements with identifiers corresponding to the fake duplicates sent to the Comparator Site. Remover Site passes to Site B all elements except duplicates with identifiers delivered by the Comparator Site.

## 4 CDKSU Algorithm

Our CDKSU algorithm (Secure Union with Common Decrypting Key) conducts multiparty association rules mining algorithms on horizontally partitioned data. It preserves data privacy in *semi-honest* model [18]. It assumes that the parties involved will honestly follow the protocol, but can later try to infer additional information from whatever data they receive through the protocol. The algorithm is similar to KCS (HPSU), and differs mainly in the way it safely determines the union of locally supported itemsets without revealing the originator of the particular itemset. Despite that privacy protection in this algorithm is based on the same concept (except for CDK which is described in a separate section).

Let us assume that a transaction database  $DB$  is horizontally partitioned among  $n \geq 3$  sites (namely  $S_1, S_2, \dots, S_n$ ). Each site has a private transaction database  $DB_1, DB_2, \dots, DB_n$  where  $DB_i$  resides at site  $S_i$ . The itemset  $X$  has the local support count of  $X.sup_i$  at site  $S_i$ , if  $X.sup_i$  of the transactions contains  $X$ .

1. Generation of locally large itemsets.

Candidates to be recognised as locally large (frequent) itemsets are generated using *Apriori* property [5], from the intersection of the list of locally large itemsets from a previous operation with the list of globally large itemsets from previous operation, so from these locally large itemsets, which were recognised as globally large. In the first iteration these are all one element itemsets. Parties locally prune candidates and leave out only locally large itemsets. In order to hide the number of locally large itemsets, fake sets are added. The number of fake itemsets is randomly generated from a Uniform distribution in such a way, that the final number of the locally large itemsets does not exceed the number of the candidates. We do not have to concern about fake item set content because when calculating a secure union they are encrypted. Its items have simply got uniform random identifiers and one of them has random identifier from a range dedicated for fake items. Without reducing the security of the algorithm, it allows to reject fake sets after decrypting the element of the union.

2. Secure determination of the union of locally large itemsets.

When using the method described in [3.7] locally large itemsets union is safely designated. In order to do this, commutative cipher with CDK is used. Each site has a secret key, which enciphers elements. Each *itemset* is enciphered by each site. Next, the union of sets of itemsets that belongs to each party is calculated, as described in [3.7]. In order to hide the number of parties that possess the given itemset, the union of sets that belongs to the parties assigned an even ordinal number are calculated by one of the parties, and the parties that are assigned an odd ordinal number are calculated by another party, the results are then summed up. The obtained set of all the itemsets is decrypted by *Site D* in order to get a plaintext result. Sets that are left after rejecting the fake sets are candidates for globally large itemsets. Fake itemsets can be recognised, because they include an element with an identifier assigned to false sets. It allows to reject the false sets as soon as the element from the union are deciphered.

3. Secure determination of globally large itemsets.

In order to check if the set  $X$  is globally large, it is checked whether its exceeding support which equals  $X.sup - s * |DB| = \sum_{i=1}^n (X.sup_i - s * |DB_i|)$  is nonnegative. That means it meets the global support threshold. It is explained in details in [25]. Exceeding the support of every candidate for globally large itemset is calculated by using Secure Sum algorithm described in [3.5]. On its basis, new globally frequent itemsets are designated.

#### 4. Secure determination of strong association rules.

Out of globally large itemsets all the possible association rules are designated. Next, analogically to the computation of globally large itemsets exceeding support, exceeding confidence of rules is designated using Secure Sum described in 3.5. Exceeding confidence of rule  $X \rightarrow Y$  for the party  $S_i$  equals  $\sum_{i=1}^n (XY.sup_i - c * X.sup_i)$ , where  $XY.sup_i$  means a local support for the set  $X \cup Y$ , and  $c$  is the minimal support threshold. Rules that meet minimal support threshold are sent to all the sites as a result.

The algorithm used in the *semi-honest* model is not fully secure under the definitions of secure multiparty computation. It should be pointed out, that similar to KCS it reveals the number of itemsets having a common support between sites (.e.g. party No. 2, 4, and 6 all support some itemset.) without revealing the content of these itemsets. By comparison, in computation with a trusted third party this information is not revealed.

## 5 CDKRSU Algorithm

The CDKRSU algorithm is similar to the CDKSU algorithm, except that (in contrast to the latter and to KCS) its security in the semi-honest model corresponds to that provided in computations with a trusted party. No information on the number of commonly supported item sets is leaked. To hide locally large item sets duplicated between the sites, the SDR (Secure Duplicate Removing) protocol is applied at the beginning of the process of secure computation of the union of locally large item sets. After encrypting their locally large item sets, the sites send them to other nodes using the SDR protocol. The protocol compares locally large sets in two consecutive nodes without revealing their content. If two identical sets are detected, one of them is removed without revealing that fact to the duplicated-set owners. The removal is carried out so that the owners do not know that one of the copies is removed. To avoid removing a duplicate occurring in all nodes, the duplicate removal must ensure that no loop is created. Because of that, the first transmission of encrypted locally large sets between one pair of sites is conducted without the SDR protocol. One of these sites must have an even identifier, and the other an odd identifier. Unions of sets of even and odd sites are computed separately (as described in [25]), so that the number of sets simultaneously supported by such two sites is hidden. After completing the full encryption cycle during computing the union of locally large sets, no duplicate sets occur anymore. The duplicated sets are encrypted only once.

To improve efficiency, it is possible to generate a random data block of a proper length instead of computing a cryptographic hash of a fake set. Such approach is advantageous if the pseudo-random generator is more efficient than the hashing function.

## 6 Implemented System

For research purposes, the system has been implemented in Java. Such implementation allows to compare efficiency of each algorithm without favoring any of them. The system consists of a set of multithreaded components corresponding to the building blocks used to build the multiparty data mining algorithms. The blocks are connected in a declarative manner (using the Inversion of Control container), so that data flows are created. All connections are defined in configuration files to enable the implementation of new algorithms. It is possible to define local and network connections between components to build real multiparty systems. All algorithms have been built in the same framework, and any differences in the implementation correspond exactly to the differences in the algorithm logic. Even those elements in which the algorithms differ have been built, as far as possible, from the same components and using exactly the same technology.

## 7 Performance Tests

In order to compare the efficiency of implementations of algorithms performance tests were conducted. Additionally the performance of CDKSU algorithm with exponential Pohlig-Hellman cipher (PH) against CDKSU with Elliptic Curve Pohlig-Hellman (ECPH) cipher was compared.

### 7.1 Test Environment

The system in the present test configuration consisted of four sites. Each party had two computers (Microsoft Windows XP Professional SP3; *IntelCore<sup>TM</sup>2* Quad 2.66GHz; 2GB RAM; HD: 250GB, SATA2, 7200rpm, 16MB cache) – one reserved for the database, and the second for the system engine. Computers were connected to the LAN with a throughput of 80.0 Mbits / sec. Each party had Oracle 11g database.

### 7.2 Test Data Sets and Parameters

Performance tests were conducted on a randomly generated test data. It was generated in such a way that it resembled the sales transaction data which is typical of the data analyzed in the process of discovering association rules. A periodical exponential pseudorandom generator was used. It favored groups of elements that simulated sales transactions. The datasets with 200, 400, 600, 800, 1000, 1200, 1400 and 1600 thousands of transaction were prepared. The number of possible elements of the transaction for all data sets is 100 and the maximum number of elements in a single transaction is 30. On average transactions includes 15 elements. Each of the data set has been uniformly distributed between the four parties.

Due to the large, variable number of random distorting data, each measurement was performed in two versions. Pessimistic case was with the maximum amount of distorting data and optimistic with the minimum.

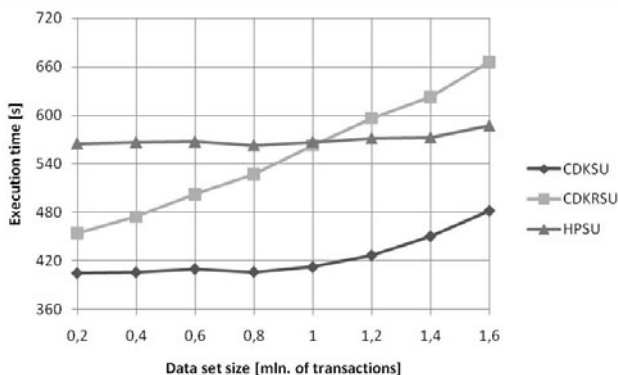
We ran tests with a various minimal support threshold. Preliminary tests confirmed that the minimum certainty threshold of a rule certainty has negligible impact so we run all the test with fixed 70% threshold.

When comparing the performance of the CDKSU algorithm with the exponential Pohlig-Hellman cipher (PH) against the CDKSU with the Elliptic Curve Pohlig-Hellman (ECPH) cipher encryption parameters were selected according to the guidelines from [30,34,1] to provide a similar protection for both algorithms. It corresponds to the protection afforded by a conventional 112-bit symmetric algorithm:

- Elliptic Curve Pohlig-Hellman cipher with prime239v1 parameters (with 239-bit prime number) as described in standards [30,34]. (In the graphs marked as ECPH.)
- Pohlig-Hellman with a 2048-bit key. (In the graphs marked as PH.)

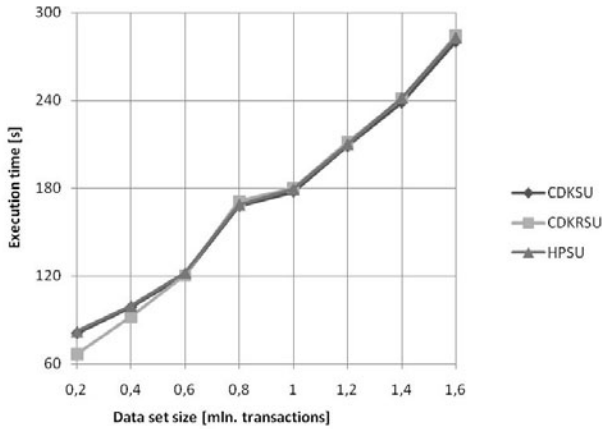
### 7.3 Test Results

The diagrams below shows the results of comparative performance tests. It should be noted that in addition the functional tests were also performed.

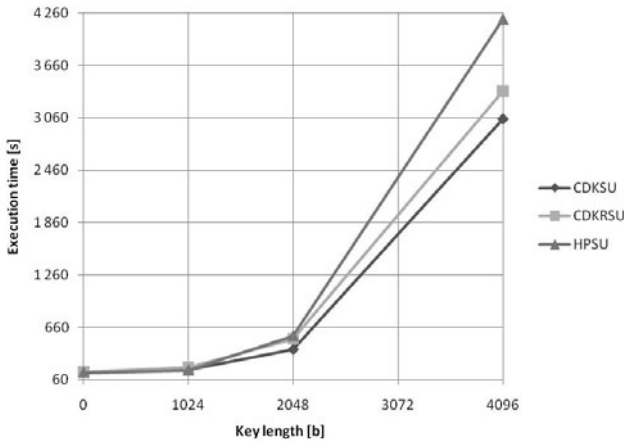


**Fig. 1.** Comparison of CDKSU,CDKRSU and KCS (HPSU). Pessimistic execution time versus data set size. The 2048b Pohlig-Hellman cipher was used together with a 15% minimum support threshold.

**Comparison of CDKSU, CDKRSU and KCS (HPSU).** On most charts, it appears that CDKSU algorithm has better performance than KCS. Especially figure 1 shows the difference between KCS and CDKSU very clearly. It illustrates the pessimistic case where the number of candidates is equal to the maximum number of locally large itemsets with frequent fake sets. All the fake locally sets



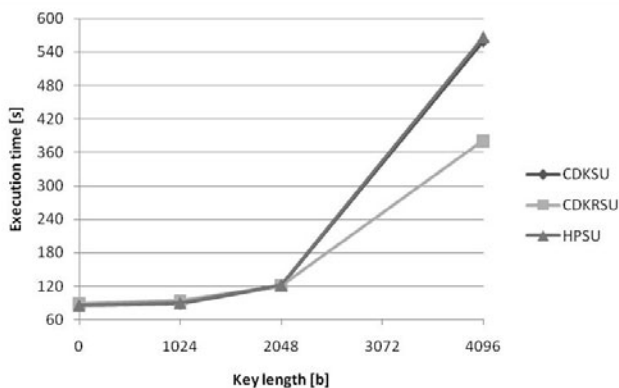
**Fig. 2.** Comparison of CDKSU,CDKRSU and KCS (HPSU). Optimistic execution time versus data set size. The 2048b Pohlig-Hellman cipher was used together with a 15% minimum support threshold.



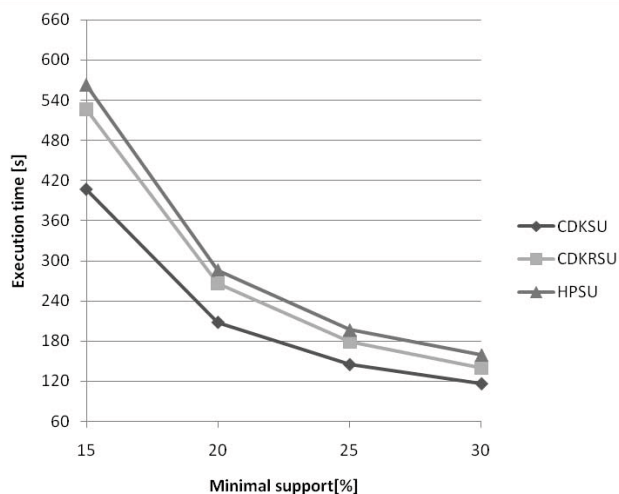
**Fig. 3.** Comparison of CDKSU,CDKRSU and KCS (HPSU). Pessimistic execution time versus key size. The Pohlig-Hellman cipher was used together with a 15% minimum support threshold and the data set that has 800k transactions.

are part of the union and are decrypted. CDK significantly reduces the cost of decryption. Figure 1 shows also that in the CDKRSU algorithm, using SDR imposes an additional cost, in particular due to comparing the fake duplicates. With the largest test sets, for all algorithms an exponential curving of the characteristics is visible due to the system working near the performance limit. Figure 2 shows that, in optimistic case no significant performance differences between the three algorithms exist. Optimistic tests differed only in the amount of transferred data as shown on figure 7.





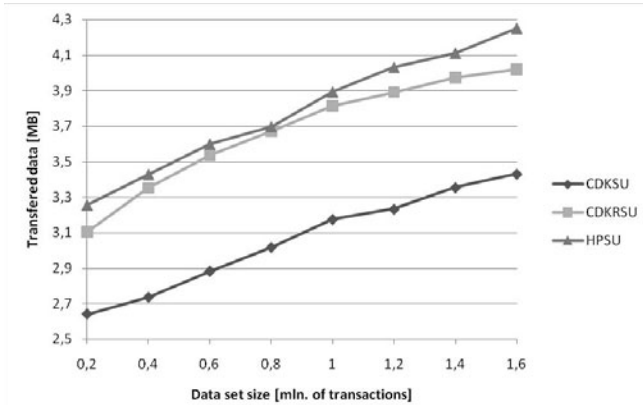
**Fig. 4.** Comparison of CDKSU,CDKRSU and KCS (HPSU). Optimistic execution time versus key size. The Pohlig-Hellman cipher was used together with a 15% minimum support threshold and the data set that has 800k transactions.



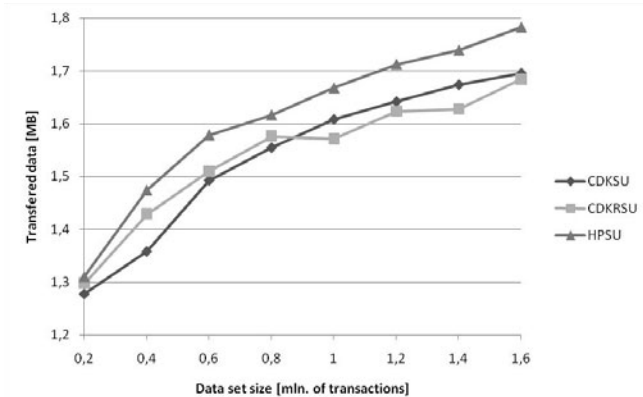
**Fig. 5.** Comparison of CDKSU, CDKRSU and KCS (HPSU). Pessimistic execution time versus minimum support threshold. The 2048b Pohlig-Hellman cipher was used together with the data set with 800k transactions.

Figures 3 and 4 show the exponential dependence between the key length and the algorithm execution time. It results from the computational complexity of exponential ciphers.

Figure 4 shows an interesting property of the CDKRSU algorithm. If the cost of encrypting a duplicated locally large set is higher than the cost of applying SDR, using the CDKRSU algorithm is advantageous not only in terms of security, but also in terms of efficiency. Figure 5 shows differences in pessimistic execution



**Fig. 6.** Comparison of data transferred by CDKSU, CDKRSU and KCS (HPSU). Pessimistic amount of transferred data versus data set size. The 1024b Pohlig-Hellman cipher was used together with a 15% minimum support threshold and 70% minimum support confidence threshold.

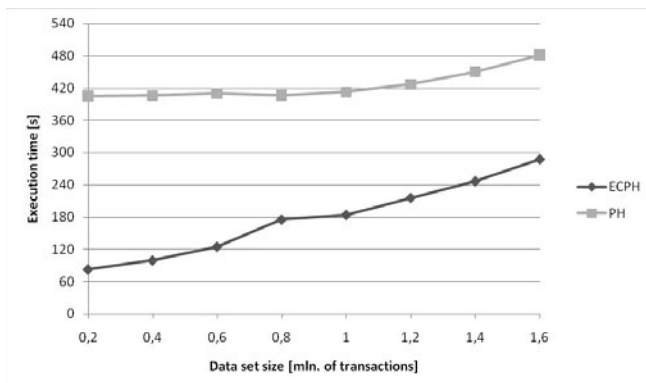


**Fig. 7.** Comparison of data transferred by CDKSU, CDKRSU and KCS (HPSU). Optimistic amount of transferred data versus data set size. The 1024b Pohlig-Hellman cipher was used together with a 15% minimum support threshold and 70% minimum support confidence threshold.

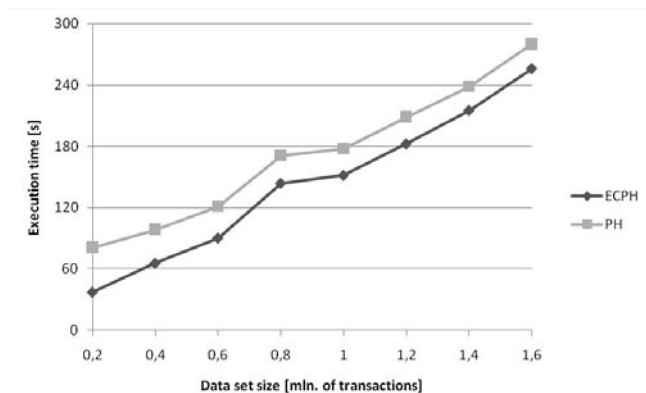
times, as well as the asymptotic decrease of the execution time depending on the minimum support threshold, which is a typical feature of association-rule discovery algorithms based on the Apriori algorithm. Figure 6 shows that the elimination of the cyclical decryption thanks to using CDK decreases the transfer between the sites, as well as that the SDR cost in CDKRSU is lower than the gain resulting from such decrease. Figure 7 clearly shows the logarithmic relationship between the input data set size and the execution time. It should be pointed out

that in the tested implementation, the amount of data transferred between the sites is so small compared to the network capacity that it has no impact on the algorithms efficiency.

Execution times to a small extent depend on the size of the input data set size because of the fixed number of possible elements of the transaction. The number of possible elements of the transaction determines the number of candidates in the first round. Their number is so large that it determines the execution time.

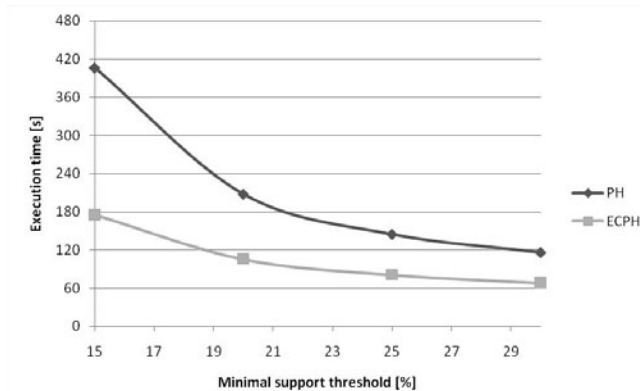


**Fig. 8.** Comparison of CDKSU algorithm with exponential cipher against CDKSU with elliptic curve cipher. Pessimistic execution time with 15% minimal support threshold, versus input data set size.



**Fig. 9.** Comparison of CDKSU algorithm with the exponential Pohlig-Hellman cipher (PH) against CDKSU with the Elliptic Curve Pohlig-Hellman (ECPH) cipher. Optimistic execution time with 15% minimal support threshold, versus input data set size.

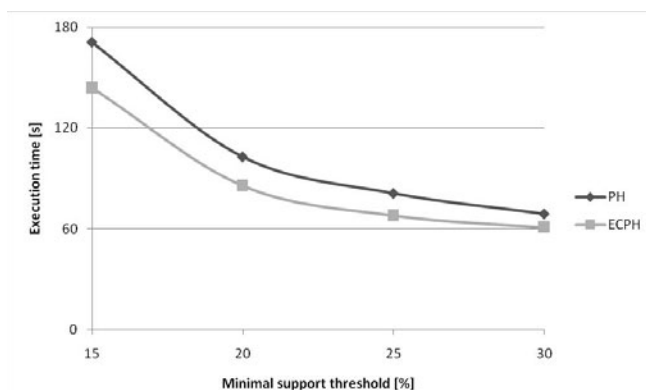
**Comparison of CDKSU Algorithm with the Exponential Cipher Against CDKSU with the Elliptic Curve Cipher.** All performance Tests have shown that the use of ECC greatly sped up the algorithm. Performance gain is not linear and depends both on the size of the input set and the minimum support threshold. This is clearly shown especially in figures 8 and 10 illustrating the pessimistic case. There is also visible that when using encryption based on elliptic curves the efficiency of the entire algorithm depends mainly on the size of the input data set. It appears that the minimum support threshold have much less impact. Reducing the productivity gap with the increase in the size of the test set resulted from the increasing databases overload. This was caused by the fact that the database hard drives were not designed for parallel access.



**Fig. 10.** Comparison of CDKSU algorithm with the exponential Pohlig-Hellman cipher (PH) against CDKSU with the Elliptic Curve Pohlig-Hellman (ECPH) cipher. Pessimistic execution time versus minimal support threshold. Input data set has 800k transactions.

## 8 Conclusion

Tests have shown that the efficiency of the applied cryptographic algorithm contributes to the efficiency of the whole algorithm to a great extent. The tests have shown a great increase in the efficiency of the CDKSU algorithm when compared with KCS, which is due to the use of CDK. In the CDKRSU algorithm, using SDR imposes an additional cost but it is not crucial when we want to use a fully secure algorithm. Also the tests have shown an increase in the speed of processing by 2 to 6 times in the pessimistic case of the CDKSU algorithm due to the use of ECC. As it was expected, elliptic curve cryptosystems are more computationally efficient than the exponential cryptosystems. The computational cost of the encryption based on the curves (with the protection equivalent to 112-bit symmetric encryption algorithms) is approximately six times smaller [1]. Although elliptic curve arithmetic is slightly more complex. It was not clear how this affected the



**Fig. 11.** Comparison of CDKSU algorithm with the exponential Pohlig-Hellman cipher (PH) against CDKSU with the Elliptic Curve Pohlig-Hellman (ECPH) cipher. Optimistic execution time versus minimal support threshold. Input data set has 800k transactions.

efficiency of the whole data mining algorithm. Such increase in the efficiency is crucial, since in the production environment, the input data sets can be much bigger. To our knowledge, this is the first paper describing the practical use of the Elliptic Curve Pohlig-Hellman cipher and its implementation tests. Also, we have not come across anybody else studying the ECC used in the multiparty data mining. In further research, the prospect of the use of elliptic curve homomorphic ciphers seems to be interesting. Of crucial importance is the performance gain that stems from the fact that ECC can be used in algorithms proposed in [45,47,42]. Examples of such homomorphic ciphers can be found in [32]. The execution time is difficult to predict not only because of the arrangement of the input data, but the considerable influence of random data used to conceal the data structure belonging to one party as well. The efficiency of the mentioned algorithms has not been fully tested. Closer attention should be devoted to the influence of the arrangement of the input data and the efficiency of the algorithms whose data are real and not artificially generated should be tested.

