

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 Hervé Panetto
Tharam Dillon Stefanie Rinderle-Ma
Peter Dadam Xiaofang Zhou
Siani Pearson Alois Ferscha
Sonia Bergamaschi Isabel F. Cruz (Eds.)

On the Move to Meaningful Internet Systems: OTM 2012

Confederated International Conferences:
CoopIS, DOA-SVI, and ODBASE 2012
Rome, Italy, September 10-14, 2012
Proceedings, Part I

Volume Editors

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

Hervé Panetto, University of Lorraine, France; herve.panetto@univ-lorraine.fr

Tharam Dillon, La Trobe University, Australia; tharam.dillon7@gmail.com

Stefanie Rinderle-Ma, Universität Wien, Austria; stefanie.rinderle-ma@univie.ac.at

Peter Dadam, Universität Ulm, Germany; peter.dadam@uni-ulm.de

Xiaofang Zhou, University of Queensland, Australia; zxf@itee.uq.edu.au

Siani Pearson, HP Labs, UK; siani.pearson@hp.com

Alois Ferscha, Johannes Kepler Universität, Austria; ferscha@soft.uni-linz.ac.at

Sonia Bergamaschi, University of Modena, Italy; sonia.bergamaschi@unimore.it

Isabel F. Cruz, University of Illinois at Chicago, IL, USA; isabelcfcruz@gmail.com

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-33605-8

e-ISBN 978-3-642-33606-5

DOI 10.1007/978-3-642-33606-5

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012947230

CR Subject Classification (1998): D.2.2, D.2.11-12, H.3.4-5, I.2.4, C.2.4, J.1, K.4.4, K.6.5, C.2.0, H.4.1-3, H.5.2-3, H.2.3-4, F.4.1, F.3.2, I.2.6-7, F.1.1, K.6.3

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

© Springer-Verlag Berlin Heidelberg 2012

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)

General Co-Chairs' Message for OnTheMove 2012

The OnTheMove 2012 event held in Rome, during September 10–14, further consolidated the importance of the series of annual conferences that was started in 2002 in Irvine, California. It then moved to Catania, Sicily in 2003, to Cyprus in 2004 and 2005, Montpellier in 2006, Vilamoura in 2007 and 2009, in 2008 to Monterrey, Mexico, and was held in Heraklion, Crete, in 2010 and 2011. The event continues to attract a diverse 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 applicational issues. As we all realize, email, the internet, and even video conferences are not by themselves optimal nor sufficient for effective and efficient scientific exchange.

The OnTheMove (OTM) Federated Conference series was 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 year's OTM Federated Conferences event therefore once more provided an opportunity for researchers and practitioners to understand, discuss, and publish these developments within the broader context of distributed, ubiquitous computing. To further promote synergy and coherence, the main conferences of OTM 2012 were conceived against a background of three interlocking global themes:

- Virtual and Cloud Computing Infrastructures and Security
- Technology and Methodology for an Internet of Things and its Semantic Web
- Collaborative and Social Computing for and in 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), covering the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. In 2011 security issues, originally topics of the IS workshop (since 2007), became an integral part of DOA as “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 simultaneously incorporates (a) theory, (b) conceptual design and development, (c) methodology and pragmatics, and (d) application in particular case studies and industrial solutions.

As in previous years 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 emergent areas related to the general area of Web-based distributed computing. We were also glad to see that five of our earlier successful workshops (EI2N, OnToContent, ORM, INBAST, and SeDeS) re-appeared in 2012, in some cases for the fifth or even seventh time, and often in alliance with other older or newly emerging workshops. Three brand-new independent workshops could be selected from proposals and hosted: META4eS, SINCOM, and SOMOCO. The Industry Track, started in 2011 under the auspicious leadership of Hervé Panetto and OMG's Richard Mark Soley, further gained momentum and visibility.

Incidentally, 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 most happy to see that once more in 2012 the number of quality submissions for the OnTheMove Academy (OTMA) substantially increased. OTMA implements our unique interactive formula to bring PhD students together, and aims to represent our “vision for the future” in research in the areas covered by OTM. It is managed by a dedicated team of collaborators led by Peter Spyns and Anja Metzner, and of course by the OTMA Dean, Erich Neuhold. In the OTM Academy, PhD research proposals are submitted by students for peer review; selected submissions and their approaches are then presented by the students in front of a wider audience at the conference, and are independently and extensively analyzed and discussed in front of this audience by a panel of senior professors.