## References

1. U. N. S. Agency. The Case for Elliptic Curve Cryptography web page, [http://www.nsa.gov/business/programs/elliptic\\_curve.shtml](http://www.nsa.gov/business/programs/elliptic_curve.shtml)
2. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: PODS 2001: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 247–255. ACM, New York (2001)
3. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 86–97. ACM Press, New York (2003)

4. Agrawal, R., Shafer, J.C.: Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering* 8(6), 962–969 (1996)
5. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) *Proceedings of 20th International Conference on Very Large Data Bases VLDB 1994*, Santiago de Chile, Chile, September 12–15, pp. 487–499. Morgan Kaufmann (1994)
6. Atallah, M.J., Du, W.: Secure Multi-Party Computational Geometry. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) *WADS 2001*. LNCS, vol. 2125, pp. 165–179. Springer, Heidelberg (2001)
7. Blake, I.F., Seroussi, G., Smart, N.P.: *Elliptic curves in cryptography*. Cambridge University Press, Cambridge (2000)
8. Canetti, R., Ishai, Y., Kumar, R., Reiter, M.K., Rubinfeld, R., Wright, R.N.: Selective private function evaluation with applications to private statistics. In: *PODC 2001: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, pp. 293–304. ACM, New York (2001)
9. Canny, J.: Collaborative filtering with privacy. In: *SP 2002: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 45–57. IEEE Computer Society, Washington, DC, USA (2002)
10. Cheung, Han, Ng, Fu, Fu: A fast distributed algorithm for mining association rules. In: *PDIS: International Conference on Parallel and Distributed Information Systems*, pp. 31–42. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD, Los Alamitos (1996)
11. Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., Zhu, M.Y.: Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations* 4 (2003)
12. Estivill-Castro, V., Brankovic, L.: Data Swapping: Balancing Privacy Against Precision in Mining for Logic Rules. In: Mohania, M., Tjoa, A.M. (eds.) *DaWaK 1999*. LNCS, vol. 1676, pp. 389–398. Springer, Heidelberg (1999)
13. Evfimievski, A., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. In: *KDD 2002: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 217–228. ACM, New York (2002)
14. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: Cachin, C., Camenisch, J.L. (eds.) *EUROCRYPT 2004*. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
15. Frieser, J., Popelinsky, L.: Dioda: Secure mining in horizontally partitioned data. In: *International Workshop on Privacy and Security Issues in Data Mining, PSDM 2004* (September 2004)
16. Fu, A.W.-C., Wong, R.C.-W., Wang, K.: Privacy-preserving frequent pattern mining across private databases. In: *ICDM 2005: Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 613–616. IEEE Computer Society, Washington, DC, USA (2005)
17. Goethals, B., Laur, S., Lipmaa, H., Mielikäinen, T.: On Private Scalar Product Computation for Privacy-Preserving Data Mining. In: Park, C.-s., Chee, S. (eds.) *ICISC 2004*. LNCS, vol. 3506, pp. 104–120. Springer, Heidelberg (2005)
18. Goldreich, O.: Secure multi-party computation (2002) (manuscript)
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: *STOC 1987: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pp. 218–229. ACM, New York (1987)
20. Gorawski, M.: Advanced data warehouses. *Studia Informatica* 30(3B) (2009)

21. Gorawski, M., Slabinski, L.: Implementation of homomorphic encryption in privacy-preserving clustering distributed data. In: 2nd ADBIS Workshop on Data Mining and Knowledge Discovery, Thessaloniki, Greece September 6 (2006); in conjunction with 10th East-European Conference on Advances in Databases and Information Systems ADBIS 2006, pp. 37–47 (September 2006)
22. Gorawski, M., Stachurski, K.: On efficiency and data privacy level of association rules mining algorithms within parallel spatial data warehouse. In: First International Conference on Availability, Reliability and Security (ARES 2006), pp. 936–943. IEEE Computer Society (2006)
23. Jagannathan, G., Pillaipakkammatt, K., Wright, R.N.: A new privacy-preserving distributed k-clustering algorithm. In: SDM (2006)
24. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: KDD 2005: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 593–599. ACM, New York (2005)
25. Kantarcioglu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowl. Data Eng.* 16(9), 1026–1037 (2004)
26. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: ICDM 2003: Proceedings of the Third IEEE International Conference on Data Mining, page 99. IEEE Computer Society, Washington, DC, USA (2003)
27. Kolbitz, N.: *A Course in Number Theory and Cryptography*. Springer, Heidelberg (1994)
28. Lindell, Y., Pinkas, B.: Privacy Preserving Data Mining. In: Bellare, M. (ed.) *CRYPTO 2000*. LNCS, vol. 1880, pp. 36–53. Springer, Heidelberg (2000)
29. Meng, D., Sivakumar, K., Kargupta, H.: Privacy-sensitive bayesian network parameter learning. In: ICDM 2004: Proceedings of the Fourth IEEE International Conference on Data Mining, pp. 487–490. IEEE Computer Society, Washington, DC, USA (2004)
30. N.I. of Standards and Technology. FIPS PUB 186-2: Digital Signature Standard (DSS). National Institute for Standards and Technology, Gaithersburg, MD, USA (January 2000)
31. O’Leary, D.E.: Some privacy issues in knowledge discovery: The oecd personal privacy guidelines. *IEEE Expert: Intelligent Systems and Their Applications* 10(2), 48–52 (1995)
32. Paillier, P.: Trapdooring Discrete Logarithms on Elliptic Curves Over Rings. In: Okamoto, T. (ed.) *ASIACRYPT 2000*. LNCS, vol. 1976, pp. 573–584. Springer, Heidelberg (2000)
33. Pohlig, S.C., Hellman, M.E.: An improved algorithm for computing logarithms over  $gf(p)$  and its cryptographic significance. *IEEE Transactions on Information Theory* 24, 106–110 (1978)
34. Qu, M.: Standards for efficient cryptography sec 2: Recommended elliptic curve domain parameters (2010)
35. Rizvi, S.J., Haritsa, J.R.: Maintaining data privacy in association rule mining. In: VLDB 2002: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 682–693, VLDB Endowment (2002)
36. Schneier, B.: *Applied Cryptography, 2nd edn. Protocols, Algorithms, and Source Code in C*. John Wiley & Sons (2002)

37. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: KDD 2002: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 639–644. ACM, New York (2002)
38. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: KDD 2003: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 206–215. ACM, New York (2003)
39. Vaidya, J., Clifton, C.: Privacy preserving naive bayes classifier on vertically partitioned data. In: 2004 SIAM International Conference on Data Mining (2004)
40. Vaidya, J., Kantarcioglu, M., Clifton, C.: Privacy-preserving naive bayes classification. VLDB J. 17(4), 879–898 (2008)
41. Wagstaff, J., Samuel, S.: Cryptanalysis of Number Theoretic Ciphers, 1st edn. Computational Mathematics Series. Chapman & Hall/CRC Press, Boca Raton, FL, USA (2003)
42. Wright, R., Yang, Z.: Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In: KDD 2004: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 713–718. ACM, New York (2004)
43. Subramaniam, H., Wright, R.N., Yang, Z.: Experimental Analysis of Privacy-Preserving Statistics Computation. In: Jonker, W., Petković, M. (eds.) SDM 2004. LNCS, vol. 3178, pp. 55–66. Springer, Heidelberg (2004)
44. Yao, A.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society, Washington, DC, USA (1986)
45. Yi, X., Zhang, Y.: Privacy-preserving distributed association rule mining via semi-trusted mixer. Data Knowl. Eng. 63(2), 550–567 (2007)
46. Zaki, M.J.: Parallel and distributed association mining: A survey. IEEE Concurrency 7(4), 14–25 (1999); Special issue on Parallel Mechanisms for Data Mining
47. Zhan, J., Matwin, S., Chang, L.: Privacy-preserving collaborative association rule mining. J. Netw. Comput. Appl. 30(3), 1216–1227 (2007)



# Segmenting and Labeling Query Sequences in a Multidatabase Environment

Aybar C. Acar<sup>1</sup> and Amihai Motro<sup>2</sup>

<sup>1</sup> Bilkent University, Department of Computer Engineering,  
Ankara 06800, Turkey

`aacar@cs.bilkent.edu.tr`

<sup>2</sup> George Mason University, Department of Computer Science,  
Fairfax, VA 22030 USA

`ami@gmu.edu`

**Abstract.** When gathering information from multiple independent data sources, users will generally pose a sequence of queries to each source, combine (union) or cross-reference (join) the results in order to obtain the information they need. Furthermore, when gathering information, there is a fair bit of trial and error involved, where queries are recursively refined according to the results of a previous query in the sequence. From the point of view of an outside observer, the aim of such a sequence of queries may not be immediately obvious.

We investigate the problem of isolating and characterizing subsequences representing coherent information retrieval goals out of a sequence of queries sent by a user to different data sources over a period of time. The problem has two sub-problems: *segmenting* the sequence into subsequences, each representing a discrete goal; and *labeling* each query in these subsequences according to how they contribute to the goal. We propose a method in which a discriminative probabilistic model (a Conditional Random Field) is trained with pre-labeled sequences. We have tested the accuracy with which such a model can infer labels and segmentation on novel sequences. Results show that the approach is very accurate (> 95% accuracy) when there are no spurious queries in the sequence and moderately accurate even in the presence of substantial noise (~70% accuracy when 15% of queries in the sequence are spurious).

**Keywords:** Data Management, Information Integration, Query Processing.

## 1 Introduction

In many database applications users perform complex tasks by breaking them down to a series of smaller, simpler queries the results of which then become terms in a larger expression. While the user interface may be presenting this to the user as a simple button click, multiple queries are usually being evaluated at the DBMS level to satisfy the request. In some cases this is consciously done by the users themselves, usually when their user interface does not allow complex

queries or if they are experimenting with the data in order to find the correct query.

Particularly in the area of virtual databases (multidatabases), this decomposition of queries into smaller components is a necessary part of the process. In a multidatabase the actual data resides in a multitude of data sources, each possibly with different schemas and access methods. Therefore a query posed to the multidatabase is broken into multiple components, each designed to retrieve a piece of the required information from the relevant local data source. For the purposes of this paper, we call the party posing the queries the ‘user’. However, this user may just as well be an automated application integrating data from different sources (e.g. a mash-up).

Query consolidation [1] is the reversal of this query decomposition process. A consolidator attempts to predict the global query that produced a set of smaller component queries as a result of decomposition. Although it is not possible to uniquely identify a global query that produces a given set of component queries, it is possible to estimate some of the likely global queries. The research on consolidation so far assumes that the local queries are available as a sound and complete set.

However, in the real world, queries from a given user arrive as a sequence over time. There is no marker that indicates which queries in a sequence constitute a meaningful set. In the absence of such information, all the consolidating system has are logs of queries for local sources. The set of queries that are part of a global query must therefore be extracted from the sequence in the logs or collated in real time as these queries arrive. Furthermore it would be very useful to the consolidation effort to individually label the queries in a segment with their probable roles in the bigger picture. The purpose of this paper is the investigation of this problem.

In Section 2, some of the relevant literature and basic methods are introduced. Section 3 defines the exact problem, discusses relevant issues and the information available in the solution of the problem. Section 4 details our approach to the problem using conditional random fields. Section 5 presents experimental results on two different scenarios. Finally Section 6 concludes the paper with a summary of findings and future work.

## 2 Background

### 2.1 Session Identification

Most of the previous literature on session identification and analysis of query logs is geared towards Web server access log mining. In access log mining, sequences of document requests from a Web server are analyzed instead of database queries. A second form of server log analysis is the analysis of keyword and parameter queries sent to search engines and other information retrieval services (e.g., digital libraries, travel sites, &c.) This type of analysis is based on clustering queries in order to find trends [2] and other clues as to the general information demands of the users in order to better design services [10].

In terms of session identification, the most prevalent and simplest method of isolating user sessions is the *timeout* method. In the timeout method, the user is assumed to have started a new task after a given amount of time has passed without activity. For Web information retrieval the threshold is reported [8] to be 10 to 15 minutes, for optimal separation of subsequent user sessions. Another method proposed in [6] is based on the fact that the users spend less time on auxiliary pages (navigation pages &c.) than on content pages. The method assumes that a session is finished when a user navigates to content page and times out. The timeout method is applicable to the database query environment as well, as a supportive addition to the method proposed in the following section.

Finally, a method is proposed in [5] called *maximal forward reference*. In this method the user is assumed to be in the same session as long as the next page requested has a link from the current page. This approach is relevant to the problem at hand as it uses a metric of relevance between two requests in order to cluster them into the same session. A similar approach is used in the method proposed in the following section, to cluster relevant queries together.

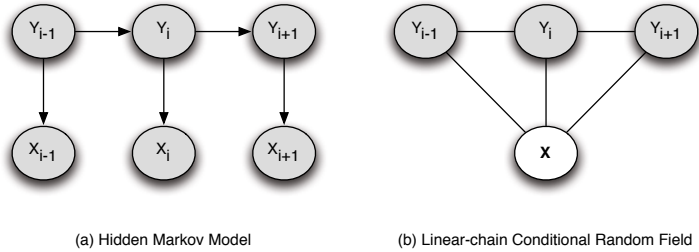
As far as query log analysis in database environments, there is one relevant study [18] that uses methods based on statistical language analysis in order to do session identification. Their method is based on training a statistical model with previous examples, predicting session boundaries where a sequence of queries becomes improbable with respect to previous experience. The method differs from the one proposed by this paper in that it requires an in-order arrival. These concepts will be discussed further in the following section.

## 2.2 Conditional Random Fields

A common approach to segmenting sequence data and labeling the constituent parts has been the use of probabilistic graphical models. Particularly, *hidden Markov models* (HMMs) [16] have been used for sequence data such as natural language text, speech, bio-sequence data and event streams. An HMM defines the joint probability distribution  $p(\mathbf{X}, \mathbf{Y})$ , where  $\mathbf{X}$  is a random variable over possible sequences of observations and  $\mathbf{Y}$  is a random variable over the possible sequences of labels (i.e. the possible states of the process generating said observations) that can be assigned to the same input. Because they define a joint probability, HMMs are termed *generative* models.

A generative model can be used to generate new sequences obeying a certain distribution defined by the parameters of the model. These parameters are either coded directly or optimized by using training data. However, given a certain sequence of observations, finding the most likely sequence of labels involves using Bayes' Rule and knowledge of prior probabilities. Furthermore, defining a joint probability requires enumerating all possible sequences, both for observations and labels. Given that the number of possible sequences increases exponentially with the sequence length, this is generally intractable unless strong assumptions are made about the independence of sequence elements and long-range dependencies thereof.

In contrast, a different class of graphical models, known as *discriminative models* model the conditional distribution  $p(\mathbf{Y}, \mathbf{x})$  for a particular sequence of observations,  $x$  and cannot generate the joint distributions. Since the problem at hand when trying to label a sequence of queries is finding the most probable sequence of labels given a sequence of observations (i.e. the observation sequence is already fixed), discriminative models are better suited to the task.



**Fig. 1.** Comparison of a first order hidden Markov model to a simple chain conditional random field

Such a discriminative model is the *Conditional Random Field* [12]. A conditional random field (CRF) can be regarded as an undirected graphical model, or Markov network [11], where each vertex represents a random variable. The edges of the graph represent dependencies between the variables. In the simplest case, the graph is a linear chain of state (label) variables  $Y_i$  dependent on the previous state and globally conditioned on the sequence of observations  $\mathbf{X}$  (see Fig. 1b) but higher order and arbitrary graphs are possible. As long as each state variable  $Y_i$  obeys the Markov property (i.e., is conditionally dependent only on its clique in the graph), the graph is a conditional random field.

Given this structure, it is therefore possible to factorize the joint distribution of  $\mathbf{Y}$  into potential functions, each one contained to a clique of vertices in the CRF. For example, in the case of a linear chain CRF such as that given in Fig. 1, the arguments of any such function will be  $Y_i, Y_{i-1}$ , and  $\mathbf{X}$ . As long as the feature functions are positive and real valued, the product will satisfy the axioms of probability, provided that it is normalized by a factor. Therefore the definition of any potential function  $f_k$  for a clique consisting of a particular observation sequence  $\mathbf{x}$  and elements  $y_i$  and  $y_{i-1}$  of a particular state sequence  $\mathbf{y}$  would be:

$$f_k(y_i, y_{i-1}, \mathbf{x}, i) = r \in \mathbb{R}, 0 \leq r \leq 1$$

The values of these potential functions can then be aggregated over the complete sequence to obtain the feature functions:

$$F_k(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^n f_k(y_i, y_{i-1}, \mathbf{x}, i)$$

In trying to segment and label a sequence of queries, we are interested only in *decoding* a particular state sequence  $\mathbf{y}$ . In other words, given a particular sequence of observations we would like to find out the sequence  $\mathbf{y}$  that would maximize  $p(\mathbf{y}|\mathbf{x})$ . This conditional distribution associated with the CRF is:

$$p(\mathbf{y}|\mathbf{x}, \lambda) = \frac{1}{Z(\mathbf{x})} \exp \sum_{k=1}^K \lambda_k F_k(\mathbf{y}, \mathbf{x}) \tag{1}$$

where  $Z(x)$  is the aforementioned normalization factor, and  $\lambda$  is the set of parameters (or weights) associated with each of the  $K$  feature functions. For any given application, the training of a CRF involves estimating these parameters. This parameter estimation is done using maximum likelihood training. Given a set of training sequences consisting of an observation sequence and the associated label sequence,  $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})\}$ , the product of Eqn. 1 over all the  $j$  pairs of sequences is the likelihood. In maximum likelihood training the set  $\lambda$  that maximizes the log-likelihood function:

$$\mathcal{L}(\lambda) = \sum_j \left[ \log \frac{1}{Z(\mathbf{x}^{(j)})} + \sum_k \lambda_k F_k(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}) \right] \tag{2}$$

is found. The log-likelihood function is concave and differentiable. However, equating the differential to zero does not necessarily result in a closed-form in terms of the parameters. Therefore the parameters that maximize the likelihood cannot found analytically [17]. However the problem is amenable to numerical solution by iterative scaling [3,9] or quasi-Newtonian methods [14].

### 3 Methodology

#### 3.1 Problem Overview and Assumptions

Throughout this paper, ‘goal’ will be used in the same sense as ‘session’. In previous literature, the term ‘session’ is used in multiple meanings. Occasionally, it is used to indicate the range of transactions between a login-logout cycle or during a given timeframe. In the scope of our problem, however, a session is generally meant to denote a single information retrieval task of the user, either done in a single query or by combining multiple queries. In order to avoid this ambiguity, the set of queries that make up a task of the user will be referred to as a goal.

We start by assuming that we can identify users by some identification mechanism such as a cookie, through authentication, from their network address &c. Therefore the query sequence we are observing is for a single user at a time. However, the sequence is not necessarily of queries sent to the same data source. This data can be collected in two ways. The user can be monitored directly, either using the logs of the client she is using directly, or through the logs of a proxy or gateway she uses to access the sources. In this case, the provenance of the queries in the sequence is more certain (unless someone else is using the

terminal). Alternatively, given a common identifier (such as IP or a cookie) we can monitor the logs of the data sources the subject uses. In this case, the queries belonging to the same user are collected from the various data sources and are compiled into a single sequence, in order of time stamps.

In either case we assume the existence of a list of queries in chronological order, coming from the same user. The queries, for the rest of this paper, will be labelled  $Q_i$  in a sequence  $\mathbf{Q} = Q_1 \dots Q_n$  such that  $Q_i$  is chronologically earlier than  $Q_{i+1}$ .

For each query  $Q_i$ , we assume to have access to a set of *intrinsic features*. These are given in Table [3.1](#).

**Table 1.** Intrinsic Features of Query  $Q_i$

Name	Notation	Description
Attributes	$\Pi_i$	The set of fields projected by the query.
Dependencies	$\mathcal{F}_i^*$	The functional dependency closure of fields in the query.
Constraints	$\mathbf{C}_i$	The set of constraints associated with the query.
Source	$D_i$	The unique identifier of the data source the query was sent to.
Answer Set	$\mathbf{R}_i$	The set of tuples returned in response to the query.
Timestamp	$t_i$	The universal time at which the query was posed.

We assume that these pieces of information are available for every query in the sequence. These intrinsic features will be used to generate more detailed feature functions which will in turn be used to train and decode the CRF.

We only consider conjunctive queries. Hence, each query in the sequence is represented by a Horn clause. For example, consider a query sent to an online bookstore's data service, asking the ISBN, title and year of books by Dickens:

$$Q_{ex} = \{(ISBN, Title, Year) | \exists AuthorID (Author("Dickens", AuthorID) \wedge Book(ISBN, Title, Year, AuthorID))\}$$

This query might then have the following intrinsic features:

$$\begin{aligned} \Pi_{ex} &= \{ISBN, Title, Year\} \\ \mathcal{F}_{ex}^* &= \{ISBN \rightarrow Title, ISBN \rightarrow Year, \dots\} \\ \mathbf{C}_{ex} &= \{(Author.Name = "Dickens"), (Book.AuthorID = Author.ID)\} \\ D_{ex} &= \text{http://soap.mybooks.com/Search.wsdl} \\ \mathbf{R}_{ex} &= \{(0679783415, "David Copperfield", 1850), \dots\} \\ t_{ex} &= 2011-04-24T20:39Z \end{aligned}$$

Given a query sequence and the intrinsic features of each member of the sequence, one needs to solve two distinct problems:

**Segmenting.** Otherwise known as *boundary detection*, this involves finding when one goal in the sequence ends and the other begins.

**Labeling.** Within each segment (goal) we also seek to find the function of each member query. This aids in reproducing the original goal exactly.

Both problems have analogues in natural language processing. The segmenting problem is analogous to *sentence boundary detection* and the labeling is very similar to *part-of-speech tagging*. In order to induce the most useful features for the task, we now consider a model for the sequence structure.

### 3.2 Assembly of Goals

In the simplest case, the goal,  $G$ , is the conjunction of the queries  $Q_i$  in the sequence. In terms of relational algebra, this is essentially the joining of the answer sets,  $R_i$ , into a single universal relation. However, it may be the case that the component queries do not offer a join path. Consider a sequence of two queries,  $Q_1$  and  $Q_2$  with attributes  $\Pi_1 = \{SSN, Name\}$  and  $\Pi_2 = \{Phone, Address\}$ . The straightforward conjunction (join) of these will result in a cartesian product, which is meaningless. Therefore, perhaps it is better to consider these two queries as two separate goals of one query each, rather than a goal of two queries. However, the arrival of a third query  $Q_3$  with attributes  $A_3 = \{SSN, Phone\}$  would change the conclusion. The three queries can now be joined into a goal consisting of three queries.

The problem of isolating goals from a sequence of queries can be stated as follows: Given a sequence of queries  $\mathbf{Q} = Q_1 \dots Q_n$  coming from a user, find all user goals within the sequence that are *maximal and coherent*.

We define a goal,  $\mathbf{G}$ , therefore, as the largest subsequence,  $\mathbf{G} \sqsubseteq \mathbf{Q}$ , of the whole query sequence  $\mathbf{Q}$ , that is cohesive (i.e. that has a join path). We formalize this idea of coherence by defining the following feature between any two queries  $Q_i$  and  $Q_j$ :

$$joinable(i, j) = \begin{cases} 1 & \text{if } \exists \mathbf{X} \subseteq \Pi_i, \mathbf{Y} \subseteq \Pi_j ((\mathbf{X} = \mathbf{Y}) \wedge (\mathbf{X} \rightarrow \Pi_i \vee \mathbf{Y} \rightarrow \Pi_j)) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The *joinable* function is thus 1 only if there is the possibility of a lossless join between the answer sets of the two queries. This lossless join property can be checked using the closures,  $\mathcal{F}^+$ , of the queries. Note that, in evaluating losslessness we only check functional dependencies and not inclusion. Considering the fact that constituent relations originate from independent sources, enforcing or requiring inclusion would not be realistic.

So in this simple case, a goal goes on as long as the incoming queries are joinable to the existing ones in the goal. We call these queries “subgoals”. Ultimately, all subgoals are joined to assemble the goals. We now consider subgoals that are composed of more than a single query.

### 3.3 Composite Subgoals

We define a composite subgoal as one that is created by the disjunction (union) of its constituent queries.

In constructing a composite subgoal, a user posing a sequence of queries to several sources is possibly collecting information from each of them in order to create a more comprehensive view of the information. An example would be to send a query selecting for the Italian restaurants in Washington D.C. to two or more different dining registries. The answers obtained are more likely to be unified rather than intersected since the aim here is to garner as many answers as possible.

Another case which requires aggregation rather than cross-referencing is when the complete answer cannot be obtained in a single query due to particular limitations of the data source. The most common reason for this are the *variable binding limitations* imposed by the query languages of the data sources.

As a simple example to this, consider an online telephone directory for some large organization. Generally, the service will not have the option of listing all the telephones at once. More likely, the directory will require the user to constrain (bind) at least one variable, such as the last name, department, &c. A user wanting the whole list will then be forced to ask for one department at a time, or worse, a name at a time. Monitoring such a series of queries, it would be wrong to label them as a conjunction when clearly the intention is to combine the answers into a large list.

The previous example also hints at another aspect of the problem: We have to assume that the user knew or obtained the list of departments (a table itself) from somewhere, then iteratively expanded each item with the associated telephone listings. Which in turn means that some query sequences may involve recursive queries. As a more elaborate example, consider a bibliography database that limits binding by requiring that some information about a paper (e.g. title) be given. The only way a user can get all papers, or at least as many papers as possible, is to start with one paper he knows and to recursively expand the citations until no new papers can be found.

Recursive plans may be used in case of poorly structured data services as well. Consider an airline reservation system which only lists direct flights given a source or destination airport. A user trying to get from Washington to Havana will not be able to find a direct flight. Instead, the user will ask for flights out of Washington and will ask followup queries asking for possible destinations from those places, until he can recursively find a path to Havana with as few stopovers as possible (e.g. Washington-Toronto-Havana). Again, the aim here is not to join the answers to these queries, so they need to be treated differently.

Each of these examples illustrate a different form of recursion. In the case of the bibliography example, a depth-first traversal is the best option. In the case of the airline example, breadth-first or iterative deepening is needed as depth first recursion will probably be unnecessarily expensive. A third form of traversal is the case where the user recurses using some form of heuristic. Assume the user wants to reach a certain person in a social network service. Similar to the



bibliography example, the only way she can accomplish this is to find a string of friends that can connect them. If the user knows that the person lives in a certain city, she may direct the recursion at each level to expand only people living in that city, knowing that this will result in finding a path more rapidly.

Ultimately, there is one common property of recursive query plans that allows and outside observer to identify them as such. Each query in such a sequence will bind one of its attributes to a value obtained from the result of a previous query in the sequence. We can formalize this using the intrinsic features of queries previously defined. The feature indicating that two queries  $Q_i$  and  $Q_j$  may belong to a recursive sequence is given as:

$$recursive(i, j) = \begin{cases} 1 & (i < j) \wedge \exists x \in (\mathbf{\Pi}_i \cap \mathbf{\Pi}_j); \exists y \in \pi_x(\mathbf{R}_i) [(x = y) \in \mathbf{C}_j] \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

In creating composite subgoals, a requirement is that the results of the queries, that are to be part of the subgoal, be *union-compatible* as defined in relational algebra. In other words, it is not possible to unify two results unless they have the same arity and indeed not meaningful unless they share the same attributes. Therefore, we need a measure to indicate the overlap in attributes between two queries. This can be simply stated as:

$$overlap(i, j) = \frac{\mathbf{\Pi}_i \cap \mathbf{\Pi}_j}{\mathbf{\Pi}_i \cup \mathbf{\Pi}_j} \quad (5)$$

The binary feature associated with overlap is the complete overlap:

$$completeoverlap(i, j) = \begin{cases} 1 & \text{if } overlap(i, j) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

An analogous and useful feature would compare the overlap in the set of constraints between two queries:

$$constraintoverlap(i, j) = \frac{\mathbf{C}_i \cap \mathbf{C}_j}{\mathbf{C}_i \cup \mathbf{C}_j} \quad (7)$$

Thus far, the features we have considered have been totally about the queries themselves. When working in a multidatabase environment, the provenance of the queries is also very useful. In other words, the source to which the query has been sent and from whence the answer originated might further help in defining the role of the query. Particularly if two or more subsequent queries are sent to the same source, there is a higher likelihood that the user is attempting a unification or recursion. In terms of query optimization it is more efficient to ask everything in one query per source. However, if for some reason (e.g. binding limitations or a less expressive query language at that particular source) the user is not able to accomplish their subgoal in one query, they will be required to collect the data in pieces. For example, a complete overlap between two queries sent to the same source is a good indicator that the user is combining subsets (e.g.

phone numbers in different departments) of the same information. We therefore introduce the following feature:

$$samesource(i, j) = \begin{cases} 1 & \text{if } D_i = D_j \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

### 3.4 Application of Conditional Random Fields

Apart from the complexity of assigning relevance to goals there is some difficulty associated with building the goals in the first place. Several aspects of the sequence will give clues as to the generation of goals:

*Do the goals arrive one at a time?* The question here is whether the user is done with one task before she starts another one. If this is not true, parts of different goals may arrive interleaved with each other. Without interleaving, the problem of goal identification is simply a problem of boundary detection, namely finding the first query of each goal. Otherwise, goals have to be extracted explicitly.

*Do goals evolve coherently?* This is related to whether the queries within a goal arrive in the correct order. For the goal to be coherent, each arriving query has to be relevant to at least one other query that is already part of the goal. If the queries arrive out of order in the worst case it is possible that there will not exist a coherent goal until the last query of the goal arrives. As an example consider the queries  $\Pi_1 = \{A, B, C\}$ ,  $\Pi_2 = \{K, L, M\}$ ,  $\Pi_3 = \{X, Y, Z\}$  and  $\Pi_4 = \{A, K, X\}$ . A goal consisting of these queries will evolve coherently only if  $Q_4$  arrives first. Whether or not this is assumed to be true will determine the solution method.

*Are queries used by multiple goals?* This factor determines the number of possible goals that can be generated given a sequence of length  $N$ . If we assume that each query can be part of multiple goals, the set of possible goals in the query sequence  $\mathbf{Q}$  will be the power set of  $2^{\mathbf{Q}}$ . If it is assumed that each query in  $\mathbf{Q}$  belongs to just one goal, the set of possible goals becomes the partition of  $\mathbf{Q}$ . The number of parts in  $\mathbf{Q}$  can be at most  $N$ , i.e. single-query goals.

In the simplest case, it can be assumed that queries arrive in order, in a non-interleaved fashion and that each query is part of a single goal only. In this case, the goal identification starts combining arriving queries into a goal. The first query to violate the coherence of the goal will be deemed the beginning of the next goal and the previous goal will be closed.

In the most complicated case, it is accepted that queries can arrive out of order, interleaved with queries of other goals and certain queries might be used by multiple goals. In this case the system will have to keep track of multiple goals at once, evolving some or all of them as new queries are received.

For the purposes of this paper, a middle ground between these two extremes is assumed. We assume that each query will be part of one goal only. Interleaving

will be neglected. The in-order arrival of queries is not a requirement to the extent that the sequence can be kept in memory. As will be seen subsequently, we work in a sliding window on the sequence starting from the last query going back as far as the window size. A query arriving prematurely will still be added into its goal as long as it is still within this window.

We start by defining a finite state machine which will take one query at a time and transition from one state to another. The state to which the FSM transitions to after the arrival of a new query will become the label of that query. The outline of the state machine is given in Fig. 2

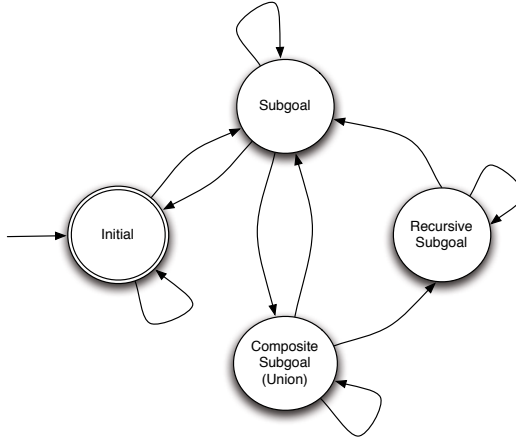


Fig. 2. Labeling Finite State Machine. Only relatively likely transitions are shown.

We now define this finite state machine as a probabilistic one, describing it with a conditional random field. The probability distribution is thus calculated as given in Eq. 1. In our case, the sequence  $\mathbf{x}$  is the query sequence  $\mathbf{Q}$  (i.e. the observations) and sequence  $\mathbf{y}$  is the sequence of states,  $\mathbf{L}$  the FSM traverses (i.e. the labels). Each transition of the FSM to the *Initial* state denotes the beginning of a new goal. After each arrival of a new query (observation) the CRF is decoded in order to find the state (label) sequence  $\mathbf{L}$  that results in the highest conditional probability  $P(\mathbf{L}|\mathbf{Q})$ . Hence, each time a new query arrives, the conditional probability is recalculated and may therefore change the labels on earlier queries as well.

Figure 3 shows an example sequence. Also provided in the figure are the operations performed on the queries by the user. The latter information is naturally not available to the party monitoring the queries but serve to illustrate the example. Table 2 illustrates the labeling for the incoming queries incrementally as each one arrives. The first two queries are joinable, therefore they are labeled as the initial query and a conjunctive subgoal. The third

query initially has no relationship to the preceding two. It is therefore labelled as the beginning of a new goal. The next few are found to be queries most likely to be combined (i.e. union-ed) with the third query and are thus labelled. The sixth query is again found to be unrelated to the existing goals and labelled as the beginning of a new goal. Query 7 happens to be the “keystone” query in that it has join paths to both query 6 and the previous goals. Therefore when query 7 arrives, the labels on the previous queries are changed, coalescing the three goals into one. This is possible since with each new query, the labels of all the previous queries are reexamined and a new Viterbi path<sup>1</sup> is calculated for the probabilistic FSM. Subsequently, query 8 arrives and is first deemed to be a single subgoal. However, as more evidence arrives, i.e. queries 9 and 10, it now becomes more probable that the three constitute a recursive subgoal and thus the labels are rearranged.

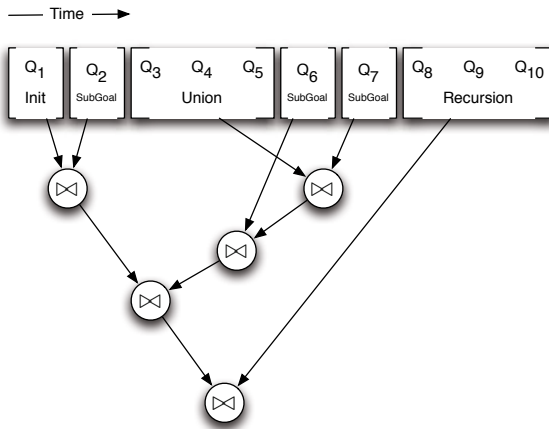


Fig. 3. Consolidation of Example Sequence

### 3.5 Features of the Conditional Random Field

In order to complete the definition of the CRF that represents this probabilistic FSM, the set of features  $F_k(\mathbf{L}, \mathbf{Q})$  have to be defined. There are two forms of features, namely *state* and *transition* features. The state features are defined in terms of identity functions. There is an identity function for each label type and each position in the sequence. For example the identity function defining whether the label  $L_i$  is the initial state would be:

$$initial(i) = \begin{cases} 1 & \text{if } L_i = INITIAL \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

<sup>1</sup> The sequence of transitions with highest probability.

**Table 2.** Evolution of Example Sequence

Time	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>	Q <sub>5</sub>	Q <sub>6</sub>	Q <sub>7</sub>	Q <sub>8</sub>	Q <sub>9</sub>	Q <sub>10</sub>
1	<b>I</b>									
2	I	<b>S</b>								
3	I	S	<b>I</b>							
4	I	S	I	<b>U</b>						
5	I	S	I	U	<b>U</b>					
6	I	S	I	U	U	<b>I</b>				
7	I	S	<b>U</b>	U	U	<b>S</b>	<b>S</b>			
8	I	S	U	U	U	S	S	<b>S</b>		
9	I	S	U	U	U	S	S	<b>R</b>	<b>R</b>	
10	I	S	U	U	U	S	S	R	R	<b>R</b>

I: Init, S: Subgoal, U: Union, R: Recursion  
 New or updated labels are shown in boldface

There will be one such identity function for each label per query. For example, given a sequence of 10 queries and the 4 labels defined in the FSM (i.e. Initial, Subgoal, Union, and Recursion), there will be 40 identity functions. Rather, we say that there are 4 state features per clique in the CRF.

The basic transition features have already been given in equations 3 through 8. Since conditional random fields are log-linear models, in order to capture the query model correctly we need to use conjunctions of the basic features (eqns. 3-8). For example, a compound feature particularly useful in determining recursion is (*completeoverlap*  $\wedge$  *recursive*  $\wedge$  *samesource*). Up to this point, for each clique in the CRF, we have 9 basic features (those given in Eqns. 3-8) and two features for absolute and relative timeout) and 4 state identity features. Along with the possible conjunctions (i.e. the powerset of the 13 features) we may consider up to 4095 features in evaluating the CRF.

We also need to define the scope of each of the features. Recall that a feature function is  $f_k(L_i, L_{i-1}, \mathbf{Q}, i)$ . However, the functions we have defined so far take two arguments (queries) to be compared according to some criterion (e.g., joinability, recursive nature &c.). For each clique, one of these queries is the current one (i.e.,  $Q_i$ ). What this query is being compared to in the sequence is dependent on the scope. We propose two different scopes, namely local and global. The global scope effectively compares all the queries in the range to  $Q_i$  and succeeds if there is at least one successful comparison. For example the feature  $f_{samesource}^{global}(L_i, L_{i-1}, \mathbf{Q}, i)$  would succeed if there is at least one other query in the sequence that has the same source as  $Q_i$ . Therefore given a basic feature  $g_k(Q_i, Q_j)$  a global feature is then:

$$f_k^{global}(L_i, L_{i-1}, \mathbf{Q}, i) = \max_{\forall j \neq i} (g_k(Q_i, Q_j))$$

In contrast, a local feature only checks the queries proximal in the sequence to  $Q_i$ . Hence, for example for a window of 5, the local feature would be:

$$f_k^{local}(L_i, L_{i-1}, \mathbf{Q}, i) = \max\left(\max_{i-5 < j < i} (g_k(i, j)), \max_{i < j < i+5} (g_k(i, j))\right)$$

Note, however, that the state features previously mentioned and any conjunctions containing them can only be local within a window of 2. Otherwise, the probability of a given label would not be independent of previous states except the neighboring one, violating the Markov property.

The final enrichment to the feature space involves the *time-shifting* of features, such that clique potentials can be affected by previous and future observations. A time shifted feature  $f_i^{<j>}$  would be defined as:

$$f_i^{<j>} = f(Q_{i+j}) \tag{10}$$

So, a feature  $f_i^{<-2>}$  would be the said feature for the observation two queries before  $Q_i$  used to evaluate the potential for the  $i$ -th clique in the CRF. Again this is limited to the transition features since the potential of a given clique cannot be a function of the label of state not neighboring it. Therefore, features containing any of the state identity functions can only be shifted by -1. According to these restrictions and assuming we use time shifts of -2, -1, 0, 1, 2, global and local forms for all the possible features, we may have up to 12278 features per clique.

### 3.6 Gain-Based Feature Selection

Note that the previous estimate is the combinatorial maximum of features obtained in the model we have thus far built. Some of these features will not be predictive at all and we thus need to prune those that have little utility in order to keep the model manageable and more importantly to avoid over-fitting the training data.

In order to do this, we use gain based feature selection. Recall that the training of a CRF involves optimizing the weight parameter vector  $\lambda$  in the log-likelihood function (Eqn. 2) in order to obtain the maximum likelihood. This is done in supervised way by iteratively maximizing for a set of training data. Each training sequence of queries has an associated sequence of correct labels. Thus we are able to optimize the parameters,  $\lambda$ , to obtain the weights that will have the least training error. Since some of our features are expected to be less discerning, the changes in the associated weights will have less impact on the training error. We can therefore use the method described in [15]. The gain of a feature  $f$  is defined as:

$$G_A(f) = \max_{\mu} (\mathcal{L}_{A+f\mu} - \mathcal{L}_A)$$

where  $\mathcal{L}_{A+f\mu}$  and  $\mathcal{L}_A$  are the log-likelihoods of the CRF with or without the feature  $f$ , given the training data. If done naively, this gain calculation will be very time consuming, given the large number of features. Fortunately, there are several optimizations including mean field approximation and limiting the gain calculation only to mislabeled outputs (cf. [15] for details). Using this gain-based pruning we show in the next section that we are able to use approximately a tenth of the features while gaining better accuracy and better generalization.

## 4 Experimentation

We performed some simple experiments on the approach thus far explained in order to validate its accuracy and examine its behaviour. Two publicly available benchmark databases, TPC-H<sup>2</sup> and Mondial<sup>3</sup>, were used. Of these, TPC-H is the simulated database of a business including supplier and customer information, invoices, orders and so forth. It has 8 tables. These 8 tables, for the purposes of our experiment, are treated as different data sources. Furthermore, the larger tables have been vertically sliced into smaller parts, each one as a different source. This allows us to simulate vertically distributed data in a multidatabase environment. Therefore, with the subdivisions our TPC-H test scenario simulates 17 different sources in a simulated global virtual database (multidatabase) schema.

The Mondial database has a larger schema and is populated with real-world geopolitical data from several resources (e.g. the CIA World Fact Book). Originally it has 23 tables, the larger of which were likewise vertically divided in order to obtain 30 simulated data sources.

Some parts of the divided tables were given binding limitations (e.g., the Countries table in Mondial can only be searched by setting name of the country).

In order to create training and test data for the CRF, we then automatically generate 500 query sequences for each. This was done by first creating a universal relation for each test scenario. Random views of these universal relations were then set as targets and a query plan (i.e., a query sequence) was generated for each target, along with the labels for each of the components. Due to the differences of the schemas and the random nature of the target selection, the lengths of the query sequences are distributed over a range of values. For Mondial, the sequences were between 1 and 36 queries long, with a median of 10. For TPC-H the sequences were between 1 and 23 queries long, with a median of 7.

The 500 sequences were randomly concatenated into groups of 10, each group becoming a simulated query log for one user. The CRF was trained and tested using leave-one-out cross-validation. Each database scenario was run 50 different times with different training and test cases.

Three different CRF scenarios were experimented with. The CRF experiment is the basic approach with all the possible features. The PrunedCRF experiment is the CRF scenario trained with gain-based pruning of attributes during training. The training and testing programs were implemented in Java and the experiments were run on a 2.2 MHz Athlon 64 system with 1 GB of memory operating under FreeBSD 6. Each cross-validation experiment (i.e. 50 runs) was completed in times on the order of 1-2 hours, on this platform, depending on the number of features. The inference (labeling) of the average test group took around 1.5 seconds. The unpruned method (CRF) has exactly 12,768 features for both data sets, the pruned CRF had 1,065 and 983 features for Mondial and TPC-H, respectively.

---

<sup>2</sup> <http://www.tpc.org/tpch/>

<sup>3</sup> <http://www.dbis.informatik.uni-goettingen.de/Mondial/>

The training was done using gradient ascent, particularly the L-BFGS [14] method. To avoid overfitting, the likelihood function was regularized with a Gaussian prior with variance 5. After each round of training with 49 groups the remaining test group was labelled and the labels were compared to the true labels which were decided during the generation of the query sequences. The accuracy of a test is defined as the percentage of labels that are the same with the true labels. Table 3 presents the average accuracy of each cross-validation run, one per scenario.

**Table 3.** Labeling Accuracy Results

Noise Level	Method	Test Accuracy		Noise Level	Method	Test Accuracy	
		TPC-H	Mondial			TPC-H	Mondial
0 %	CRF	94.6%	91.2%	10 %	CRF	73.2%	70.6%
	PrunedCRF	96.3%	95.4%		PrunedCRF	82.8%	79.5%
5 %	CRF	86.3%	88.1%	15 %	CRF	60.3%	65.1%
	PrunedCRF	95.2%	92.3%		PrunedCRF	68.6%	69.3%

The robustness of the CRF method to noise was also tested during these experiments. For each cross-validation run, after the training of the CRF, the test sequence was tested for accuracy initially as it was. Subsequently the test cases were each “corrupted” by inverting the values of random observation features in the test sequence. For example, if the one query was actually joinable with another, the noise induction might flip the value showing the queries as non-joinable. With this method, we aim to simulate user and observation errors. The tests were run at noise levels of 5, 10, and 15%, relating to that fraction of observation features being corrupted. Table 3 shows the results.

The results are encouraging in that even at relatively high noise levels the feature pruned CRF and the content enriched CRF are still reasonably accurate (on average above 95% accuracy at 1-in-20 error in observation). The basic CRF seems to suffer immediately from the noise, probably due to the tendency of the unpruned features to over-fit the training data. Therefore, we see that pruned feature spaces have the additional benefit of lower generalization errors, even in the case of noisy input.

As a subset of the preceding analysis, we have also investigated the boundary detection capabilities of each model. This purely to test whether the system segments the sequences correctly, regardless of the correct labeling of these segments.

The F1 scores of the boundary detection are given in Table 4. The segmentation behavior of the CRF is much better than the more complicated task of labeling, with very high accuracies up to very high noise levels. As with the labeling task, the basic, unpruned CRF suffers as noise increases, for the same reasons.



**Table 4.** Segmenting Accuracy Results

Noise Level	Method	Accuracy (F-1)		Noise Level	Method	Accuracy (F-1)	
		TPC-H	Mondial			TPC-H	Mondial
0 %	CRF	.992	.997	10 %	CRF	.971	.971
	PrunedCRF	.999	.999		PrunedCRF	.982	.988
5 %	CRF	.984	.973	15 %	CRF	.912	.903
	PrunedCRF	.987	.982		PrunedCRF	.938	.962

## 5 Conclusion

This paper investigates the problem of isolating sets of queries denoting independent user goals from a continuous sequence, either in real time or from a log. The complexities involved can be classified into two categories. The first category of difficulties is defining the boundaries of a goal. The second difficulty is the determination of the individual parts within a goal, regarding their function in forming the goal.

Even if one has a totally objective way of evaluating a given set of queries as to whether they contain an interesting goal, there are still ‘technical’ difficulties. Given a sequence of queries the set denoting a goal is not necessarily an uninterrupted sub-sequence. Two or more goals may be interleaved with each other or with noise. Some queries in the sequence may be replacements or refinements of older queries. Furthermore, it is possible that the queries needed to build a goal do not arrive in the order they are used. This defeats machine learning methods (e.g. [18]) which are sensitive to the sequence in which queries arrive. Another question is whether the user being monitored is using the same query for multiple goals or one. This distinction affects the number of goals to be evaluated greatly<sup>4</sup>.

A probabilistic and noise-tolerant method of handling the problem has been introduced. The method can accept realtime streams, constantly evolving a working scenario as new queries arrive. Scenarios that have stopped evolving are dropped for fresh ones as new evidence arrive. Furthermore, the method proposed here leverages the evidence available exclusively in a virtual database environment, namely the source information, to guide its decisions.

A future direction is the addition of semantic feature based on the data. Such a method will require semantic information about the application domain either in a latent fashion or explicitly using the constraints of the schema along with a logical inference engine. Such an approach has met with some success in the area of semantic query caching [7] and optimization [4,13]. The same principle can be applied to the present problem as well.

<sup>4</sup> If the component queries are assumed to be re-used, the number of possible goals increases exponentially, as opposed to a linear growth otherwise.