As said, all three main conferences and the associated workshops share 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 2012, the primary emphasis stayed on the distributed object infrastructure and its virtual and security aspects; for ODBASE 2012, 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 2012, the focus as usual was on the interaction of such technologies and methods with business process 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 in previous years, the organizers wanted to stimulate this cross-pollination by a program of famous keynote speakers, focusing on the chosen themes and shared by all OTM component events. We were quite proud to announce this year

- Ed Parsons, Google Inc, USA;
- Maurizio Lenzerini, U. di Roma La Sapienza, Italy;
- Volkmar Lotz, SAP Research, France;
- Manfred Reichert, U. of Ulm, Germany;
- Guido Vetere, IBM, Italy.

We received a total of 169 submissions for the three main conferences and 127 submissions in total for the workshops, an almost 20% increase compared with those for 2011. 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 CoopIS 2012 conferences decided to accept only approximately 1 full paper for every 5 submitted, while the ODBASE 2012 PC accepted less than 1 paper out of 3 submitted, not counting posters. For the workshops and DOA-SVI 2012 the acceptance rate varies but the aim was to stay consistently at about 1 accepted paper for 2-3 submitted, and this as always subordinated to proper peer assessment of scientific quality. As usual we have separated the proceedings into two volumes with their own titles, one for the main conferences and one for the workshops and posters, and we are again most grateful to the Springer LNCS team in Heidelberg for their professional support, suggestions and meticulous collaboration in producing the files ready for downloading on the USB sticks.

The reviewing process by the respective OTM Program Committees was as always performed to professional standards: each paper submitted to the main conferences was reviewed by at least three referees, with arbitrated email discussions in the case of strongly diverging evaluations. It may be worthwhile to emphasize that it is an explicit OnTheMove policy that all conference Program Committees and Chairs make their selections completely autonomously from the OTM organization itself. As in recent years, proceedings on paper were by separate request and order, and incurred an extra charge.

The General Chairs are once more especially grateful to the many people directly or indirectly involved in the setup 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 2012 certainly also financial risk, that is entailed by the organization of an event like OTM. Apart from the persons in their roles mentioned above,

we therefore wish to thank in particular explicitly our 7 main conference PC Chairs:

- CoopIS 2012: Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou;
- ODBASE 2012: Sonia Bergamaschi, Isabel F. Cruz;
- DOA-SVI 2012: Siani Pearson, Alois Ferscha.

And similarly the 2012 OTMA and Workshops PC Chairs (in order of appearance on the website): Hervé Panetto, Michele Dassisti, J. Cecil, Lawrence Whitman, Jinwoo Park, Rafael Valencia García, Thomas Moser, Ricardo Colomo Palacios, Ioana Ciuciu, Anna Fensel, Amanda Hicks, Matteo Palmonari, Terry Halpin, Herman Balsters, Yan Tang, Jan Vanthienen, Wolfgang Prinz, Gregoris Mentzas, Fernando Ferri, Patrizia Grifoni, Arianna D'Ulizia, Maria Chiara Caschera, Irina Kondratova, 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 all also owe a serious debt of gratitude to our supremely competent and experienced Conference Secretariat and technical support staff in Brussels and Guadalajara, Jan Demey, Daniel Meersman, and Carlos Madariaga.

The General Chairs also thankfully acknowledge the academic freedom, logistic support, and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB); Université de Lorraine CNRS, Nancy; and Universidad Politécnica de Madrid (UPM), without which such a project 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!

July 2012

Robert Meersman
Hervé Panetto
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 2012 Federated Conferences, consisting of three conferences, namely, CoopIS 2012 (Cooperative Information Systems), DOA-SVI 2012 (Secure Virtual Infrastructures), and ODBASE 2012 (Ontologies, Databases, and Applications of Semantics).