## References

1. Acar, A.C., Motro, A.: Inferring user goals from sets of independent queries in a multidatabase environment. In: Ras, Z., Tsay, L.-S. (eds.) *Advances in Intelligent Information Systems*. SCI, vol. 265, pp. 225–243. Springer, Heidelberg (2010)
2. Beeferman, D., Berger, A.: Agglomerative clustering of a search engine query log. In: *Proceedings of Knowledge Discovery and Data Mining*, pp. 407–416 (2000)
3. Bilmes, J.: A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report ICSI-TR-97-021, University of Berkeley (1997)
4. Cardiff, J., Catarci, T., Santucci, G.: Semantic query processing in a heterogeneous database environment. *Journal of Intelligent and Cooperative Information Systems* 6(2), 151–192 (1997)
5. Chen, M.-S., Park, J.S., Yu, P.S.: Efficient data mining for path traversal patterns. *Knowledge and Data Engineering* 10(2), 209–221 (1998)
6. Cooley, R., Mobasher, B., Srivastava, J.: Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems* 1(1), 5–32 (1999)
7. Godfrey, P., Gryz, J.: Semantic query caching for heterogeneous databases. In: *Proceedings of Knowledge Representation Meets Databases*, pp. 6.1–6.6 (1997)
8. He, D., Goker, A.: Detecting session boundaries from web user logs. In: *Proceedings of the BCS-IRSG 22nd Annual Colloquium on Information Retrieval* (2000)
9. Jin, R., Yan, R., Zhang, J., Hauptmann, A.: A Faster Iterative Scaling Algorithm for Conditional Exponential Model. In: *Proceedings of the 20th Int. Conf. on Machine Learning*, pp. 282–289 (2003)
10. Joachims, T.: Unbiased evaluation of retrieval quality using clickthrough data. Technical report, Cornell University, Department of Computer Science (2002)
11. Kindermann, R., Snell, J.: *Markov random fields and their applications*. American Mathematical Society, Providence (1980)
12. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Proceedings of the 18th Int. Conf. on Machine Learning*, pp. 282–289 (2001)
13. Levy, A.Y., Sagiv, Y.: Semantic query optimization in datalog programs. In: *Proceedings of Principles of Database Systems*, pp. 163–173 (1992)
14. Liu, D., Nocedal, J.: On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming* 45(1), 503–528 (1989)
15. McCallum, A.: Efficiently inducing features of conditional random fields. In: *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2003)*, pp. 403–411 (2003)
16. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2), 257–286 (1989)
17. Wallach, H.: *Efficient Training of Conditional Random Fields*. Master's thesis, University of Edinburgh (2002)
18. Yao, Q., Huang, X., An, A.: A Machine Learning Approach to Identifying Database Sessions Using Unlabeled Data. In: Tjoa, A.M., Trujillo, J. (eds.) *DaWaK 2005*. LNCS, vol. 3589, pp. 254–264. Springer, Heidelberg (2005)

# Combining Resource and Location Awareness in DHTs

Liz Ribe-Baumann

Ilmenau University of Technology  
liz.ribe-baumann@tu-ilmenau.de

**Abstract.** Distributed hash tables are designed to provide reliable distributed data management, but present challenges for networks in which nodes have varying characteristics such as battery or computing power. Assuming that nodes are aware of their resource availability and relative network positions, this paper presents a novel distributed hash table protocol which uses nodes' resource levels to remove load from weak nodes, whose overuse may cause delays or failure, while using nodes' positions to reduce cross-network traffic, which may cause unwanted network load and delays. This protocol provides nodes with links that are physically near with high resource availability, and simultaneously provides scalability and an  $O(\log(N))$  routing complexity with  $N$  network nodes. Theoretical analysis and simulated evaluation show significant decreases in the routing and maintenance overhead for weak nodes, the physical distances that lookups traverse, and unwanted node failures, as well as an increase node lifetime.

## 1 Introduction

Distributed hash tables (DHTs) have received much attention over the past decade as an efficient, reliable approach to store large amounts of data in a distributed fashion. Well established DHTs such as the Content Addressable Network (CAN) [18] and Chord [21] were designed to route efficiently on homogeneous networks without location information. Since DHTs do not inherently differentiate between nodes' characteristics (for example, group associations or robustness), building a DHT on a heterogeneous network comes with many challenges. Moreover, DHTs typically route messages along nodes independent of their locations (causing unnecessary lookup delays and cross-network traffic) and must be specially designed to use location awareness. This paper considers systems on which DHTs benefit from both heterogeneous node treatment and location awareness.

Take for example a cloud environment, in which nodes have varying computing power and network connections: Nodes with high computing power should obtain more load than nodes with low computing power while routing hops should follow low-latency links in order to reduce cross-rack traffic and shorten lookup times. Or consider a second example in which widespread wireless nodes run on battery power (e.g. smart phones which are recharged at regular intervals) and

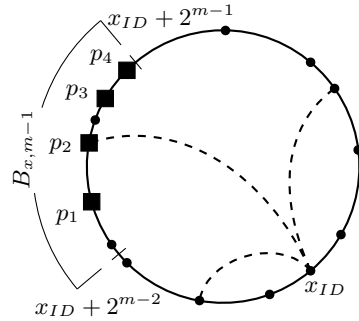
maintain a DHT using an intact infrastructure: Nodes with high power availability can handle more load than nodes with very limited power availability while latencies between nodes vary greatly depending on nodes' up and down links. While in both scenarios all nodes may cooperatively share the storage load, reducing the maintenance and routing load on weaker nodes would improve the networks' performance. Moreover, in the second case, minimizing low power nodes' maintenance and routing load can help to lengthen nodes' lifetimes between recharging. A homogeneous use of the nodes would ultimately result in shorter battery lifetimes and thus higher failure rates of low power nodes, thus effecting both the network's robustness and overall storage capacity. In this paper, we use this example of nodes with varying power availability as our point of reference. However, heterogeneous networks of nodes with varying bandwidth, time-to-live, computing power, or other suitable measures can be treated analogously, and therefore, our considerations are formulated in the general terms of network nodes' "resource availability."

The three main approaches to addressing heterogeneous node capabilities have been the use of hierarchical DHT structures, virtual nodes, or node movements within the identifier space, aiming to balance either communication or storage overhead. However, these approaches are difficult to adapt to our scenario. Firstly, the structural approaches which are capable of reducing the communication overhead of low resource nodes incorporate only two resource levels: "have" or "have not" (for example [110,25]). Secondly, the virtual nodes and node movement approaches, which allocate varying quantities of data to each physical node, actually introduce more maintenance overhead and churn into the network. Neither is ideal for a network in which nodes have varying access to resources such as energy or computing power, since they assume that higher resource availability infers larger storage capacity. However, considering that the main communication load in a DHT is accrued from maintenance, it is not nodes' storage capacities that need to be treated heterogeneously but rather the maintenance overhead required of the nodes.

This paper presents a novel resource aware and location aware DHT which reduces low resource nodes' load through routing and network maintenance while providing nodes with links which are physically near. We use two different failure models for testing: one which assumes that a node's failure depends on its resource availability and the number of messages it sends and receives (e.g. battery-power as nodes' resource), and a second which assumes that a node's resource availability is correlated with its failure probability (e.g. node lifetime as nodes' resource). We take a primarily structural, or link-based, approach to this problem, essentially transferring a large portion of maintenance and routing responsibilities to high resource nodes.

Our DHT is based on coordinates similar to and inspired by those used by Vivaldi [5] and structured similar to DHash++ [6], yet, it has significantly different behavior with regard to resources. In contrast to other flat DHTs, we choose links within finger intervals which have suitable combinations of high resource level and low distance (see Figure 1) and adjust the maintenance frequency of each link depending on its resource level. This paper's main contributions are:

<i>ProspectiveLinks</i> [ $m-1$ ]			
NodeID	Phy.Dist.	Res.Level	Res.Dist.
$p_2ID$	1.6	2	2.6
$p_3ID$	1.4	1	3.4
$p_4ID$	0.9	0	3.9
$p_1ID$	4.1	3	4.1



**Fig. 1.** Key ring with six nodes in  $x$ 's  $m - 1^{\text{st}}$  finger interval  $B_{x,m-1}$ , four of which  $x$  knows in its prospective links list (squares). A finger is established to  $p_2$ , the known node with the best resource distance (dependent on distance and resource level) to  $x$ .

- A novel flat DHT which incorporates both location and resource awareness;
- analytical observations of its routing complexity, links' resource levels and distances, link failure probabilities, and maintenance overhead; and
- simulated comparisons of node lifetime and message failures to resource naive and location aware DHTs with two failure models.

This paper includes an examination of related work in Section 2; discussion of foundational network conditions and assumptions in Section 3; a description of our novel DHT in Section 4; a brief discussion of analytical results in Section 5; and a comparison of the behavior of our DHT with existing DHTs using simulation in Section 6.

## 2 Related Work

Well established distributed hash tables such as CAN (Content Addressable Network) [18], Chord [21], and Kademlia [17] were designed to route efficiently without location information. Proximity-awareness has begun to permeate areas related to DHTs, including caching and replication protocols and hybrid overlays [7][16], as well as DHTs' original designs. Location awareness is integrated into DHTs using a combination of proximity-aware identifier selection (PIS, such as Mithos [22] and SAT-Match [19]), proximity-aware neighbor selection (PNS, such as DHash++ [6]), and proximity-aware route selection (PRS, such as Tapestry [24]), and has generally been directed at reducing overall traffic or average round trip times [12].

Now recall that the three main approaches to balancing load in heterogeneous DHTs are the use of hierarchical DHT structures, virtual nodes, and node movements within the identifier space. Within hierarchical DHTs, nodes are often grouped by some defining characteristic such as group associations (e.g. departments within a university) or peer capabilities ("have" or "have not"). Systems with group structures such as Canon [9], Hieras [23], and Cyclone [2

use routing protocols which tend to route lookups as far as possible within one group before forwarding them on to a different group. While resource availability levels could be used to define groups, current approaches pose several problems: lookup routing would have to start in the (sparse) highest layers in order to relieve weaker nodes from routing responsibilities, thus compromising scalability; low level nodes would primarily maintain links within their group, i.e. to low level nodes, increasing their necessary link maintenance; and location awareness could only be provided within single groups, i.e. resource levels. On the other hand, in hierarchical DHTs based on two-tiered peer capabilities (such as [1,10,25]) where nodes assume the roles of super-peer or leaf-peer, lookups are routed directly from leaf nodes to parent nodes. The parent (or super) nodes are fully responsible for performing lookup routing, completely neglecting the varying nuances of nodes' resource availabilities.

In contrast, with virtual nodes, each physical network node balances its load independently by hosting a varying number of virtual overlay nodes, each with its own set of keys and links [11,15]. And similar to virtual nodes, node movements within the identifier space achieve load balance by adjusting the data that each node stores [3,8]. Nodes with low load choose new nodeIDs that are close to nodes with high load, thus taking over some of their load.

Consider now, for example, a Chord implementation run on servers and smart-phones alike: servers have unlimited energy availability and large storage capacities while smart-phones have very limited energy and storage capacities. Since nodes' delays may vary greatly depending on their connectivity, fingers should clearly be chosen across low latency links (i.e. to "near" nodes). While virtual nodes or node movements could help to suitably distribute data, both would fail to relieve weaker nodes with dwindling energy of costly maintenance and routing responsibilities. On the other hand, the integration of a super-peer structure with "have" or "have not" nodes would invariably over-use or under-use the resources of the energy restricted devices. Subsequent node failures would then lead to a drop in the overall network capacity and thus compromise the network's scalability.

### 3 Network Assumptions

We assume that each node  $x$  has sufficiently correct two dimensional virtual (i.e. not necessarily geographical) network coordinates  $(x_1, x_2)$ , such as used in Vivaldi [5] for determining latencies. The *physical distance* between two nodes  $x$  and  $y$  is  $d_{phy}(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$ . Before proceeding, we establish a definition for nodes' resource availability levels and discuss what failure scenarios and underlying DHT properties this work is based on.

#### 3.1 Resource Availability

Recall that we base this work's resource awareness on nodes with varying battery strength, from cell phones with very limited power to servers with an inexhaustible power source. While we model our analysis and simulations after nodes

with varying power availability, the proposed protocol need not be restricted to this use case. It requires only that each node  $x$  has a resource availability that can be expressed as an integer value  $x_R \in \{0, 1, \dots, l_{max}\}$  for some fixed maximum level  $l_{max}$ . We assume that  $x_R = 0$  is the lowest possible resource level (but still operational) while  $x_R = l_{max}$  implies unbounded resources. Note that resource levels must be globally defined so that a given resource level on differing node types is comparable, i.e. nodes with identical resource levels have comparable available resources. If we consider power availability with  $l_{max} = 3$ , that could mean that a handheld operating on battery power may have resource level two when fully charged, but a cell phone with a weaker battery may only reach a resource level one when fully charged.

We use a Zipf distribution for nodes' resource levels, reflecting trends for node lifetime found in peer-to-peer networks, where node lifetime tends to follow a heavy-tailed Pareto distribution [4,20] (the continuous counterpart of the Zipf distribution). The probability that a random node has resource level  $\ell \in \{0, 1, \dots, l_{max}\}$  depends on the power  $m$  of the Zipf-distribution:

$$p_\ell := P(x_R = \ell) = \frac{1}{(\ell + 1)^m} \cdot \frac{1}{\sum_{j=0}^{l_{max}} 1/(j + 1)^m}. \tag{1}$$

### 3.2 Node Failure

In analysis and simulation, we approach node failure from two different perspectives: On the one hand, we assume that nodes with higher resource levels have lower failure probability (based on nodes with varying time-to-live); on the other hand, we assume that failure is caused by node activities which drain a node's resources until the node fails (based on nodes with varying power availability). For the later analysis of the first case, we need a node's conditional failure probability distribution  $P(F_x|x_R = \ell)$ , given the event  $x_R = \ell$ . For our observations, we again choose a Zipf-distribution and assume that this conditional probability is proportional to  $P(R = \ell)$ , i.e.  $P(F_x|x_R = \ell) = \alpha \cdot P(R = \ell)$ . Assuming that there are  $\gamma$  simultaneous node failures, we thus have the constraint:

$$\frac{\gamma}{N} = \sum_{j=0}^{l_{max}} P(F_x|x_R = j)P(x_R = j) = \sum_{j=0}^{l_{max}} \alpha P(x_R = j)^2. \tag{2}$$

Thus, we use  $P(F_x|x_R = j) = \alpha P(x_R = j)$  for  $\alpha = \gamma / \left(N \sum_{j=0}^{l_{max}} P(x_R = j)^2\right)$ .

### 3.3 DHT Foundation

We chose to build our DHT on Chord [21] mainly because Chord is the basis of the location aware DHash++ and has a rather simplistic structure. Our protocol is, in essence, an extension and could be adapted for many other DHTs. Analogous to Chord, we use consistent hashing [14] to distribute keys to nodes. Each node  $x$  chooses a random (or hashed) nodeID  $x_{ID}$  from the binary key space

$0 \dots 2^m - 1$ , which is viewed as a ring with key values increasing in a clockwise direction. Each node positions itself at its nodeID on the key ring and establishes links to its immediate predecessor and successor as well as a successor list with its  $r$  nearest successors, making repairs possible after unexpected node failures. Each key  $\kappa$  is assigned to the first node whose nodeID is equal to or succeeds  $\kappa$  on the key ring. The asymmetric *key distance* from a node  $x$  (or key) to a node  $y$  (or key) via their nodeIDs is:

**Definition 1 (Key Distance).** *The key distance from  $x$  to  $y$  is the clockwise distance on the key ring from  $x_{ID}$  to  $y_{ID}$ :*

$$d_{key}(x, y) := y_{ID} - x_{ID} \pmod{2^m}.$$

## 4 Resource and Location Aware DHT

Our novel DHT, which we call *RBFM* for resource based finger management, uses a flat approach to integrate resource awareness into nodes' links, as opposed to the more typically hierarchical approach. We recall from Section 2 that the typical flat approaches of virtual nodes and node relocation actually introduce additional maintenance overhead. We assume that lower resource nodes have an integral role in storing data and propose a system design which distributes data evenly on all nodes while addressing the maintenance and routing responsibilities of heterogeneous nodes. Heterogeneous data distribution and the for DHTs necessary replication are not included in the scope of this paper (see Section 7). Based on Chord, each node in RBFM chooses its links based on other nodes' key distances, physical distances, and resource levels - choosing for each finger interval a link with a balance of low physical distance and high resource level.

*Links.* Similar to DHash++ [6] and borrowing the terminology from Chord [21], each node  $x$  with nodeID  $x_{ID}$  chooses one link - or finger -  $x.f[i]$  per finger interval  $B_{x,i} := [x_{ID} + 2^{i-1}, x_{ID} + 2^i)$  for  $i \in \{1, 2, \dots, m\}$ . The corresponding node that  $x.f[i]$  points to is notated  $x.f[i].node$ . But while DHash++ chooses the node with the smallest physical distance to  $x$  in each finger interval, we chose a node based on both its physical distance *and* resource level, or its *resource distance*. This resource distance is inspired by Vivaldi's [5] distance metric for network coordinates which uses an additional height dimension to distance a node from the entire network. Similarly, we use a node's resource level to distance it from the entire network, ensuring that the lower a node's resource level is, the further it will be distanced from every other node. First, we define each node  $x$ 's resource height  $x_h$  via a resource height function  $h : \{0, 1, \dots, l_{max}\} \rightarrow \mathbb{R}^+$  for some stretch constant  $c > 0$ :

$$x_h = h(x_R) := c \cdot (l_{max} - x_R), \ell \in \{0, 1, \dots, l_{max}\}. \quad (3)$$



**Definition 2 (Resource Distance).** *The resource distance between nodes  $x$  and  $y$  with coordinates  $(x_1, x_2), (y_1, y_2)$ , resource levels  $x_R, y_R \in \{0, 1, \dots, l_{max}\}$ , and resource heights  $x_h = h(x_R)$  and  $y_h = h(y_R)$  is:*

$$d_{res}(x, y) = d_{phy}(x, y) + x_h + y_h.$$

In order to gain information about other nodes’ resource distances, coordinates and resource levels are piggybacked on network messages. Each node  $x$  maintains a *prospective links* list which contains a list of the  $k$  best known nodes in terms of resource distance for each finger interval  $B_{x,i}, i \in \{1, 2, \dots, m\}$ . Thus, at most  $k$  nodes in  $B_{x,i}$  with the shortest resource distances to  $x$  are saved via their nodeIDs and resource distances to  $x$ . When receiving a message from sender  $y$ , node  $x$  uses  $y$ ’s coordinates and resource level to determine  $d_{res}(x, y)$  and update its prospective links list accordingly (see Algorithm 1).

---

**Algorithm 1.** Updating prospective links list with  $\leq k$  entries

---

```

procedure SUGGESTPROSPECTIVELINK(nodeInfo)
    finger = getFingerInterval(nodeInfo.key)
    dist = getResourceDist(nodeInfo.coordinates, nodeInfo.resourceHeight)
    if prospLinkList.contains(finger, nodeInfo.key) then
        prospLinkList.updateNode(finger, dist, nodeInfo)
    else if dist < prospLinkList.getFarthestLinkDist(finger) or
        prospLinkList.size(finger) < k then
        prospLinkList.addNode(finger, dist, nodeInfo)
        while prospLinkList.size(finger) > k do
            prospLinkList.removeFarthestLink(finger)
        end while
    end if
end procedure

```

---

Each node  $x$  maintains a *finger table* with one finger  $x.f[i]$  in each  $B_{x,i}$  for  $i \in \{1, 2, \dots, m\}$ : if *prospective links* contains at least one entry for  $B_{x,i}$ , then the entry with the smallest resource distance is contacted with a finger request (see Figure 1); otherwise, the owner (i.e. successor) of key  $x_{ID} + 2^{i-1}$  is contacted as in Chord (see Algorithm 2). An entry from the prospective links list is deleted as soon as it is used for a finger request, ensuring that prospective links are up-to-date and alive. The prospective links list entries are also continually updated with fresh node information, so the network automatically adapts to changes in node resource levels or coordinates. Note that if there is a finger interval that contains no node, then multiple fingers will point to the same node, as in Chord. On the other hand, if there is at least one node in a finger interval  $B_{x,i}$ , then  $x.f[i]$  will point to a node in  $B_{x,i}$ .

As we will show, the larger  $i$  is (i.e. the larger the finger interval), the higher we can expect  $x.f[i]$ ’s resource level to be. This means that high resource level nodes tend to have more incoming fingers than low resource level nodes.

---

**Algorithm 2.** Establishing and maintaining fingers 1 to  $m - 1$

---

```

procedure MAINTAINFINGER(finger)
    lookupKey = myKey + getOffset(finger)
    if prospLinkList.size(finger) > 0 then
        listEntry = prospLinkList.getClosestEntry(finger)
        lookupKey = listEntry.key
        prospLinkList.removeUsedEntry(listEntry)
    end if
    sendLookup(lookupKey)
end procedure

```

---

*Routing.* Lookup routing is performed greedily, identical to unidirectional routing in Chord [21]: A node  $x$  which needs to lookup a key  $\kappa$  in  $0 \dots 2^m - 1$  forwards the lookup to the closest predecessor of  $\kappa$  in its routing table (including its *successor list* and its own nodeID  $x_{ID}$ ). If  $x$  is the closest predecessor, then the key is maintained by  $x$ 's successor, and the routing is completed after one hop.

Since fingers are not deterministically defined in this approach, allowing fingers to be spaced more irregularly, the expected (and worst case) number of hops necessary to locate a key is higher than in Chord. However, this increase can be expressed as a constant factor, leaving us with the same  $O(\log N)$  complexity as in Chord. In fact, simulation results show that the difference in routing lengths is in fact negligible. Note that we use the term *with high probability* to express a probability  $\geq 1 - 1/N$ .

**Theorem 1.** *Given a network with  $N$  nodes, with high probability, a message is routed from any node to the successor node of any key in  $O(\log(N))$  hops.*

*Proof.* Assume that a node  $x$  is to forward a message to the node  $y$  responsible for key  $\kappa$ , and let  $p$  be  $\kappa$ 's immediate predecessor node. We consider how many hops are necessary to reach  $p$ . Let  $p$  be in  $x$ 's  $i^{\text{th}}$  finger interval  $B_{x,i} = [x_{ID} + 2^{i-1}, x_{ID} + 2^i)$ . Then either:

- $x.f[i]$  is a predecessor to  $p$ , and forwarding to this finger reduces  $d_{key}(x, p)$  by at least  $2^{i-1}$ , or
- $x.f[i]$  is a successor to  $p$ . Since  $p$  is a successor to the key  $x_{ID} + 2^{i-2}$ ,  $x.f[i-1].node$  is in the interval  $[x_{ID} + 2^{i-2}, p_{ID}]$ . Forwarding to this finger reduces  $d_{key}(x, p)$  by at least  $2^{i-2}$ .

We see that the key distance  $d_{key}(x, p) \leq 2^i - 1$  is reduced by a factor of at least  $2^i / 2^{i-2} = 4$  for each forwarding. Thus, we are within one key of  $p$  after at most  $\log(3/4) \cdot m$  steps:

$$1 = 2^m (3/4)^{\log(4/3)}.$$

Furthermore, after  $\log(3/4) \cdot \log N$  steps we are within  $2^m/N$  keys of  $p$ , and - considering the consistent hashing used to generate nodeIDs and assuming that nodes are uniformly distributed in the keyspace - with high probability there are no more than  $O(\log N)$  nodes this keyspace. So  $p$ , and with it  $y$ , are reached in at most  $O(\log N)$  hops. □

*Node Joins and Failures.* In order to join the DHT, a node  $x$  must have valid network coordinates, choose a nodeID and resource level, and contact one participating node. Once  $x$  has established links to its immediate predecessor  $p$  and successor  $s$  on the key ring,  $p$  and  $s$  send their *prospective links* lists to  $x$ , which  $x$  uses to initialize its own list, and corresponding keys are transferred from  $s$  to  $x$ . The node  $x$  continually updates its *prospective links* and periodically performs finger maintenance (see Algorithm 2) to establish and maintain its fingers.

Node failure is handled as in Chord and extended to remove a prospective link once a node notices that it has failed. While replication is necessary to ensure data availability for ungraceful node failures, this is not included in the scope of this paper.

*Updating Links.* Given a dynamic network with frequent node joins, failures, movements, and changing resource levels, links must be updated on a regular basis to uphold the network's routing characteristics. Links to nodes which have failed or no longer have the minimum resource distance in a finger interval must be reassigned; links to newly joined nodes must be established. Note that a node's outgoing fingers do not change when its resource level changes, but its incoming links most likely will since its new resource level will eventually be updated in other nodes' prospective links lists. Since we correlate a node's resource availability with its robustness, we choose the frequency with which a finger  $f$  is updated depending on  $f.node$ 's resource level: Fingers to high resource level nodes require updates less often, since high resource nodes are less dynamic. This reduces the maintenance load for both lower resource nodes, which initiate link maintenance, and high resource nodes, which respond to the link maintenance.

We perform link maintenance on a finger  $f$  after a time interval which depends on a reference interval  $t_{ref}$  and the resource level  $f.node_R$ . One possible *finger maintenance interval*  $g : \{0, 1, \dots, l_{max}\} \rightarrow \mathbb{R}^+$  in dependency of a resource level  $\ell$  is, for example  $g(x_R) = t_{ref} \cdot (x_R + 1)^2$ .

By this function, a finger with resource level 0 would be updated after the interval  $time_{ref}$  and a finger with resource level 3 would be updated after the interval  $16time_{ref}$ . By adjusting the function  $g(\ell)$ , the developer has a direct possibility to tune the network's degree of robustness and maintenance overhead. While routing robustness and maintenance overhead usually imply a direct trade off, the observation of nodes' resource availability softens this trade off by decreasing the maintenance overhead of specific, not all, links. However, this is only the case when a node's failure or reliability is reflected by its resource level.

## 5 Analysis

We provide measures that indicate that nodes' fingers in RBFM are robuster and show how weak nodes' maintenance load in RBFM is significantly smaller when compared to the resource naive, location naive Chord and the resource naive, location aware DHash++. Since we do not consider replication protocols, we are

not interested in data loss. We expect RBFM to use lower level nodes less for routing, which balances load according to nodes' available resources; to decrease lookups' physical distance traveled compared to Chord (but not DHash++), which decreases cross-network load; to reduce link failure, which improves routing performance; and reduce maintenance overhead, which especially benefits low resource nodes. Dependencies between neighboring nodes' fingers make a global statistical analysis impractical, so we assess these criteria on a finger basis by comparing the resource level, physical distance, failure probability, and generated maintenance messages of nodes' individual fingers. While the former evaluations provide a mere anticipation for the system's expected behavior and require much interpretation, the evaluation of maintenance messages provides a clearer view of the systems' changed load.

Note that for our analytical observations, we assume that nodeIDs are uniformly distributed. In reality, nodeIDs are often a deterministic SHA-1 hash of node's IP addresses, but unless the SHA-1 function is tampered with (which is considered hard to do) it distributes nodes nearly uniformly in the key space.

### 5.1 Expected Resource Level and Distance of Fingers

We start with the physical distance traveled and lookups' used resource levels by determining the probability distributions of the physical distance and resource level of a random node  $x$ 's  $i^{\text{th}}$  finger. Since  $x.f[i]$  is looked for in a finger interval of size  $2^{i-1}$ , the larger  $i$  is, the more nodes  $x$  has to choose  $x.f[i].node$  from. Since  $x$  chooses the node to which it has the smallest resource distance, this node tends to be physically closer with higher resource availability as  $i$  increases. In fact, our analysis confirms that long key distance fingers tend to be physically close nodes with high resource levels (see Figure 2).

We derive these probability distributions for a node  $x$ 's  $i^{\text{th}}$  finger using the probability distribution  $P(x_R = \ell) = p_\ell$  for the resource levels from (III) and some distribution of nodes with respect to  $x$  in the coordinate space. This distribution is expressed as the probability that a random node will have a given physical distance to  $x$ . The following theorem gives us the probability distributions for the physical distance to and resource level of a finger chosen by  $x$  from a given number of nodes  $k$ .

**Theorem 2.** *Given is a node  $x$  and a set  $S$  of  $k$  nodes with resource levels in  $\{0, 1, \dots, l_{max}\}$ . Let  $y \in S$  be a node with the minimum resource distance to  $x$  in  $S$ ,  $f_D(t)$  be the probability density function (pdf) over all nodes' physical distances, and  $F_D(t)$  its cumulative distribution function. Then the probability that  $y$  has resource level  $\ell \in \{0, 1, \dots, l_{max}\}$  is*

$$P(R_{k,min} = \ell) = \underbrace{k}_{a.} \underbrace{p_\ell}_{b.} \underbrace{\int_0^\infty \left(1 - \sum_{j=0}^3 F_D(t + h(\ell) - h(j)) p_j\right)^{k-1} f_D(t) dt}_{c.} \quad (4)$$

and the pdf for  $y$ 's physical distance  $t \geq 0$  is

$$f_{D_{min}}(t) = k f_D(t) \sum_{\ell=0}^{l_{max}} p_\ell \left( 1 - \sum_{j=0}^{l_{max}} p_j F_d(t - h(j) + h(\ell)) \right)^{k-1}. \quad (5)$$

A proof is omitted due to space constraints, but we provide a rough sketch by explaining the terms in (4):

- a. Number of possible nodes that may have minimum resource distance.
- b. Probability that  $y_R = \ell$ .
- c. Probability that for all nodes  $v \in S/\{y\} : d_{res}(x, v) \geq d_{res}(x, y)$ .
- d. Probability that, with fixed  $d_{phy}(x, y) = t$  and  $y_R = \ell$ , for all nodes  $v \in S/\{y\} : d_{res}(x, v) \geq d_{res}(x, y)$ .

While node  $x$ 's  $i+1^{st}$  finger interval contains  $2^i$  keys, it only contains an expected  $k = \frac{N}{2^m} \cdot 2^i$  nodes. Choosing the node with the minimum resource distance to  $x$  from these  $k$  nodes, (4) gives us the probability that  $x$ 's  $i^{th}$  finger has resource level  $\ell \in \{0, 1, \dots, l_{max}\}$  and (5) gives us the pdf of its physical distance to  $x$ . Note that it is not possible to make any statements about the fingers' resource levels or physical distances without some assumption about the nodes' distribution within the coordinate space. For this reason, we consider one specific and simple case, where nodes are uniformly distributed around  $x$  on a disk of radius  $r$  (this means that we observe only the disk's center node):

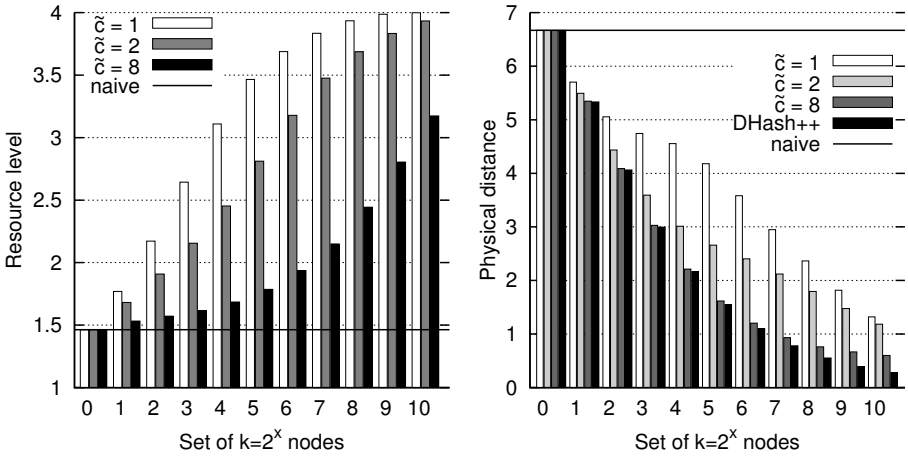
$$f_D^u(t) = \begin{cases} 2t/r^2 & 0 \leq t \leq r \\ 0 & \text{else} \end{cases}, \quad F_D^u(t) = \begin{cases} t^2/r^2 & 0 \leq t \leq r \\ 0 & \text{else.} \end{cases} \quad (6)$$

To simplify (4) and (5) for this distance distribution, we use a concrete instance of the height function  $h(x_R)$  from (3) with  $c := r/l_{max}\tilde{c}$ , which determines the resource height as a fraction of the network's physical radius  $r/\tilde{c}$  multiplied by  $(l_{max} - x_R)/l_{max}$ . Then using (4), we obtain a probability distribution which is independent of  $r$

$$\begin{aligned} P(R_{k,min} = \ell) &= \frac{2p_\ell}{r^2} \int_0^r \left( 1 - \sum_{j=0}^{l_{max}} F_D^u(t + h(\ell) - h(j)) p_j \right)^{k-1} t dt \\ &= 2p_\ell \int_0^1 \left( 1 - \sum_{j=0}^{l_{max}} F_D^{u'} \left( t + \frac{j - \ell}{c \cdot l_{max}} \right) p_j \right)^{k-1} t dt \end{aligned}$$

with

$$F_D^{u'}(t) = \begin{cases} t^2 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{else.} \end{cases}$$



**Fig. 2.** The expected resource level and physical distance of the node with minimum resource distance from sets of  $2^0, 2^1, \dots, 2^{10}$  nodes are shown for  $\mu = 2$  and  $r = 10$  along with the respective values for DHash++ and a resource and location naive Chord

The expected values of these probabilities are depicted in Figure 2 for  $l_{max} = 3$ , specific values of  $k$  ( $2^0, 2^1, \dots, 2^{10}$ ), and  $c = 1.5$ , as are the corresponding expected values for the physical distance to the node with minimum resource distance. Although we do not expect that  $x$  knows all of the nodes in each  $B_{x,i}$ , the expected number of nodes per finger interval doubles per interval and we presume that each node knows a fair number of nodes per finger interval. A node's  $\lceil \log(1/N) + m \rceil^{\text{th}}$  finger interval is the first in which a node is expected to be found, thus, each set of  $2^i$  nodes in Figure 2 corresponds to a node's  $i + 1 + \lceil \log(1/N) + m \rceil^{\text{th}}$  finger interval in a network of  $N$  nodes.

Figure 2 shows us how stretch affects fingers' resource levels and physical distances, with low stretch ( $\tilde{c} = 8$ ) favoring lower physical distances and high stretch ( $\tilde{c} = 1$ ) favoring higher resource levels in an apparent trade off. For a middle stretch of  $\tilde{c} = 2$ , a finger's expected resource level are doubled when there are  $2^6$  random nodes to choose from, while a mere  $2^4$  nodes are needed to reduce its expected physical distance by more than  $1/2$ . Note that the expected physical distance of a finger in a location unaware Chord is constant and given for  $k = 1 = 2^0$ , as is the expected resource level of both Chord and DHash++. Interpreting the results for the special case in Figure 2, we expect that resource and location aware fingers cause a higher number of routing hops to be sent across high level, physically close nodes, resulting in less traffic on low level nodes and less cross-network traffic, and ultimately resulting in robuster lookups.

### 5.2 Failures

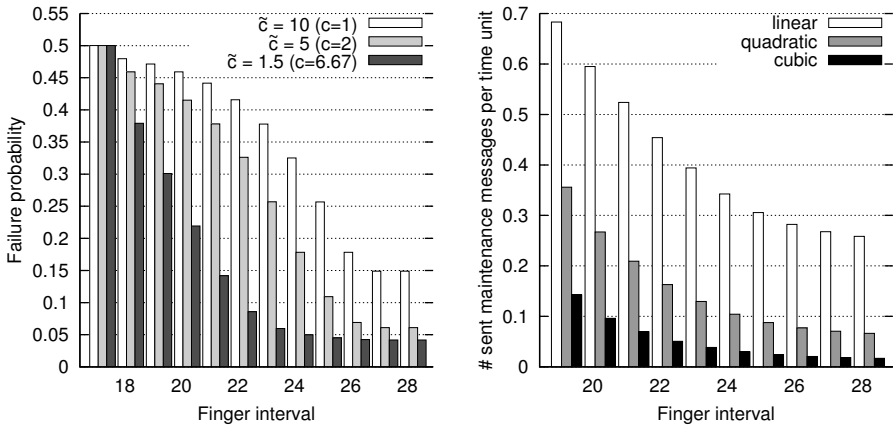
Recall that we assume two failure scenarios: nodes which fail according to a given probability distribution and nodes whose resources are depleted with when

sending and receiving nodes until node fail. For the first case, we now consider the failure probability of a random node  $x$ 's fingers given a fixed number of failures where the failure probabilities depend on nodes' resource levels. For the second case, we consider the expected number of maintenance messages for  $x$ 's fingers per unit of time.

*Random failures.* Let  $f_i = x.f[i].node$  for simplicity and assume that there are  $k_i = \lfloor N \cdot 2^{i-m} \rfloor$  nodes in  $B_{x,i}$  for  $i \geq \lceil \log(1/N) + m \rceil$ . Using the conditional failure probabilities for the nodes from (2) and the resource probabilities for the fingers from (4), we have:

$$\begin{aligned}
 P(f_i \text{ fails}) &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails and } (f_i)_R = j) \\
 &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails } | (f_i)_R = j) \cdot P((f_i)_R = j) \\
 &= \sum_{j=0}^{l_{max}} P(F_x | x_R = j) \cdot P(R_{k_i, min} = j).
 \end{aligned}$$

Again using the example distance distribution from Equation 6, Figure 3 shows an example scenario which demonstrates how the stretch affects fingers' failure probabilities. Note that for Chord and DHash++, this probability is constant for all fingers, resulting in fingers which are more likely to fail. Thus, we would



**Fig. 3.** Scenario with 100000 nodes, 32-bit keyspace,  $l_{max} = 3$ ,  $r = 10$ , and  $\mu = 2$ . The probability that a finger in a given interval fails is shown on the left for variable stretch  $\tilde{c}$  when half of the network nodes fail. The expected number of sent messages per finger per time unit for linear, quadratic, and cubic finger maintenance intervals is shown on the right, where the 19<sup>th</sup> finger is the first in which a node is expected.

expect fewer finger failures for the proposed protocol and thus a lower number of lookup failures.

*Resource depletion.* Restricting ourselves to maintenance messages only, we can use the finger maintenance interval function  $g$  and (4) to find the expected number of maintenance messages for a finger  $f_i$  per unit of time (see Figure 3):

$$E(\# \text{ messages for } f_i) = E\left(\frac{1}{g(R_{k_i, \min})}\right) = \frac{1}{\sum_{\ell=0}^{\ell_{\max}} g(\ell) \cdot P(R_{k_i, \min} = \ell)}.$$

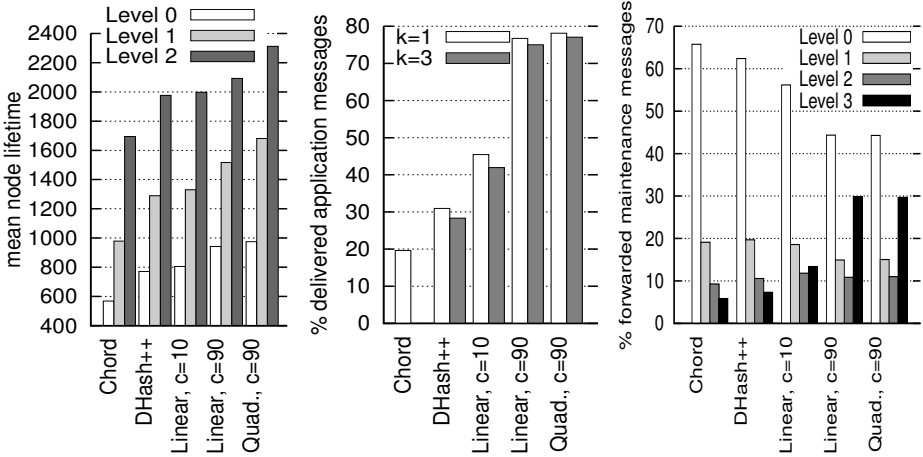
By summing up these expected number of maintenance messages over all of a node's fingers, we obtain the expected number of maintenance messages sent by a node during one unit of time. Note that in Figure 3, a constant finger maintenance interval  $g(\ell) = t_{ref}$  (i.e. as in Chord) would send one message per unit of time. Similar estimations can also be found for incoming maintenance messages. From Figure 3 we see that for nodes using quadratic finger maintenance intervals, each finger expectedly requires less than 40% of the maintenance messages necessary with a constant finger maintenance interval (and less than 15% for most fingers). Since maintenance overhead accounts for a large portion of the total network load, we expect that this reduced maintenance overhead would lead to a significant increase in the nodes' lifetimes.

## 6 Evaluation

We evaluated the benefits of RBFM's resource usage by focusing on the direct measures of node lifetimes and lookup failures as well as the indirect measures of the mean resource levels and physical distances used for routing and the total number of maintenance messages sent. The results support our assumption that better finger characteristics do indeed result in improvements in these global behaviors. We used the simulation environment OmNET++ with the overlay framework OverSim [13], which provides implemented overlays such as Chord and Kademlia and uses a coordinate sets with over 200000 coordinates. We extended the functionality of Chord to integrate nodes' resource levels and coordinates, maintain a prospective links list, and use an alternative selection of fingers using the prospective links list. We tested our system in two high-failure scenarios with four resource levels, 10000 nodes with quadratically Zipf-distributed resource availability ( $\mu = 2$ ), nodes' network coordinates provided by the OverSim framework, and a network coordinate diameter of approximately 2000.

In the first scenario, one-third of all nodes fail simultaneously according to a Zipf-distribution on their resource levels, as suggested in Section 3. The second scenario takes a simplistic approach to simulating dwindling battery power: nodes in the bottom three levels are assigned "resources" according to their resource levels - 100 for level 0, 200 for level 1, and 400 for level 2 - which are then decremented for each sent and received message. Top level nodes do not lose resources, due to their inexhaustible resources. A node's resource level is thus decreased with node activity until it is depleted and the node fails. This scenario





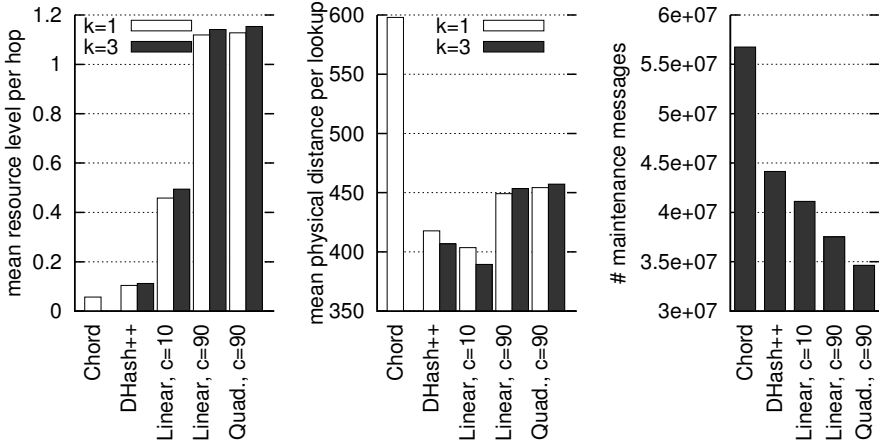
**Fig. 4.** The leftmost figure shows the mean node lifetimes for varying protocols and resource levels for  $k = 1$ . The middle figure shows the percentage of delivered application lookups using prospective links lists with one and three entries. The rightmost figure shows the percent of the total *forwarded* maintenance messages that each level forwarded for  $k = 1$  and quadratic finger maintenance intervals for RBFM.

is meant to demonstrate the differences between node lifetimes and message loss of the various protocols. All scenarios used a  $t_{ref} = 60$  second reference interval between finger maintenance, and RBFM used both linear and quadratic finger maintenance functions,  $g(x_R) = t_{ref} \cdot (x_R + 1)$  and  $g(x_R) = t_{ref} \cdot (x_R + 1)^2$ . Stretch and the number of prospective fingers  $k$  were also varied.

A dummy application on each node generated a lookup to a random key every 60 seconds, which served to test the deliverability of messages to random nodes. For both scenarios, the differences between the protocols’ average hop counts for these lookups were nearly negligible. The average hop count varied by at the most 5% with 6.5 to 6.8 hops, and the standard deviation of hops per lookup was, at its highest, 1.8.

Due to space limitations, we do not further discuss the results for the scenario in which one-third of all nodes simultaneously fail, but one result is noteworthy. Message failures were comparable at around 20 – 25% for Chord, DHash++, and RBFM with a linear finger maintenance interval for stretch  $c = 10, 90$ , but surprisingly poor (45 – 55%) for the quadratic finger maintenance function for  $c = 90$ . This indicates the importance of adequately frequent finger maintenance for all levels when large scale failures can be expected.

In the second scenario, each node reduced its resources by 0.05 for each sent and received message. New nodes were added once nodes had failed to ensure that there were a total of 10000 nodes with a quadratic Zipf-distribution of their resource levels. Thus, a total of between 34000 and 59000 nodes (depending on the protocol) were introduced to the network over the measurement period of 4000 seconds. All results are based on the means of three runs which yielded



**Fig. 5.** The left figure demonstrates that the mean resource level per hop with a suitable configuration of RBFM is nearly twenty times that of Chord and ten times that of DHash++. The middle figure shows the mean physical distance traveled by a lookup, the right figure the total number of maintenance messages sent (and forwarded).

negligible variances. The most direct results for node robustness are the mean node lifetimes for various levels and the application lookup deliverability, as shown in Figure 4.

Both results were slightly surprising, in that the resource-naive DHash++ performed significantly better with respect to resources than Chord although we expected both to behave similarly. This improvement can be rationalized by DHash++’s prospective links list. On the one hand, finger maintenance is often routed directly to an existing finger in the prospective links list, generating less maintenance overhead and thus less resource usage (as also seen in Figure 5). On the other hand, nodes with higher resource levels also have longer lifetimes, making them more well known to other nodes by their continually backpacked information. Since this information can propagate longer, these higher level nodes will invariably take up a larger portion of the prospective fingers lists and thus other nodes’ fingers, giving it an indirect resource-awareness which improves lookup deliverability.

The remaining results reflect our analysis, with lower lookup failure and higher node lifetimes for RBFM configurations. Note that the comparative values for the lower resource levels are our primary interest, since the highest number of nodes belong to these volatile levels. As expected, the highest mean node lifetime is achieved by the RBFM configuration which sends the most infrequent of the tested maintenance messages: the quadratic finger maintenance interval. And contrary to the first failure scenario, this configuration is clearly adequate for this scenario’s constant churn as it has the highest message delivery rate. Figure 4 indicates that the percentage of hops routed over lower levels is significantly reduced for RBFM, depending on the stretch. As expected from the analysis,

a substantial portion of forwarding responsibilities have been transferred from lower level to higher level nodes using RBFM. Figure 5 further demonstrates the RBFM reduction in maintenance messages, with a decreasing tendency for increasing stretch (i.e. increasing resource-awareness integrated into the resource distance) and infrequent finger maintenance interval.

The mean resource level per hop and mean physical distance traveled of all application lookups are shown in Figure 5. Note the increase in mean resource level of DHash++ compared to Chord, as mentioned above. While both measures appear to depend on the stretch, as opposed to the finger maintenance interval, RBFM with a linear finger maintenance interval and low stretch actually has a lower mean physical distance traveled than DHash++. This is surprising, since we consider resource awareness and location awareness to be trade offs, and provides motivation for further study. On the other hand, the increase in mean resource levels per hop in Figure 5 surpassed our expectations from the analysis. This increase is due to a combination of the higher resource fingers as in Figure 2 and the fact that higher fingers are used more frequently in routing.

## 7 Future Work

Using analytical examinations, we have shown that our proposed protocol builds more robust links and significantly reduces the maintenance and routing load on weak nodes. In addition to confirming these findings, simulation has shown that, depending on the scenario, our protocol offers better routing robustness while reducing the overall maintenance overhead and shifting the remaining overhead from weak to strong nodes. This furthermore increases the mean node lifetime in a battery-powered network scenario. RBFM provides an opportunity to allocate storage and query load to low resource nodes while preventing delays and protecting nodes from failure due to maintenance and routing load. The protocol developed here is essentially an extension of Chord and could be considered for other similar DHTs. Future work should consider data replication within this extension and the feasibility of developing a hierarchical DHT to obtain similar resource and location awareness.