Executive Committee

General Co-Chairs

Robert Meersman	VU Brussels, Belgium
Tharam Dillon	La Trobe University, Melbourne, Australia
Pilar Herrero	Universidad Politécnica de Madrid, Spain

OnTheMove Academy Dean

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

Industry Case Studies Program Chair

Hervé Panetto	University of Lorraine, France
---------------	--------------------------------

CoopIS 2012 PC Co-Chairs

Stefanie Rinderle-Ma	University of Vienna, Austria
Peter Dadam	Ulm University, Germany
Xiaofang Zhou	University of Queensland, Australia

DOA-SVI 2012 PC Co-Chairs

Siani Pearson	HP Labs, UK
Alois Ferscha	Johannes Kepler Universität Linz, Austria

ODBASE 2012 PC Co-Chairs

Sonia Bergamaschi	Università di Modena e Reggio Emilia, Italy
Isabel F. Cruz	University of Illinois at Chicago, USA

Logistics Team

Daniel Meersman
Carlos Madariaga
Jan Demey

CoopIS 2012 Program Committee

Marco Aiello	Massimo Mecella
Joonsoo Bae	Jan Mendling
Zohra Bellahsene	John Miller
Nacer Boudjlida	Arturo Molina
James Caverlee	Nirmal Mukhi
Vincenzo D'Andrea	Miyuki Nakano
Xiaoyong Du	Moira C. Norrie
Schahram Dustdar	Selmin Nurcan
Kaushik Dutta	Werner Nutt
Johann Eder	Gerald Oster
Rik Eshuis	Hervé Panetto
Renato Fileto	Barbara Pernici
Hong Gao	Lakshmish Ramaswamy
Ted Goranson	Manfred Reichert
Paul Grefen	Antonio Ruiz Cortés
Amarnath Gupta	Shazia Sadiq
Mohand-Said Hacid	Ralf Schenkel
Jan Hidders	Jialie Shen
Stefan Jablonski	Aameek Singh
Paul Johannesson	Jianwen Su
Epaminondas Kapetanios	Xiaoping Sun
Rania Khalaf	Susan Urban
Hiroyuki Kitagawa	Willem-Jan van den Heuvel
Akhil Kumar	François B. Vernadat
Frank Leymann	Maria Esther Vidal
Guohui Li	Barbara Weber
Rong Liu	Mathias Weske
Sanjay K. Madria	Andreas Wombacher
Tiziana Margaria	Jian Yang
Leo Mark	Shuigeng Zhou
Maristella Matera	

DOA-SVI 2012 Program Committee

Michele Bezzi	Schahram Dustdar
Lionel Brunie	Alois Ferscha
Marco Casassa Mont	Ching-Hsien (Robert) Hsu
David Chadwick	Karin Anna Hummel
Sadie Creese	Iulia Ion
Alfredo Cuzzocrea	Martin Jaatun
Ernesto Damiani	Ryan Ko
Yuri Demchenko	Antonio Krüger
Changyu Dong	Kai Kunze

Joe Loyall
 Paul Malone
 Marco Mamei
 Gregorio Martinez
 Rene Mayrhofer
 Mohammed Odeh
 Nick Papanikolaou
 Siani Pearson
 Christoph Reich

Chunming Rong
 Charalabos Skianis
 Anthony Sulistio
 Bhavani Thuraisingham
 Albert Zomaya
 Jianying Zhou
 Wolfgang Ziegler

ODBASE 2012 Program Committee

Karl Aberer
 Harith Alani
 Marcelo Arenas
 Sören Auer
 Denilson Barbosa
 Payam Barnaghi
 Zohra Bellahsene
 Ladjel Bellatreche
 Domenico Beneventano
 Sourav S. Bhowmick
 Omar Boucelma
 Nieves Brisaboa
 Andrea Cali
 Jorge Cardoso
 Stefano Ceri
 Sunil Choenni
 Oscar Corcho
 Francisco Couto
 Philippe Cudre-Mauroux
 Roberto De Virgilio
 Emanuele Della Valle
 Prasad Deshpande
 Dejing Dou
 Johann Eder
 Tanveer Faruque
 Walid Gaaloul
 Aldo Gangemi
 Mouzhi Ge
 Giancarlo Guizzardi
 Peter Haase
 Mohand-Said Hacid
 Harry Halpin
 Takahiro Hara

Andreas Harth
 Cornelia Hedeler
 Pascal Hitzler
 Prateek Jain
 Krzysztof Janowicz
 Matthias Klusch
 Christian Kop
 Manolis Koubarakis
 Rajasekar Krishnamurthy
 Shonali Krishnaswamy
 Steffen Lamparter
 Wookey Lee
 Sanjay Madria
 Frederick Maier
 Massimo Mecella
 Eduardo Mena
 Paolo Missier
 Anirban Mondal
 Felix Naumann
 Matteo Palmonari
 Jeff Z. Pan
 Hervé Panetto
 Paolo Papotti
 Josiane Xavier Parreira
 Dino Pedreschi
 Dimitris Plexousakis
 Ivana Podnar Zarko
 Axel Polleres
 Guilin Qi
 Christoph Quix
 Benedicto Rodriguez
 Prasan Roy
 Satya Sahoo

Claudio Sartori
Kai-Uwe Sattler
Christoph Schlieder
Michael Schrefl
Wolf Siberski
Yannis Sismanis
Srinath Srinivasa
Divesh Srivastava
Yannis Stavrakas

Letizia Tanca
Goce Trajcevski
Kunal Verma
Johanna Völke
Christian von der Weth
Wei Wang
Guo-Qiang Zhang

The Coming Age of Ambient Information

Ed Parsons

Google Inc, USA

Short Bio

Ed Parsons is the Geospatial Technologist of Google, with responsibility for evangelising Google's mission to organise the world's information using geography, and tools including Google Earth, Google Maps and Google Maps for Mobile. In his role he also maintains links with Universities, Research and Standards Organisations which are involved in the development of Geospatial Technology.

Ed is based in Google's London office, and anywhere else he can plug in his laptop.

Ed was the first Chief Technology Officer in the 200-year-old history of Ordnance Survey, and was instrumental in moving the focus of the organisation from mapping to Geographical Information.

Ed came to the Ordnance Survey from Autodesk, where he was EMEA Applications Manager for the Geographical Information Systems (GIS) Division. He earned a Masters degree in Applied Remote Sensing from Cranfield Institute of Technology and holds a Honorary Doctorate in Science from Kingston University, London.

Ed is a fellow of the Royal Geographical Society and is the author of numerous articles, professional papers and presentations to International Conferences and he developed one of the first weblogs in the Geospatial Industry at www.edparsons.com.

Ed is married with two children and lives in South West London.

Talk

"The coming age of ambient information"

With the growing adoption of Internet Protocol Version 6 (IPv6) applications and services will be able to communicate with devices attached to virtually all human-made objects, and these devices will be able to communicate with each other. The Internet of Things could become an information infrastructure a number of orders of magnitude larger than the internet today, and one which although similar may offer opportunities for radically new consumer applications. What are some of the opportunities and challenges presented by ambient information.

Inconsistency Tolerance in Ontology-Based Data Management

Maurizio Lenzerini

Università di Roma La Sapienza, Italy

Short Bio

Maurizio Lenzerini is a professor in Computer Science and Engineering at the Università di Roma La Sapienza, Italy, where he is currently leading a research group on Artificial Intelligence and Databases. His main research interests are in Knowledge Representation and Reasoning, Ontology languages, Semantic Data Integration, and Service Modeling. His recent work is mainly oriented towards the use of Knowledge Representation and Automated Reasoning principles and techniques in Information System management, and in particular in information integration and service composition. He has authored over 250 papers published in leading international journals and conferences. He has served on the editorial boards of several international journals, and on the program committees of the most prestigious conferences in the areas of interest. He is currently the Chair of the Executive Committee of the ACM Symposium of Principles of Database Systems, a Fellow of the European Coordinating Committee for Artificial Intelligence (ECCAI), a Fellow of the Association for Computing Machinery (ACM), and a member of The Academia Europaea - The Academy of Europe.