## References

1. Artigas, M.S., Lopez, P.G., Skarmeta, A.F.: A comparative study of hierarchical dht systems. In: Proceedings of the 32nd IEEE Conference on Local Computer Networks, pp. 325–333 (2007)
2. Artigas, M., Lopez, P., Ahullo, J., Skarmeta, A.: Cyclone: A novel design schema for hierarchical dhts. In: IEEE P2P 2005, pp. 49–56 (2005)
3. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-attribute range queries. In: SIGCOMM 2004, pp. 353–366 (2004)
4. Bustamante, F., Qiao, Y.: Friendships that last: Peer lifespan and its role in p2p protocols. In: Douglis, F., Davison, B. (eds.) Web Content Caching and Distribution, pp. 233–246. Springer, Netherlands (2004)
5. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: SIGCOMM 2004, pp. 15–26 (2004)

6. Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a dht for low latency and high throughput. In: Proceedings of the 1st NSDI, pp. 85–98 (2004)
7. El Dick, M., Pacitti, E., Kemme, B.: Flower-cdn: A hybrid p2p overlay for efficient query processing in cdn. In: EDBT 2009, pp. 427–438 (2009)
8. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB 2004, pp. 444–455 (2004)
9. Ganesan, P., Gummadi, K., Garcia-Molina, H.: Canon in g major: Designing dhds with hierarchical structure. In: ICDCS 2004, pp. 263–272 (2004)
10. Garcés-Erice, L., Biersack, E.W., Felber, P., Ross, K.W., Urvoy-Keller, G.: Hierarchical Peer-to-Peer Systems. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1230–1239. Springer, Heidelberg (2003)
11. Godfrey, P.B., Stoica, I.: Heterogeneity and load balance in distributed hash tables. In: IEEE INFOCOM, pp. 596–606 (2005)
12. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of dht routing geometry on resilience and proximity. In: SIGCOMM 2003, pp. 381–394 (2003)
13. Baumgart, I., Heep, B., Krause, S.: Oversim: A scalable and flexible overlay framework for simulation and real network applications. In: IEEE P2P (2009), <http://www.oversim.org/wiki>
14. Karger, D.R., Lehman, E., Leighton, F.T., Panigrahy, R., Levine, M.S., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: STOC, pp. 654–663 (1997)
15. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: SPAA 2004, pp. 36–43 (2004)
16. Maniymaran, B., Bertier, M., Kermarrec, A.-M.: Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks. In: ICDCS 2007, pp. 33–33 (June 2007)
17. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the Xor Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
18. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: SIGCOMM 2001 (2001)
19. Ren, S., Guo, L., Jiang, S., Zhang, X.: Sat-match: A self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks. In: IPDPS 2004, pp. 83–91 (April 2004)
20. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of the Multimedia Computing and Networking (2002)
21. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM 2001, pp. 149–160 (2001)
22. Waldvogel, M., Rinaldi, R.: Efficient topology-aware overlay network. SIGCOMM Comput. Commun. Rev. 33, 101–106 (2003)
23. Xu, Z., Min, R., Hu, Y.: Hieras: A dht based hierarchical p2p routing algorithm. In: ICCP 2003, pp. 187–194 (2003)
24. Zhao, B.Y., Kubiatiowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley (2001)
25. Zoels, S., Despotovic, Z., Kellerer, W.: Cost-based analysis of hierarchical dht design. In: IEEE P2P 2006, pp. 233–239 (2006)

# SQL Streaming Process in Query Engine Net

Qiming Chen and Meichun Hsu

HP Labs,  
Palo Alto, California, USA  
Hewlett Packard Co.  
{qiming.chen,meichun.hsu}@hp.com

**Abstract.** The massively growing data volume and the pressing need for low latency are pushing the traditional store-first-query-later data warehousing technologies beyond their limits. Many enterprise applications are now based on continuous analytics of data streams. While integrating stream processing with query processing takes advantage of SQL's expressive power and DBMS's data management capability, it raises serious challenges in dealing with complex dataflow, applying queries to unbounded stream data, and providing highly scalable, dynamically configurable, elastic infrastructure.

To solve these problems, we model the general graph-structured, continuous dataflow analytics as a SQL Streaming Process with multiple connected and *stationed* continuous queries; then we extend the query engine to support cycle-based query execution for processing unbounded stream data chunk-wise with sound semantics; and finally, we develop the Query Engine Net (QE-Net) over the Distributed Caching Platforms (DCP) as a dynamically configurable elastic infrastructure for parallel and distributed execution of SQL Streaming Processes.

We extended the PostgreSQL engines for building the QE-Net infrastructure. Our experience shows its merit in leveraging SQL and query processing to analyze real-time, graph-structured and unbounded streams. Integrating it with a commercial and proprietary MPP based database cluster is being investigated.

## 1 Introduction

Due to the massively increasing data volumes and demands for low latency, many enterprise applications are based on the continuous analytics of data streams which can run orders of magnitude more efficiently than the traditional store-first-query-later technologies [6]. Executing data intensive stream analysis by query engines allows us to take advantage of the expressive power of SQL, the streaming functionality of query processing, and in general, the database innovations made in decades. This goal is very different from building a stream processing system from scratch, with the mature DBMS technology left behind and then re-invented.

In order to support complex data streaming applications by SQL and query engines, the following problems need to be solved.

- A single SQL query represents tree-structured operations; the query result cannot be routed to more than one destination, and in the pipelined fashion. As a result, a single SQL query has limited dataflow expressive power at the process level.
- A SQL query, such as one with aggregation, may not be definable on the unbounded stream data. Essentially, an infinite data stream can only be analyzed in granules, which requires us to apply a SQL query to the incoming data chunk by chunk falling in consecutive windows (conceptually a chunk can be as small as a single tuple). Given that it is also required to trace the application states continuously across chunk boundaries for supporting sliding window based, history sensitive operations. Meeting these two requirements is challenging since they are conflict wrt the existing query processing techniques.
- Proving a service grid is the key to support parallel and distributed stream analytics, which raises the challenges in continuously running, connecting and coordinating multiple queries on multiple query engines. Further, it is necessary to facilitate a flexible and dynamically configurable infrastructure, compared with the statically configured Map-Reduce (M-R) system [5] for dealing with pre-partitioned, bounded data on disks.

In this project we tackle these problems in three dimensions:

- Graph-structured SQL Streaming Process;
- Granule-based stream analytics;
- Query Engine Net (QE-Net).

We model graph-structured, continuous dataflow analytics by SQL Streaming Process with multiple *stationed*, long-standing Continuous Queries (CQs). These CQs are executed by distributed query engines and connected through multi-nodes memory sharing.

Since the semantics of a query execution is only definable on a bounded data set, an unbounded data stream must be processed in granule; we developed the cycle based CQ model to allow the query to be executed cycle by cycle for processing the unbounded stream data chunk by chunk, with each execution cycle applied to a bounded chunk of data.

A QE-Net, as an *elastic stream analytics infrastructure*, is made of multiple query engines on a server cluster with Infiniband-based high-speed interconnection. These query engines serve as the “*executors*” of SQL query based dataflow operations; they are *dynamically* configured for higher flexibility and availability, compared with the *statically* configured Map-Reduce platform. SQL is the common language across QE-Net. For streaming analytics, the queries are *stationed* CQs with data-driven executions, and synchronized by the common data chunking criterion. The query results are exchanged through write/read a unified share-memory across multiple server nodes which is supported by the recently-popular Distributed Caching Platform (DCP) [1,11].

This novel platform is built by integrating and extending several technologies we developed at HP Labs in query process [3], dataflow language [4] and stream processing [3,5]. While staying in the SQL world we also take advantage of the NoSQL mechanisms, such as M-R and DCP, for enhanced scalability and availability.

The rest of this paper is organized as follows: Section 2 introduces SQL Streaming Process; Section 3 describes how to handle unbounded stream data granularly; Section 4 overviews the QE-Net infrastructure. Section 5 concludes the paper.

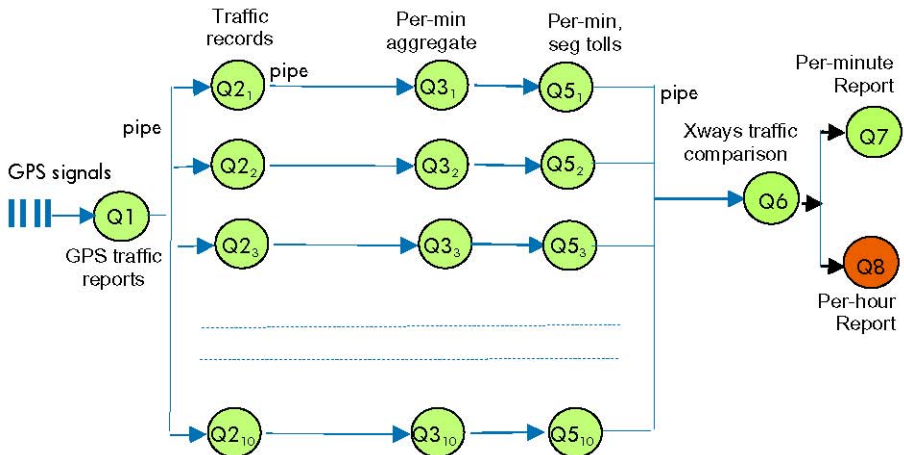
## 2 Graph-Structured SQL Streaming Process

Since complex streaming applications are often expressed as graph-structured dataflows, we introduce the notion of *SQL Stream Process*. A SQL Streaming Process represents a continuous dataflow application at the process level by one or more correlated CQs, which form sequential, concurrent or nested steps. A query may invoke User Defined Functions (UDFs) including relation-in, relation-out UDFs, referred to as Relation-Valued Functions (RVFs) as we previously explored [4]. The result-set of a query at a step, becomes the data source of the successor queries.

We use a simplified as well as extended Linear-Road (LR) benchmark [9] to demonstrate our framework. The LR benchmark models the traffic on 10 express ways; each express way has two directions and 100 segments. Cars may enter and exit any segment. The position of each car is read every 30 seconds and each reading constitutes an event, or stream element, for the system. A car position report has attributes *vehicle\_id*, *time* (in seconds), *speed* (mph), *xway* (express way), *dir* (direction), *seg* (segment), etc. With the simplified benchmark, the traffic statistics for each highway segment, i.e. the number of active cars, their average speed per minute, and the past 5-minute moving average of vehicle speed, are computed. Based on these per-minute per-segment statistics, the application computes the tolls to be charged to a vehicle entering a segment any time during the next minute. As an extension to the LR application, the traffic statuses of the 10 express ways are compared and then reported every minute and every hour.

A simplified SQL Streaming Process example for the above traffic analysis is given in Fig. 1 with the traffic records following the schema below, as the source data.

GPS\_car\_event [ *vehicle\_id*, *time*, *xway*, *dir*, *seg*, *speed*, ... ]



**Fig. 1.** A time-window based snapshot of the SQL Streaming Process for LR traffic analysis where queries are pipeline cascaded by “pipes” (queues); the semantics of this snapshot is defined on a bounded chunk of stream elements

The process contains the following queries.

- Query  $Q_1$  captures GPS events from cars with the timestamps in seconds, and converts them to minute based tuples; these events bear the positions and speeds of cars;
- $Q_2, \dots, Q_{2_{10}}$  capture the data partitioned by express way, each partition containing the GPS events of cars on all the segments of an express way;
- $Q_3, \dots, Q_{3_{10}}$  apply to the data partitioned by express way, with each aggregating the per-minute traffic volume and average speed on every segment of an express way;
- $Q_5, \dots, Q_{5_{10}}$  each computes the per-minute, per-segment tolls of an express way based on the above information; the tolls are applied to the next minute;
- $Q_6$  analyzes and compares the traffic status of those express ways based on the above traffic statuses and tolls;
- $Q_7$  and  $Q_8$  generate two kinds of traffic analysis reports, one on the minute basis and the other on the hourly basis.

The pseudo specification of this SQL Streaming Process is illustrated below where the UDFs involved in queries are registered with the query engine in the regular way which is not shown here. For simplicity we represent a *query variable*, e.g. the result of  $Q_i$  using the corresponding query name with prefix \$, e.g.  $\$Q_i$ .

```

Create SQL Streaming Process LR_Traffic_Analysis {
  Source { STREAM GPS_Event_Stream };

  Q1 := SELECT vehicle_id, FLOOR(time/60)::INTEGER AS minute, xway, dir, seg, speed
        FROM GPS_Event_Stream
        [GRANULE: 60 seconds]
  Q2 := SELECT q1.vehicle_id, q1.minute, q1.xway, q1.dir, q1.seg, q1.speed
        FROM $Q1 q1 WHERE q1.xway = 1
        [GRANULE: 1 minute, MAP HASH PARTITION KEY: vehicle_id]
  ....
  Q3 := SELECT minute, xway, dir, seg,
        AVG(speed) AS avg_speed, COUNT(DISTINCT vehicle_id) AS cars_volume
        FROM $Q2 GROUP BY minute, xway, dir, seg /* for unique minute and xway */
        [GRANULE: 1 minute, REDUCE GROUP KEY: dir, seg ]
  ....
  Q5 := SELECT minute, xway, dir, seg, avg_speed, cars_volume,
        Calc_Toll (avg_speed, cars_volume) FROM $Q3;
  ....
  Q6 := SELECT * FROM Analyze ($Q5, $Q5, ..., $Q5_{10})
        [INPUTMODE: BLOCK, GRANULE: 1 minute ]
  Q7 := SELECT * FROM Report_by_Minute ($Q6)
        [INPUTMODE: BLOCK, GRANULE: 1 minute ]
  Q8 := SELECT * FROM Report_by_Hour ($Q6)
        [INPUTMODE: BLOCK, GRANULE: 60 minutes ]
}

```

In fact, the queries  $Q_2$ ,  $Q_3$ ,  $Q_5$  for an express way are specified as a parameterized sub-process, for simplicity we do not go through the detailed syntax here.

Let us first consider a snapshot of this dataflow process on one chunk of the stream data, e.g. the events falling in the one-minute time boundary; in this case the queries



behave like one-time queries. Then this dataflow process has definable semantics on the bounded input data, so are the involved queries. The aggregation oriented queries such as  $Q_3$ ,  $Q_8$ , can be completed on such a bounded chunk of data (otherwise if the input data are infinite the aggregation is undefinable). This indicates that to apply a CQ or a dataflow process on infinite stream data, it is necessary to punctuated data into chunks as specified by the GRANULE property.

Now let us assume that the input stream data are infinite. In this case, all the queries are CQs running continuously. Apply a CQ to a data stream returns a data stream as well; therefore the “query variables” are themselves streams. However, when apply a CQ to the input stream chunk by chunk, the output stream represents the sequence of chunk-wise data processing results, which we referred to as “granular stream processing” and will be described in more detail later.

The functions used in  $Q_6$ ,  $Q_7$ ,  $Q_8$  are RVFs we discussed in [4] which take relations (query results) as input; however, as GRANULE is specified, these functions take chunks of the corresponding relations as their input.

A dataflow process describes the operations orchestrated along the data flow paths. A streaming process is a dataflow process with unbounded input data streams. In a SQL Streaming Process described above, the component queries are CQs with long-standing query instances.

**Capture Stream Data by CQ.** To fuel queries continuously with unbounded data, we replace the database table, which contains a set of tuples on disk, by the special kind of table function, called *Stream Source Function* (SSF) that returns a sequence of tuples to feed a query without first storing on disk. Accordingly, the query engine's *table-scan* access method is replaced by a special kind of *function-scan*. A SSF can listen or read data/events sequence and generate stream elements tuple by tuple continuously.

**Connect “Stationed” CQs by “Pipes”.** Based on our approach, a CQ is executed as a long-standing query instance running continuously, rather than as multiple one-time query instances. To describe the execution environment of CQs, we introduce the notion of *station* for hosting a query, and the notion of *pipe* as the FIFO stream container for connecting stations. At a minimum, a station is specified with a name, the hosted CQ, the outgoing pipes, and the query engine for executing the CQ; a pipe is defined with an ID, a *relation schema* for type-preservation, an origin and a destination. A pipe is an abstract object that can be instantiated to a *queue* or a *stream table* (in memory or on disk). The results of a query may be replicated to multiple pipes for multiple destination stations.

### 3 Analyzing Unbounded Stream Data Granularly

#### 3.1 Chunk-Wise Stream Processing by Cycle-Based Continuous Queries

The difficulty of using regular SQL queries for stream processing is that a SQL query is not definable on unbounded data since it cannot return complete result, and if the query involves aggregation, it never returns any result. Our solution is to *cut* the input stream data into a sequence of chunks with each chunk representing a bounded data set on that a query is definable, and after processing a chunk of data, to rewind the query instance for processing the next chunk of data.

We introduce the notion of Cycle-based CQ (CCQ) to characterize such queries. In general, the execution of CCQ on an infinite stream is made in a sequence of *cycles*, one on each stream chunks. To allow this query to apply to the stream data one chunk at a time, and to return a sequence of chunk-wise query results, the input stream. To support cycle based query execution for chunk-wise data processing, we developed the *cut-and-rewind* query execution mechanism, namely, cut a query execution based on the cycle specification and then rewind the state of the query without shutting it down, for processing the next chunk of stream data in the next cycle.

*Cut* is originated in the SSF at the bottom of the query tree. SSFs have a general form of *STREAM(SS, cycle-spec)* specifies that the stream source *SS* is to be “cut” into an unbounded sequence of *chunks*  $SS_{C_0}, SS_{C_1}, \dots$ , where all tuples in  $SS_{C_i}$  occur before any tuple in  $SS_{C_{i+1}}$  in the stream. The “cut point” is specified in the *cycle-spec*. Upon detection of end-of-cycle condition, the SSF signals *end-of-cycle* to the query engine resulting in the termination of the current query execution cycle. In general the end of a cycle is determined when the first stream element belonging to the next cycle is received; that element will be cached to be processed first in the next cycle.

Upon termination of an execution cycle, the query engine does not shut down the query instance but *rewinds* it for processing the next chunk of stream data. Rewinding a query is a top-down process along the query plan instance tree, with specific treatment on each node type. In general, the intermediate results of the standard SQL operators (associated with the current chunk of data) are discarded but the application context kept in UDFs (e.g. for handling sliding windows) is retained. Since the query instance remains alive across cycles, data for sliding-window oriented, history sensitive operations can be kept continuously. Bringing these two capabilities together is the key in our approach.

### 3.2 Granule-Based Execution of Continuous SQL Stream Process

In a SQL Streaming Process, every query is a *stationed* CQ that uses a SSF to get the stream data from its successors. Under the cut-and-rewind mechanism, a CQ is running cycle by cycle and therefore referred to as Cycle-based CQ (CCQ).

- The default granule is one tuple; a CQ with the default granule and without aggregation, should not be cut and rewound.
- However, as a rule any query on infinite stream that is defined on punctuated input or involves aggregation, such as  $Q_3, Q_8$ ... above, must be defined as a CCQ, which is indicated by the GRANULE property.
- A CCQ continuously generates an unbounded sequence of query results, one on each *chunk* of the stream data.
- The *paces of dataflow* wrt timestamps are different at different operators (queries). The input tuples to  $Q_1$  is time-stamped by second, to others is by minute, and after  $Q_8$  is by hour.

A query, including a CCQ, may have PER\_TUPLE and BLOCK input modes, with the default mode as PER\_TUPLE. A query, e.g.  $Q_8$ , in the BLOCK mode means that it cannot generate results until reading in the whole relation or chunk (if it is a CCQ), which is orthogonal to chunking.

### 3.3 Data Parallel Execution of Sequential Operators

We support scalability at two levels – process level and operation level, by two partitioning mechanisms. *Process partitioning* is to instantiate multiple instances of a SQL streaming process or sub-process and deploy them on different engine sets for parallel execution. *Query partitioning* is to support the data-parallel execution of one or more queries connected sequentially. In the above GPS events based traffic analysis example, for a single express way the stream data are hash partitioned by *vehicle\_id* across the nodes for running  $Q_2$ , and  $Q_2$  and  $Q_3$  are paired for M-R computation as illustrated in Fig. 2. This data-parallel execution is planned and launched by the system based on the process specification.

Back to the above GPS events based traffic analysis example, the *Map* queries generate local aggregates; the *Reduce* queries fuse the local aggregation results. The *Map* queries are cut and rewind every 60 seconds (1 minute), with each running cycle by cycle for providing minute based partial aggregates. The *Map* results are shuffled to the *Reduce* sites after each execution cycle based on the *network replicated hash-tables*. The *Reduce* query is again equipped with a SSF for receiving the *Map* results. The *Map* results provide timestamps for the *Reduce* operation to be synchronized. Both *Map* and *Reduce* queries run in the per-minute cycle.

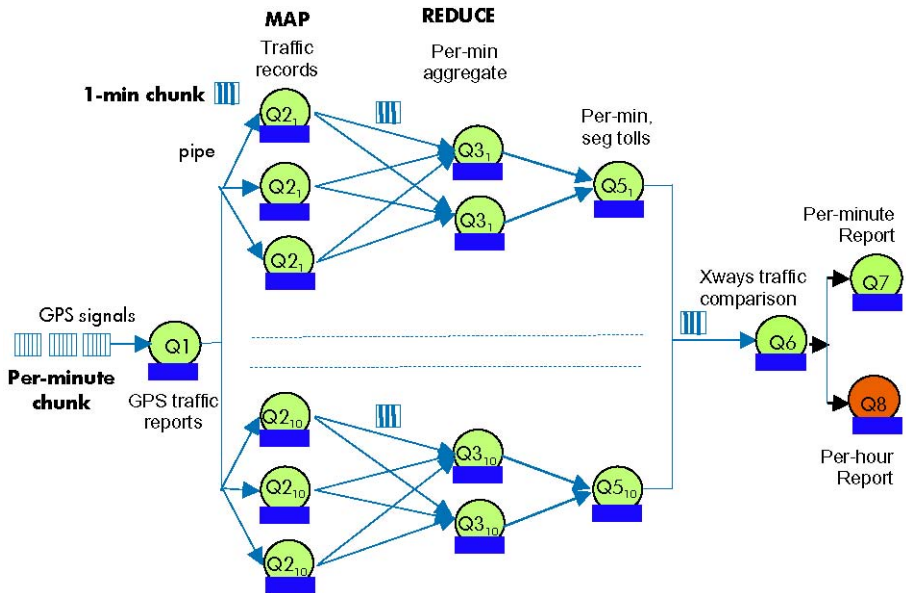


Fig. 2. The SQL Streaming Process is planned, parallelized for data-parallel execution of its sequential building blocks based on the Map-Reduce computation model

## 4 Query Engine Net for Executing SQL Stream Process

We execute a SQL Streaming Process by the Query Engine Net (QE-Net) running on multiple server nodes. The QE-Net is a distributed, scalable, pluggable infrastructure underling the dataflow programming paradigm to develop cloud applications for processing continuous unbounded streams of data. On this goal we share the spirit of Microsoft Orleans [2], Yahoo! S4 [10], Microsoft Dryad [8], and IBM Spade [7]. However, different from these systems, the common language of our infrastructure is SQL, and the basic executors are query engines which allow us to compose various SQL compliant engines and to take advantage of SQL's expressive power and query processing technology for stream analytics.

A QE-Net is made of multiple query engines (query nodes) on a server cluster with Infiniband-based high-speed interconnection, with the following major characteristics.

- The QE-Net is a grid of *analysis engines* serving as “*executors*” of SQL-based dataflow operations. The primary function of the QE-Net is to execute graph-based data streaming, rather than to offer distributed data stores.
- The query engines are *dynamically* configured for executing a SQL Streaming Process, which, compared with the *statically* configured Map-Reduce platform, offers enhanced flexibility and availability. Note that a query engine is able to execute multiple CQs belong to different processes, and therefore can be used in the execution of multiple processes.
- The common language across QE-Net is SQL which makes it homogeneous at the streaming process level; the servers can be heterogeneous as far as they run the query engines with the required capability.
- For streaming analytics, the queries are *stationed* CQs with execution driven by infinite stream data; they synchronize by the understood data granule criterion.

To connect multiple queries efficiently in a graph-structured analytic SQL dataflow process, providing a memory sharing mechanism for caching the source and destination data of these queries is a reasonable choice, and since the dataflow process is executed by the query engines running on multiple server nodes, a unified view to the memory on distributed servers is the key. To provide such “everyone talks to sharable data cache” programming paradigm, we adopt Distributed Caching Platform (DCP) [1], such as Memcached [11].

## 5 Conclusions

In this project we tackle these problems in three dimensions. First, we model graph-structured streaming analytics as SQL Streaming Process composed with multiple connected continuous queries. Next, we extend the query engine for dealing with infinite stream data granularly. Finally, we integrate the Query Engine Net (QE-Net) and the Distributed Caching Platform (DCP) as a highly scalable and *elastic infrastructure* for the parallel and distributed execution of SQL Streaming Processes.

This novel platform is prototyped, using PostgreSQL engines, by integrating and extending several technologies we developed at HP Labs. Integrating it with a commercial MPP based analytic database cluster, is being investigated.

## References

1. Nori, A.: Distributed Caching Platforms. In: VLDB (2010)
2. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. VLDB Journal 2(15) (June 2006)
3. Chen, Q., Hsu, M., Zeller, H.: Experience in Continuous analytics as a Service (CaaS). In: EDBT (2011)
4. Chen, Q., Hsu, M.: Continuous MapReduce for In-DB Stream Analytics. In: Proc. CoopIS (2010)
5. Dean, J.: Experiences with MapReduce, an abstraction for large-scale computation. In: Int. Conf. on Parallel Architecture and Compilation Techniques. ACM (2006)
6. Franklin, M.J., et al.: Continuous Analytics: Rethinking Query Processing in a Network Effect World. In: CIDR (2009)
7. Gedik, B., Andrade, H., Wu, K.-L., Yu, P.S., Doo, M.C.: SPADE: The System S Declarative Stream Processing Engine. In: ACM SIGMOD (2008)
8. Isard, M., Budi, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: Distributed data-parallel programs from sequential building blocks. In: EuroSys 2007 (March 2007)
9. Jain, N., et al.: Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. In: SIGMOD (2006)
10. Neumeyer, L., Bruce, R., Anish, N., Anand, K.: S4: Distributed Stream Computing Platform. In: KDCLOUD 2010, Sydney, Australia (December 2010)
11. Memcached (2010), <http://www.memcached.org/>

# Instance-Based ‘One-to-Some’ Assignment of Similarity Measures to Attributes

## (Short Paper)

Tobias Vogel and Felix Naumann

Hasso Plattner Institute, University of Potsdam, Germany  
{firstname.lastname}@hpi.uni-potsdam.de

**Abstract.** Data quality is a key factor for economical success. It is usually defined as a set of properties of data, such as completeness, accessibility, relevance, and conciseness. The latter includes the absence of multiple representations for same real world objects. To avoid such duplicates, there is a wide range of commercial products and customized self-coded software. These programs can be quite expensive both in acquisition and maintenance. In particular, small and medium-sized companies cannot afford these tools. Moreover, it is difficult to set up and tune all necessary parameters in these programs. Recently, web-based applications for duplicate detection have emerged. However, they are not easy to integrate into the local IT landscape and require much manual configuration effort.

With DAQS (Data Quality as a Service) we present a novel approach to support duplicate detection. The approach features (1) minimal required user interaction and (2) self-configuration for the provided input data. To this end, each data cleansing task is classified to find out which metadata is available. Next, similarity measures are automatically assigned to the provided records’ attributes and a duplicate detection process is carried out. In this paper we introduce a novel matching approach, called one-to-some or 1:k assignment, to assign similarity measures to attributes. We performed an extensive evaluation on a large training corpus and ten test datasets of address data and achieved promising results.

**Keywords:** Database Management, Database Applications, Data mining, Matching, Intrinsic, Data Quality, matching, duplicate detection, similarity measures, data cleansing.

## 1 The Need of Data Quality

The process of searching through a database and looking for pairs of records that have a high similarity and thereby identifying multiple representations of same real-world objects is called *duplicate detection*. To perform efficient duplicate detection, the challenge is to develop good similarity measures and to apply them on the corresponding attributes in the data to be cleaned. The mapping of similarity measures to attributes is an assignment problem.

Traditionally, this assignment is created manually, which is time-consuming and tedious. The degree of automation in the duplicate detection process increases when those assignments are generated automatically.

Another benefit of the service paradigm comes with the multi-tenancy: the service can learn and improve the assignment quality when being exposed to many different datasets better than it could do in a single client scenario.

Data that are to be de-duplicated come in various guises. For example, if the data is not in relational format, it is unclear *which* attributes to compare. If the datatype or semantics are unknown, it is hard to decide *how* to compare different attributes. We address these challenges towards a service-based duplicate detection technique without human interaction, which is integrated into the DAQS (**D**ata **Q**uality as a **S**ervice) project. In this paper, we propose

- A classification of different duplicate detection tasks based on the amount of available meta data
- A technique to classify attributes according to their semantics in the schema
- The novel 1:k assignment problem between attributes and similarity measures and an extended version of the Hungarian Algorithm to solve it
- A comprehensive evaluation on ten datasets showing the feasibility and effectiveness of our approach

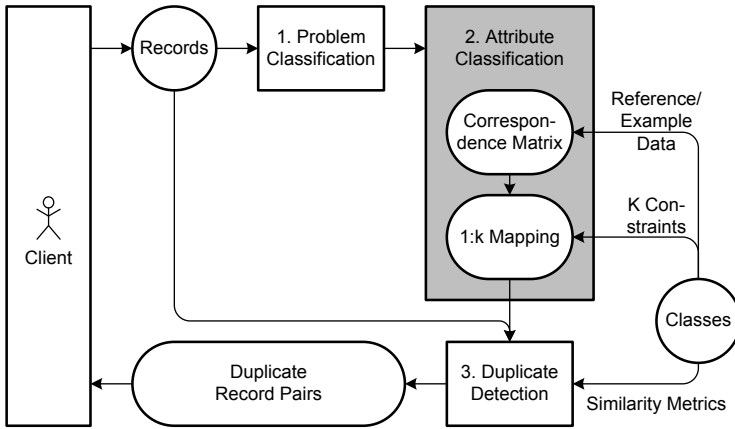
The envisioned duplicate detection service works in three phases, as illustrated in Fig. 1. Each phase is described in detail in the following sections. In the *problem classification phase* (Sec. 2) the provided records are analyzed to determine their format and how to treat them during further processing, e. g., whether information retrieval techniques have to be applied, whether schema information are present, etc. In the subsequent *attribute classification phase* for each attribute a corresponding similarity measure is found automatically using the 1:k assignment technique. This phase is explained further in Sec. 3 and is the main focus of this paper. Section 4 presents an evaluation of the assignment results. The third *duplicate detection phase* executes the duplicate detection logic, i. e., the algorithm that decides against which pairs of records to apply the similarity measures. It is not the main focus of this paper and is thus described in Sec. 5 together with related work. Finally, Sec. 6 summarizes the approach and proposes future work.

## 2 Problem Classification

Duplicate detection relies on knowledge about what the attributes are, which attributes to compare and how to compare the attributes’ values. Consequently, result quality degrades the less information is available.

Figure 2 shows a severity-classification of the problem, depending on the type of available metadata. Optimal conditions ((1) in Fig. 2) are present, if the semantics of the attributes are clear. In particular, we know not only the primitive datatype (String, Integer, etc.) but the concrete semantics (name, phone, date-of-birth, etc.). The comparison function as well as the mapping between the tuple’s attributes can be derived from them. Consequently, the duplicate detection run can be performed nearly automatically.

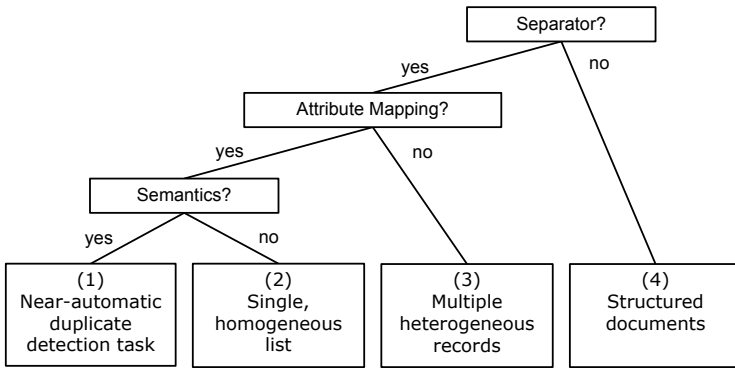
In class (2), semantic classes have to be assigned by a human expert, which leads to the appropriate comparison functions. For instance, compare names using the Jaccard similarity measure.



**Fig. 1.** Workflow and architecture of the duplicate detection service

In class (3), a mapping between the values of different records has to be first established. Many methods are shown by Euzenat and Shvaiko [2] that use data types, attribute names, or values.

In class (4), the elements are structured, but the structure is unknown. To separate the elements’ attributes from each other, information retrieval techniques have to be used.



**Fig. 2.** Four different classes of duplicate detection are illustrated in a tree

With our approach of automatically determining suitable similarity measures we can transform Class 2 problems to Class 1 problems i. e., we assume having attribute mappings and separators.



### 3 Attribute Classification

The goal of attribute classification is to assign appropriate, highly specialized similarity measures to the attributes of the input data based on their semantics: Two instances of a *given name* should be compared differently than two *email addresses*.

The process of creating such assignment between attributes and classes (i. e., semantics) is called *classification*. Similarity measures are directly derived from those classes. Note, that two different classes may imply the same similarity measure, e. g., *state* and *country*. An assignment consists of *correspondences* between attributes and classes.

Classification is performed in two phases. First, for each column, the dataset’s values are compared against a set of reference data for different classes. Second, another classification is performed with the help of feature vectors, based on *example data*. Each of the phases returns a set of possible correspondences represented as a correspondence matrix.

Note that this approach is mostly language-independent. We do not know the language or country used in the dataset. However, we provide a large variety of classes with different abstraction, e. g., German street names vs. general street names as well as multi-lingual reference and training sets. We trust the algorithm to find the most appropriate class and thereby identify the language. Furthermore, for some classes, language is irrelevant, such as telephone numbers or email addresses.

*Phase 1: Dictionary-based Classification.* To calculate the similarity between an attribute of the input data and a class, we determine the ratio of the attribute’s values  $T$  that also appear in the class’ reference dataset  $R$  divided by the total number of (non-NULL) attribute values  $T$ :

$$\frac{|T \times R|}{|T|}$$

Ratios below a certain class-specific threshold (0.8 was a good threshold for most of the classes) are set to zero. We derive a preliminary assignment from this matrix with a maximum bipartite graph matcher, described in detail in Sec. 3.

*Phase 2: Machine Learning Classification.* Not all classes can be identified using a (finite) reference dataset, e. g., *telephone number* or *family name*. Therefore, we use example values available for those classes to learn the particulars of those classes for classification.

We define a set of boolean features such as single and multiple character, 2-gram and lookup features, which are applied to each single attribute value, thus creating *feature vectors*. We use the common heuristic that there is a high probability that feature vectors for values from the same attribute/domain are similar.

Once input and example data are represented by feature vectors, the input data can be classified based on the example data. We use the Naïve Bayes classifier of Weka [4] and classify each attribute value separately.

*Correspondence Matrix.* Both the dictionary-based and the machine learning classification result in a *correspondence matrix*, each. A correspondence matrix contains  $n$  attributes and  $m$  classes, with  $n \geq m$  without loss of generality. The matrix contains all

possible correspondences between attributes and classes and describes the probability of each correspondence to appear in the final assignment.

Based on such a matrix, it is not trivial to determine which attribute to assign to which class. Conceptually, the problem corresponds to the weighted bipartite matching problem: Given the correspondence matrix with assignment weights, assign to each attribute a class such that no class participates in more than one assignment and the sum of weights is maximized. Domain knowledge allows us to restrict the number of matches to certain classes. For instance, we might want to encode that a person has no more than two given names but gender can only appear once. Thus, we redefine the matching problem in Sec. 3.

To incorporate such restrictions in the matching problem, we propose the *1:k assignment* which is basically a 1:n assignment, but with varying  $n$  for each class.

*1:k Assignment and Extended Hungarian Algorithm.* Assume an acyclic, directed, bipartite graph  $G = (S, T, E)$  with a set  $S$  of source nodes  $s_i, i \in [1, n]$ , a set  $T$  of target nodes  $t_j, j \in [1, m]$ , and a set  $E$  of edges  $e_{ij}, i \in [1, n], j \in [1, m]$  with  $|E| = n \cdot m$ . In our application, source nodes are the attributes, target nodes are the classes, and edges are the correspondences between them. Further, assume a correspondence matrix  $C$  with entries  $c_{ij}$  quantifying the similarity between source node  $s_i$  and target node  $t_j$  (c. f. Tab. 1 without the columns and values in italics).

Assume also a set  $K$  of  $k$ -constraints  $k_j, k = 1, \dots, m$  where each  $k_j$  represents the number of assignments that are allowed for target node  $t_j$ . Given the input stated above, the 1:k assignment problem is to find a mapping  $M$  with nodes

$$m_{ij} = \begin{cases} 0 & \text{if } e_{ij} \text{ takes not part in the mapping} \\ 1 & \text{if } e_{ij} \text{ takes part in the mapping} \end{cases}$$

where

$$\sum_{i=1}^n m_{ij} \leq k_j \quad (\forall j = 1, \dots, m)$$

that maximizes the overall similarity of the selected participating nodes in the mapping:

$$\max \left( \sum_{\forall ij} c_{ij} \cdot m_{ij} \right)$$

The final assignment  $M$  is calculated using a global matching algorithm on the correspondence matrix. This mapping task can be solved with an extended version of the Hungarian Algorithm [7]. The Hungarian Algorithm solves the assignment problem [9], which does not allow the multi-mappings of multiple attributes to the same class. It also requires a square correspondence matrix, so  $n = m$ . In general, this is not the case ( $m \geq n$ ), so the extension is to pad  $C$  in order to construct  $C'$ . To this end ( $m - n$ ) additional rows have to be added representing non-existing source nodes:  $c'_{i'j} = 0 \quad \forall n < i' \leq m$ .

The  $k$ -constraints are incorporated by duplicating columns of  $C'$ . The value of  $k_j$  determines the total number of additional copies of the column. Note that this column

duplication requires  $\sum_{k \in K} k_j = a$  additional rows, since the matrix becomes broader, but still has to comply to the squareness condition. In total,  $(m - n) + a$  rows have to be added to generate a squared correspondence matrix with duplicated columns. See Tab. 1 for an example  $C'$  with  $K = \{k_1 = 3, k_3 = 2, \dots\}$ . It does not matter, where the additional columns or rows are inserted.  $k_i$  is also allowed to be infinite for classes that may appear arbitrarily often (e. g., boolean flags may be assigned unlimitedly). Since  $k_i$  cannot actually be set to infinity, it is sufficient to set  $k_i$  to the number of source attributes  $n$ . This gives every attribute the chance to become matched to this respective class.

**Table 1.** Extended correspondence matrix, now squared

Attributes (Source)/ Classes (Target)	Firstname	Firstname	Firstname	Lastname	Phone	Phone	Address	City
Fullname	0.8	0.8	0.8	0.6	0.1	0.1	0.2	0.3
Telephone	0.0	0.0	0.0	0.0	0.9	0.9	0.2	0.1
Street	0.2	0.2	0.2	0.4	0.1	0.1	0.9	0.7
House Number	0.0	0.0	0.0	0.0	0.7	0.7	0.7	0.2
<i>dummy</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
<i>dummy</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
<i>dummy</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
<i>dummy</i>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

With this squared correspondence matrix  $C'$ , the thus extended Hungarian Algorithm can be used to create a 1:k assignment:  $M'$ . However, the result has to be modified. Unfortunately, the Hungarian Algorithm involves all source attributes into the mapping, even those  $s_{i'}$  for which  $i' > n$ . Since the final mapping  $M$  shall have the same dimensions  $n \times m$ ,  $M'$  has to be transformed into  $M$ . This is achieved by removing all matches with the dummy  $s_{i'}$ .

$$(m_{ij}) = (m'_{i'j}) \quad i = i' = 1, \dots, n; \quad j = 1, \dots, m$$

After the mapping  $M$  is finally created, each attribute got a class assigned. We can now directly derive concrete similarity measures for the attributes of the input data. Subsequently, pairs of input records can be examined for duplicates using these similarity measures.

## 4 Evaluation

This paper describes the assignment of semantics to attributes in the context of duplicate detection. Therefore, we evaluate the classification results, rather than the effectiveness of the duplicate detection.

For classification we need instance data. We established ten test datasets with address data from various sources, described and available for download<sup>1</sup>. The datasets contain 14 attributes, on average, each dataset providing at least 50,000 tuples.

<sup>1</sup> <http://www.hpi.uni-potsdam.de/naumann/data>

The results of the complete classification process are shown in Tab. 2. As proposed by Euzenat and Shvaiko [2] we use F-Measure to describe the overall matching compliance to the manually defined gold standard. Columns 3 and 4 display the number of correctly classified attributes and the corresponding F-Measure, respectively.

**Table 2.** Combined classification results (dictionary and machine learning with Naïve Bayes)

Dataset	Number of attributes	Correct Matches	F-Measure	Close Matches	Close Match F-Measure
Fakenames	21	15	0.71	+3	0.86
Corporate	11	9	0.81	+1	0.91
Crawl1 (KT)	10	8	0.80	+0	0.80
Crawl2 (LN)	13	6	0.46	+2	0.62
Crawl3 (Po)	8	8	1.00	+0	1.00
Crawl4 (RW)	16	8	0.50	+3	0.69
ListB	9	5	0.56	+1	0.67
ListC	7	4	0.57	+1	0.71
Voters	23	12	0.52	+3	0.65
Mines	18	10	0.56	+2	0.67

In most cases, the majority of attributes has been successfully classified. The misses occur on more unusual attributes, such as credit card verification codes, UPS tracking numbers, religion, or occupation. They are not believed to be of utmost importance for the duplicate detection process.

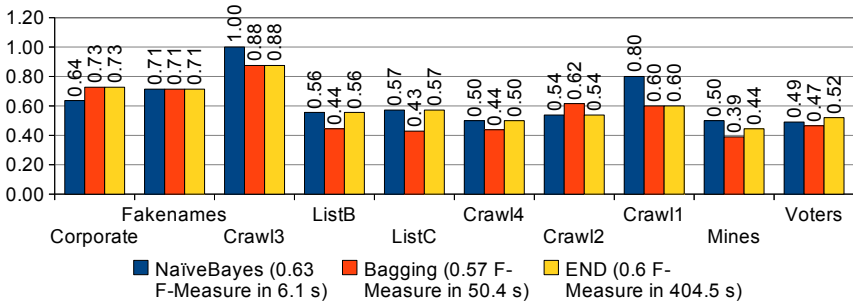
The derived similarity measure might not accurately make use of all the special characteristics of phone numbers, but might still achieve sound results. Therefore, we additionally counted “close matches”, such as weight/housenumber, place-of-birth/city-state-country-combination, or number of children/month (numeric). They are added to the strict matches and presented in Col. 5 in Tab. 2 with a corresponding F-Measure column, increasing the average F-Measure from 0.61 by 20 % to 0.73.

We also compared 1:k matching results against (unbounded) 1:n matching. With the chosen set of k-constraints, 1:k assignments are never worse than 1:n assignments. There are cases in which 1:k yields better results, especially when the classifier itself is more simplistic. This is positive, because in general, it cannot be assumed that the classes are always known and that example data is available. Such lack results in reduced classification quality and 1:k raises the F-Measure of the final assignment.

Finally, Fig. 3 shows a comparison of different machine learning algorithms and the achieved F-Measure on all datasets. Naïve Bayes provides the best cost-benefit tradeoff and was used for all the experiments. The figure’s legend shows the average F-Measure and the average classification time for each dataset and each classifier.

## 5 Related Work

*Duplicate Detection.* Efficient duplicate detection is a process that consists of two merely independent parts. First, an algorithm picks promising pairs of the set of records. Well-known heuristics are the sorted neighborhood method by Hernandez and Stolfo [5]



**Fig. 3.** The influence of different classifiers on the result is small, however the Naïve Bayes classifier is one order of magnitude faster than the others. The used classifiers are Naïve Bayes, Bagging, and Ensemble of Nested Dichotomies (END).

and its extensions, such as Monge and Elkan’s [6], as well as blocking or indexing approaches [3].

Second, a pair of duplication candidates has to be examined using a similarity measure. In case of relational data, it is common that (a subset of) the attributes are compared. With our approach, highly specialized measurements can be used. An overview on both areas can be found in [8] or [1].

*Schema Matching.* Schema matching is the technique of creating and selecting correspondences between two sets of elements, typically attributes of relations. A elaborate survey was written by Euzenat and Shvaiko [2]. Instance mappings are used in iFuice [11], where knowledge about explicit connections between different schemas is exploited. However, in the use case of customer data, those hyperlink connections are not available.

## 6 Conclusion and Outlook

We presented a technique for the automatic assignment of semantic classes to attributes of a given dataset. The only prerequisite is the availability of training data for the desired domain in form of examples and/or reference data. The matching can be further improved by providing  $k$ -constraints for the  $1:k$  matching. With this matching approach, the most relevant attributes can be identified and appropriate similarity measures can be derived.

For future work, we plan to extend the knowledge connected with the semantic classes. For example, the weighting of attributes should be different (family name vs. country) and some attributes might even be ignored, because they do not carry beneficial information for the similarity measure, e. g., IDs. This can be achieved by another learning phase with the use of known duplicate records. To further ease the usefulness for schema matching, the semantic classes will be augmented with a well-known vocabulary, e. g. with WordNET or YAGO. This helps in postprocessing steps and fosters interoperability.

Relations between different attributes should be taken into consideration. I. e., the presence of English street, city, and state names implies the existence of English ZIP codes. Another way of connecting them is the matching to composite attributes (e. g., given and family name together as full name). Finally, privacy aspects can be incorporated by not using the original values but transforming them into a metric space on client side [10].

**Acknowledgements.** We thank Dustin Lange for his valuable feedback and help on the machine learning aspects of this work.

## References

1. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. *IEEE Transactions on Knowledge & Data Engineering (TKDE)* 19 (2007)
2. Euzenat, J., Shvaiko, P.: *Ontology matching*. Springer, Heidelberg (2007)
3. Ferro, A., Giugno, R., Puglisi, P.L., Pulvirenti, A.: An Efficient Duplicate Record Detection Using q-grams Array Inverted Index. In: Bach Pedersen, T., Mohania, M.K., Tjoa, A.M. (eds.) *DAWAK 2010. LNCS*, vol. 6263, pp. 309–323. Springer, Heidelberg (2010)
4. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. *SIGKDD Exploration Newsletter* 11(1) (2009)
5. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 127–138 (1995)
6. Monge, A., Elkan, C.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: *Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery, DMKD* (1997)
7. Munkres, J.: Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics (SIAM)* 5(1) (1957)
8. Naumann, F., Herschel, M.: *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers (March 2010)
9. Papadimitriou, C.H., Steiglitz, K.: *Combinatorial optimization: algorithms and complexity*. Prentice-Hall (1982)
10. Scannapieco, M., Figotin, I., Bertino, E., Elmagarmid, A.K.: Privacy preserving schema and data matching. In: *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (2007)
11. Thor, A.: *Automatische Mapping-Verarbeitung von Web-Daten*. Dissertation, Institut für Informatik, Universität Leipzig (2007)

# Matching and Alignment: What Is the Cost of User Post-Match Effort?\*

## (Short Paper)

Fabien Duchateau<sup>1</sup>, Zohra Bellahsene<sup>2</sup>, and Remi Coletta<sup>2</sup>

<sup>1</sup> Norwegian University of Science and Technology,  
NO-7491 Trondheim, Norway  
fabierend@idi.ntnu.no

<sup>2</sup> LIRMM - Université Montpellier 2,  
161 rue Ada, 34392 Montpellier, France  
{firstname.lastname}@lirmm.fr

**Abstract.** Generating new knowledge from scientific databases, fusioning products information of business companies or computing an overlap between various data collections are a few examples of applications that require data integration. A crucial step during this integration process is the discovery of correspondences between the data sources, and the evaluation of their quality. For this purpose, the *overall* metric has been designed to compute the post-match effort, but it suffers from major drawbacks. Thus, we present in this paper two related metrics to compute this effort. The former is called **post-match effort**, i.e., the amount of work that the user must provide to correct the correspondences that have been discovered by the tool. The latter enables the measurement of **human-spared resources**, i.e., the rate of automation that has been gained by using a matching tool.

## 1 Introduction

Data integration has now been studied for years, and many applications still make this research field an interesting challenge. Discovering correspondences between the data sources is one of the first steps of this integration process. As pointed out by [1], the quality obtained during this step mainly determines the quality of the whole data integration process. For this reason, matching communities (both schema and ontology) have been very prolific in producing matching tools during the last decades to automate the discovery of correspondences. Many surveys [2–5] and books [6, 7] reflect this interest.

To evaluate the results produced by their tools, these communities mainly use common quality metrics such as precision, recall, and F-measure. However, the aim of (semi-)automatic matching is to avoid a manual, labor and error-prone process. The post-match effort, which consists of checking the discovered

---

\* Supported by ANR DataRing ANR-08-VERSO-007-04. The first author carried out this work during an ERCIM “Alain Bensoussan” Fellowship Programme.

correspondences and searching for the missing ones, should therefore be reduced at most. Yet, the available metrics hardly provide an estimation of this effort. F-measure is the harmonic mean between precision and recall, while it should add the correction cost of both measures. On the other hand, the overall (or accuracy) is a first attempt to evaluate the post-match effort [8].

Consequently, we propose a **post-match effort** measure and its inverse **human spared resources** which tackle these issues. It estimates the number of user interactions required to correct both precision and recall (i.e., to manually obtain a 100% F-measure). Thus, it takes into account the effort to (in)validate discovered correspondences, but also the search for missing ones between the data sources. This measure is sufficiently generic to be converted into the range  $[0, 1]$  or in time units (e.g., seconds) and it does not require other specific inputs than those needed to compute precision, F-measure or overall.

## 2 Preliminaries

**Correspondences** are semantic links between elements of different data sources (schemas, ontologies) which represent the same real-world concept. Contrary to [9], evaluating the quality of the mapping (i.e., the transformation function between instances of one element into those of another element) is out of scope of this paper since we focus on correspondences. We also limit correspondences to  $1:1$  (i.e., one element is matched to only one element) or to  $1:n$  (i.e., one element is matched to several elements). Currently, only a few tools produce  $n:m$  correspondences. Figure 1(b) depicts an example of two schemas (from *hotel booking* web forms) and the correspondences discovered by a matching tool.

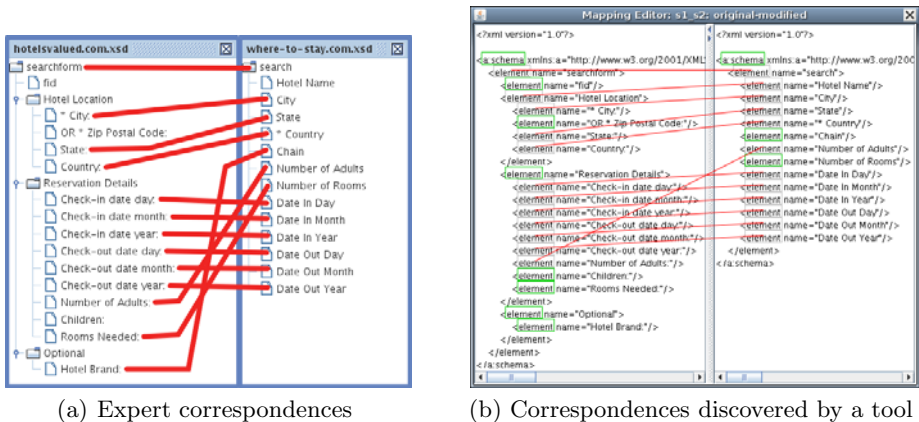


Fig. 1. A running example : hotel booking webforms

A **matching dataset** is composed of a set of data sources (schemas, ontologies) to be matched and the set of expert correspondences. This set of expert



correspondences is considered as complete and trustful. Such datasets, also called testbeds or test collections are used by most evaluation tools as an oracle, against which they can compare different approaches or tools. To evaluate the matching quality, three measures are commonly accepted in the literature. **Precision** calculates the proportion of correct correspondences extracted among the discovered ones. Another typical measure is **recall** which computes the rate of correct discovered correspondences among all correct ones. **F-measure** is a trade-off between precision and recall. We propose to complete these measures with our post-match effort measure, which is presented in the next section.

### 3 Post-Match Effort Metric

We present in this paper two related metrics: **post-match effort** and **human-spaced resources**.

#### 3.1 Intuition and Running Example

A set of discovered correspondences, provided by a schema matching tool, has two issues, namely (i) incorrect discovered correspondences and (ii) missing (correct) correspondences. Users first have to check each correspondence from the set, either to (in)validate or complete it (in case of 1:n correspondences). Then, they have to browse the schemas and discover the missing correspondences. Thus, we propose to evaluate this user post-match effort by **computing the number of user interactions** to reach a 100% F-measure, i.e., to correct the two previously mentioned issues. A user interaction is an (in)validation of one pair of schema elements (either from the set of discovered correspondences or between the schemas). We first introduce three assumptions which underlie our metric:

- **Worst case**, which means that all pairs of schema elements, which have not already been matched, must be (in)validated. In addition, the last (in)validated pair would be a correspondence.
- **Uniformity**, i.e., missed correspondences are discovered with the same frequency (and not at random). Although not realistic, this assumption mainly enables a fair comparison when evaluating the post-match effort for different tools. The worst case assumption anyhow guarantees that the last validated pair is a correct correspondence.
- Only **correspondences 1:1** are taken into account. The metric can be applied with *1:n correspondences* (represented by several 1:1 correspondences), but we do not consider more complex correspondences (namely *n:m*). However, we note that a post-processing technique could transform the 1:1 validated correspondences into complex correspondences by relying on the data.

Now, let us introduce an example. Figure [1\(b\)](#), presented in the preliminaries section, depicts a set of correspondences discovered by a matching tool between two hotel booking schemas. The expert set of correspondences is shown by figure [1\(a\)](#). We notice that one discovered correspondence is incorrect: (*Hotel Location*,

*Hotel Name*). Consequently, it has to be invalidated. Besides, the matching tool has missed two correspondences, namely (*Hotel Brand*.; *Chain*) and (*Rooms Needed*.; *Number of Rooms*). These two correspondences have to be searched among the 23 pairs that have not been validated ( $8 \times 3$  possible pairs minus 1 incorrect pair discovered by the tool).

### 3.2 Estimating the Number of User Interactions

We define the number of user interactions as a positive number which represents the number of user interactions to obtain a 100% F-measure from a set of discovered correspondences. It consists of two steps which are described below.

Given two schemas  $S_\ell$  and  $S_L$  of respective sizes  $|S_\ell|$  and  $|S_L|$ , with  $|S_\ell| \leq |S_L|$  (i.e.,  $S_L$  is a larger schema than  $S_\ell$ ), their expert set of correspondences  $E$  contains  $|E|$  correspondences. A matching tool applied against these schemas has discovered a set of correspondences  $M$ , which contains  $|M|$  correspondences. Among these discovered correspondences,  $|R|$  of them are correct, with  $0 \leq |R| \leq |M|$ . To compute the number of user interactions, only the five inputs  $|S_\ell|$ ,  $|S_L|$ ,  $|E|$ ,  $|M|$  and  $|R|$  are required. In our example, we have the following values:

- $|S_\ell| = 14$ , the number of elements in the smallest schema<sup>1</sup>.
- $|S_L| = 19$ , the number of elements in the largest schema<sup>1</sup>.
- $|E| = 13$ , the number of expert correspondences
- $|M| = 12$ , the number of correspondences discovered by the matching tool, shown in figure 1(b).
- $|R| = 11$ , the number of correct correspondences discovered by the matching tool.

**Step 1: checking of all discovered correspondences.** This step is very easy to compute. A user has to check each correspondence from the set of discovered correspondences, and (in)validate it. Thus, this requires a number of interactions equal to the number of discovered correspondences in the set,  $|M|$  in our case. We call this metric  $\text{effort}_{prec}$  since it is directly impacted by precision. Indeed, a high precision reduces the number of user interactions since there are fewer incorrect correspondences which have been discovered. Note that at the end of this step, the precision value is equal to 100%.

$$\text{effort}_{prec} = |M| \tag{1}$$

In our example, there are 12 discovered correspondences, thus  $\text{effort}_{prec} = 12$ . It means that the number of user interactions during this step is equal to 12, among which 11 validations and 1 invalidation for the incorrect correspondence.

**Step 2: manual discovery of missed correspondences.** The second step deals with the manual discovery of all missing correspondences. At the end of this step, recall reaches 100%, and F-measure too. We assume that all pairs which have not been invalidated yet must be analyzed by the user. As we consider only 1:1 correspondences, elements that have already been matched are not checked

<sup>1</sup> We do not count the root element tagged with  $\langle a:schema \rangle$ .

anymore. The main idea is to check every unmatched element from the smallest schema against all unmatched elements from the largest schema.

Due to the uniformity assumption, we manually discover a missing correspondence with the same frequency. This frequency is computed by dividing the number of unmatched elements in the smallest schema by the number of missing correspondences, as shown by Formula 2. Thanks to 1:1 correspondences assumption, the number of correct correspondences  $|R|$  is at most equal to the number of correctly matched elements in each schema (i.e.,  $0 \leq |R| \leq |S_\ell|$ ). Hence we can compute  $|S_\ell| - |R|$ .

$$\text{freq} = \frac{|S_\ell| - |R|}{|E| - |R|} \tag{2}$$

Back to our example,  $\text{freq} = \frac{14-11}{13-11} = \frac{3}{2}$  means that the user will manually find a missing correspondence for every three unmatched elements from the smallest schema.

Since we now know the frequency, we can compute the number of interactions using a sum function. We call this metric  $\text{effort}_{rec}$  since it is affected by recall. The higher recall you achieved, the fewer interactions you require during this step.  $|S_L| - |R|$  denotes the number of unmatched elements from the largest schema. With  $i$  standing for the analysis of the  $i^{th}$  unmatched element from  $S_\ell$ ,  $\frac{i}{\text{freq}}$  represents the discovery of a missing correspondence (when it reaches 1). We also uniformly remove the pairs which may have been already invalidated during step 1, by computing  $\frac{|M|-|R|}{|S_\ell|-|R|}$ . Thus, we obtain this Formula 3:

$$\text{effort}_{rec} = \sum_{i=1}^{|S_\ell|-|R|} (|S_L| - |R| - \frac{i}{\text{freq}} - \frac{|M| - |R|}{|S_\ell| - |R|}) \tag{3}$$

To sum up, for each unmatched of the smallest schema, the user has to analyze all elements of the largest schema, except for those already matched ( $|R|$ ), those already part of a match previously discovered ( $\frac{i}{\text{freq}}$ ) and those invalidated during the first step ( $\frac{|M|-|R|}{|S_\ell|-|R|}$ ). We now detail for our example the successive iterations of this sum function, which vary from 1 to 3.

- $\text{effort}_{rec}(i = 1), 19 - 11 - \frac{1}{1.5} - \frac{1}{3} = 7$
- $\text{effort}_{rec}(i = 2), 19 - 11 - \frac{2}{1.5} - \frac{1}{3} = 6\frac{1}{3}$
- $\text{effort}_{rec}(i = 3), 19 - 11 - \frac{3}{1.5} - \frac{1}{3} = 5\frac{2}{3}$

Thus, the second step to discover all missing correspondences requires  $\text{effort}_{rec} = 7 + 6\frac{1}{3} + 5\frac{2}{3} = 19$  user interactions.

Finally, to compute the number of user interactions between two schemas  $S_\ell$  and  $S_L$ , noted  $\text{nui}$ , we need to sum the values of the two steps, thus resulting in Formula 4. If the set of correspondences is empty, then using a matching tool was useless and the number of user interactions is equal to the number of pairs between the schemas.

$$\text{nui}(S_\ell, S_L) = \begin{cases} |S_\ell| \times |S_L| & \text{if } |M| = 0 \\ \text{effort}_{prec} + \text{effort}_{rec} & \text{otherwise} \end{cases} \tag{4}$$

In our example, the user needs a number of user interactions  $nui = 12 + 19 = 31$  to correct the set of correspondences produced by the tool.

### 3.3 Normalization and Generalization

The number of user interactions is not sufficient to measure the benefit of using a matching tool. Indeed, a given number of interactions may appear as an incredible effort for correcting the set of correspondences of two small data sources, but it may seem acceptable when dealing with large data sources. Thus, our post-match effort (and its inverse, human spared resources) is a normalization of this number of user interactions based on the size of the data sources. Then, we explain how to generalize the post-match effort when there are more than two data sources.

**Normalization.** From the number of user interactions, we can normalize the **post-match effort** value into  $[0,1]$ . It is given by Formula 5. Indeed, we know the number of possible pairs ( $|S_\ell| \times |S_L|$ ). Checking all these pairs means that the user performs a manual matching,  $nui = |S_\ell| \times |S_L|$  and  $pme = 100\%$ .

$$pme(S_\ell, S_L) = \frac{nui(S_\ell, S_L)}{|S_\ell| \times |S_L|} \quad (5)$$

We can also compute the percentage of automation of the matching process thanks to a matching tool. This metric, noted *hsr*, for **human spared resources**, is given by Formula 6. This measure enables the computation of the rate of automation by the matching process.

$$hsr(S_\ell, S_L) = 1 - \frac{nui(S_\ell, S_L)}{|S_\ell| \times |S_L|} = 1 - pme(S_\ell, S_L) \quad (6)$$

If a matching tool achieves a 20% post-match effort, this means that the user has to perform a 20% manual matching for removing and adding correspondences, w.r.t. a complete (100%) manual matching. Consequently, we can deduce that the matching tool managed to automate 80% of the matching process. In our dating example, the post-match effort is equal to  $pme = \frac{31}{14 \times 19} \simeq 12\%$  and human spared resources is equal to  $hsr = 1 - 0.12 \simeq 88\%$ . The matching tool has spared 88% resources of the user, who still has to manually perform 12% of the matching process.

**Generalization.** As matching scenarios may contain more than two data sources, we need to generalize the post-match effort formula. Let us consider that a matching scenario contains  $n$  data sources such as a set  $\langle S_1, S_2, \dots, S_n \rangle$ . The generalized post-match effort, noted  $pme_{gen}$ , is given by Formula 7. It is the sum of all numbers of user interactions in all possible couples of data sources, divided by the sum of all numbers of pairs in all possible couples of data sources.

$$pme_{gen} = \frac{\sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} nui(S_i, S_j)}{\sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} |S_i| \times |S_j|} \quad (7)$$

## 4 Related Work

To the best of our knowledge, the overall measure (also named accuracy in [8]) is the only one to compute a post-match effort [10]. It is computed with the following formula in the range  $[-\infty, 1]$ :

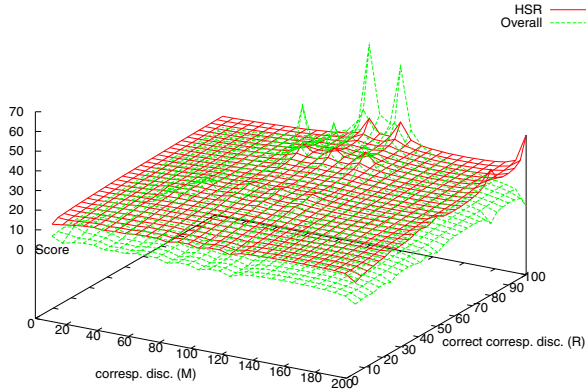
$$\text{Overall} = \text{Recall} \times \left( 2 - \frac{1}{\text{Precision}} \right) \quad (8)$$

A major drawback of this measure deals with the fact that removing irrelevant correspondences is considered as difficult (in terms of user effort) as adding missed correspondences. However, this is rarely the case in real-world scenarios. Another drawback explained by the authors deals with a precision below 50%: it implies more effort from the user to remove extra correspondences and add missing ones than to manually do the matching, thus resulting in a negative overall value which is often disregarded. On the contrary, our measure returns values in the range  $[0, 1]$  and it does not assume that a low precision involves much effort during post-match. Finally, the overall measure does not consider the size of the data sources. Yet, even with the same number of expert correspondences, the manual task for checking the discovered correspondences and finding the missed correspondences in two large data sources requires a larger effort than in small data sources. To sum up this comparison, overall is mainly more pessimistic than HSR. This is illustrated by Figure 2 which depicts the values of overall and HSR when the number of discovered correspondences ( $|M|$ ) and the number of correct discovered correspondences ( $|R|$ ) vary. In this plot, two parameters are fixed: the number of expert correspondences  $|E|$  to 200 and the average size of the data sources  $|S|$  to 500. All overall values are less than 0 when the number of correct discovered correspondences is at most half of the number of discovered correspondences, which is an obvious limitation. We also notice that overall may be more optimistic than HSR with high precision and recall values (in our plot, when  $|M|$  and  $|R|$  are close to  $|E|$ ). The reason deals with the size of the data sources, which is not considered by overall. These comments are verified with other values of  $|E|$  and  $|S|$ . Due to page limit, all plots are available online<sup>2</sup>.

## 5 Conclusion

In this paper, we have presented a former metric which computes the post-match effort while the latter estimates the percentage of automation due to the use of a matching tool. The scores computed by our measures and presented as a number of user interactions can be converted in time units. In addition, except for the size of the data sources, computing these metrics does not require more information than traditional quality measures. As a future work, we first intend to extend our measure so that it takes into account the top-K correspondences returned by several matching tools. Then, we would like to quantify pre-match effort too.

<sup>2</sup> Appendix at [http://november.idi.ntnu.no/~sim\\$fabien/appendixCoopis11](http://november.idi.ntnu.no/~sim$fabien/appendixCoopis11)



**Fig. 2.** A Comparison of Overall and HSR

## References

1. Smith, K., Morse, M., Mork, P., Li, M., Rosenthal, A., Allen, D., Seligman, L.: The role of schema matching in large enterprises. In: CIDR (2009)
2. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
3. Yatskevich, M.: Preliminary evaluation of schema matching systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento (2003)
4. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)
5. Noy, N.F., Doan, A., Halevy, A.Y.: Semantic integration. AI Magazine 26(1), 7–10 (2005)
6. Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Heidelberg (2007)
7. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping. Springer, Heidelberg (2011)
8. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, pp. 117–128 (2002)
9. Alexe, B., Tan, W.C., Velegrakis, Y.: STBenchmark: towards a benchmark for mapping systems. Proceedings of the VLDB 1(1), 230–244 (2008)
10. Do, H.-H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) NODE-WS 2002. LNCS, vol. 2593, pp. 221–237. Springer, Heidelberg (2003)

## Author Index

- Abbaci, Katia 38  
Acar, Aybar C. 367  
Adamus, Radoslaw 734  
Alhamad, Mohammed 469  
Andreescu, Laura Maria 163  
Anisetti, Marco 560  
Ardagna, Claudio A. 560
- Baumgrass, Anne 329  
Beitman, Karin Koogan 781  
Bellahsene, Zohra 421, 800  
Bouzeghoub, Mokrane 38  
Buche, Patrice 662  
Buckley, Ingrid 560
- Canavese, Daniele 617  
Casanova, Marco Antonio 781  
Cesena, Emanuele 617  
Chang, Elizabeth 469  
Chen, Qiming 403, 525  
Chowdhury, Nafisa Afrin 826  
Ciuciu, Ioana 605  
Claerhout, Brecht 605  
Coletta, Remi 421, 800  
Conforti, Raffaele 100  
Cruz Torres, Mario Henrique 155
- Dadam, Peter 82  
Damiani, Ernesto 560  
Danilow, Juan 716  
De Virgilio, Roberto 644  
Dibie-Barthélemy, Juliette 662  
Dillon, Tharam 469  
Domingues, Helves Humberto 818  
Dou, Dejing 698, 826  
Dougherty, Brian 432  
Duchateau, Fabien 421  
Dustdar, Schahram 451
- Eder, Johann 763  
Edmondson, James 542  
Ekanayake, Chathura C. 20  
Eshuis, Rik 119
- Farokhi, Soodeh 625  
Fauvet, Marie-Christine 20
- Fernandez, Eduardo B. 560  
Ferreira, João Eduardo 818  
Fleischhacker, Daniel 680  
Fortino, Giancarlo 100  
Furtado, Antonio Luz 781
- Gal, Avigdor 2  
Gama, Kiev 498  
Ghaffari, Amir 625  
Gokhale, Aniruddha 432, 507, 542  
Gomes, Raphael Valle A. 781  
Gorawski, Marcin 347  
Grefen, Paul 119  
Greveler, Ulrich 577  
Grigori, Daniela 38
- Hadjali, Allel 38  
Hakiri, Akram 507  
Herrmann, Klaus 236  
Hill, James H. 478  
Hoffert, Joe 507  
Holvoet, Tom 155  
Hsu, Meichun 403, 525  
Hummer, Waldemar 451
- Ibănescu, Liliana 662
- Jin, Tao 56  
Justus, Benjamin 577
- Kang, Yong-Bin 218  
Kegley, Russell 432  
Knuplesch, David 82  
Kojima, Isao 808  
Kon, Fabio 818  
Kop, Christian 747  
Köpke, Julius 763  
Koster, Andrew 182  
Kowalski, Tomasz Marek 734  
Krishnaswamy, Shonali 218
- Lampo, Tomas 716  
La Rosa, Marcello 20, 100  
Leitner, Philipp 451  
Lemos, Fernando 38

- Lemos, João 302  
 Lenhard, Jörg 137  
 Leone, Stefania 284  
 Liétard, Ludovic 38  
 Lima, Ricardo 498  
 Lincoln, Maya 2  
 Liu, Haishan 698  
 Liu, Xi 64  
 Loehr, Dennis 577  
 Lopes, Danilo 498  
 Ly, Linh Thao 82  
 Lynden, Steven 808  
  
 Macedo, José A.F. 781  
 Matono, Akiyoshi 808  
 Meersman, Robert 605  
 Mendling, Jan 329  
 Motro, Amihai 367  
  
 Naumann, Felix 412  
 Ngo, DuyHoa 800  
 Nikraves, Ali 625  
 Norrie, Moira C. 284  
  
 Onaindia, Eva 200  
  
 Pajares Ferrando, Sergio 200  
 Pascal, Berthou 507  
 Preston, Jonathan 432  
  
 Ribe-Baumann, Liz 385  
 Ribeiro, J.T.S. 274  
 Rinderle-Ma, Stefanie 82  
 Rocacher, Daniel 38  
 Rosa, Nelson 498  
 Rothermel, Kurt 236  
 Ruckhaus, Edna 716  
  
 Sabater-Mir, Jordi 182  
 Sacha, Krzysztof 588  
 Sadjadi, Masoud 560  
 Salas, Percy E. Rivera 781  
 Sahnikov-Tarnovski, Nikita 635  
 Satzger, Benjamin 451  
 Schefer, Sigrid 329  
 Schilders, Louis 605  
  
 Schmidt, Douglas C. 432, 478, 507, 542  
 Schönberger, Andreas 137  
 Schorlemmer, Marco 182  
 Shams, Fereidoon 625  
 Siedlecki, Zacheusz 347  
 Silvestro, Jacopo 617  
 Simão, José 302  
 Smiraglia, Paolo 617  
 Šor, Vladimir 635  
 Souza, Fabio 498  
 Srirama, Satish Narayana 635  
 Strembeck, Mark 329  
 Su, Jianwen 64, 256  
  
 Tanimura, Yusuke 808  
 ter Hofstede, Arthur H.M. 20, 100  
 Thierry, Gayraud 507  
 Torreño, Alejandro 200  
 Touhami, Rim 662  
 Tryfonopoulos, Christos 163  
  
 Veiga, Luís 302  
 Vidal, María-Esther 716  
 Vidal, Vania M.P. 781  
 Vogel, Tobias 412  
 Völker, Johanna 680  
 Vonk, Jochem 119  
  
 Wang, Jianmin 56  
 Weijters, A.J.M.M. 274  
 Wen, Lijie 56  
 White, Jules 432  
 Wirtz, Guido 137  
 Wiślicki, Jacek 734  
 Wolf, Hannes 236  
  
 Xu, Lai 321  
 Xu, Wei 256  
  
 Yan, Zhimin 256  
 Yang, Jian 64, 256, 321  
  
 Zaslavsky, Arkady 218  
 Zhang, Liang 256  
 Zhao, Weiliang 321  
 Zhong, Youliang 321