Talk

“Inconsistency tolerance in ontology-based data management”

Ontology-based data management aims at accessing, using, and maintaining a set of data sources by means of an ontology, i.e., a conceptual representation of the domain of interest in the underlying information system. Since the ontology describes the domain, and not simply the data at the sources, it frequently happens that data are inconsistent with respect to the ontology. Inconsistency tolerance is therefore a crucial feature of an in the operation of ontology-based data management systems. In this talk we first illustrate the main ideas and techniques for using an ontology to access the data layer of an information system, and then we discuss several issues related to inconsistency tolerance in ontology-based data management.

Towards Accountable Services in the Cloud

Volkmar Lotz

SAP Research, France

Short Bio

Volkmar Lotz has more than 20 years experience in industrial research on Security and Software Engineering. He is heading the Security & Trust practice of SAP Research, a group of 40+ researchers investigating into applied research and innovative security solutions for modern software platforms, networked enterprises and Future Internet applications. The Security & Trust practice defines and executes SAP's security research agenda in alignment with SAP's business strategy and global research trends.

Volkmar's current research interests include Business Process Security, Service Security, Authorisation, Security Engineering, Formal Methods and Compliance. Volkmar has published numerous scientific papers in his area of interest and is regularly serving on Programme Committees of internationally renowned conferences. He has been supervising various European projects, including large-scale integrated projects. Volkmar holds a diploma in Computer Science from the University of Kaiserslautern.

Talk

“Towards Accountable Services in the Cloud”

Accountability is a principle well suited to overcome trust concerns when operating sensitive business applications over the cloud. We argue that accountability builds upon transparency and control, and investigate in control of services in service-oriented architectures. Control needs to reach out to different layers of a SOA, both horizontally and vertically.

We introduce an aspect model for services that enables control on these layers by invasive modification of platform components and upon service orchestration. This is seen as a major constituent of an accountability framework for the cloud, which is the objective of an upcoming collaborative research project.

Process and Data: Two Sides of the Same Coin?

Manfred Reichert

University of Ulm, Germany

Short Bio

Manfred holds a PhD in Computer Science and a Diploma in Mathematics. Since January 2008 he has been appointed as full professor at Ulm University, Germany. Before, he was working in the Netherlands as associate professor at the University of Twente. There, he was also leader of the strategic research orientations on “E-health” and “Applied Science of Services”, and member of the Management Board of the Centre for Telematics and Information Technology - the largest ICT research institute in the Netherlands.

His major research interests include next generation process management technology, adaptive processes, process lifecycle management, data-driven process management, mobile processes, process model abstraction, and advanced process-aware applications (e.g., e-health, automotive engineering). Together with Peter Dadam he pioneered the work on the ADEPT process management technology and co-founded the AristaFlow GmbH. Manfred has been participating in numerous BPM research projects and made outstanding contributions in the BPMfield. His new Springer book on “Enabling Flexibility in Process-aware Information Systems” will be published in September 2012. Manfred was PC Co-chair of the BPM’08 and CoopIS’11 conferences and General Chair of the BPM’09 conference.

Talk

“Process and Data: Two Sides of the Same Coin?”

Companies increasingly adopt process management technology which offers promising perspectives for realizing flexible information systems. However, there still exist numerous process scenarios not adequately covered by contemporary information systems. One major reason for this deficiency is the insufficient understanding of the inherent relationships existing between business processes on the one side and business data on the other. Consequently, these two perspectives are not well integrated in existing process management systems.

This keynote emphasizes the need for both object- and process-awareness in future information systems, and illustrates it along several case studies. Especially, the relation between these two fundamental perspectives will be discussed,

and the role of business objects and data as drivers for both process modeling and process enactment be emphasized. In general, any business process support should consider object behavior as well as object interactions, and therefore be based on two levels of granularity. In addition, data-driven process execution and integrated user access to processes and data are needed. Besides giving insights into these fundamental properties, an advanced framework supporting them in an integrated manner will be presented and its application to complex process scenarios be shown. Overall, a holistic and generic framework integrating processes, data, and users will contribute to overcome many of the limitations of existing process management technology.

Experiences with IBM Watson Question Answering System

Guido Vetere

Center for Advanced Studies, IBM, Italy

Short Bio

Guido Vetere has attained a degree in Philosophy of Language at the University of Rome ‘Sapienza’ with a thesis in computational linguistics. He joined IBM Scientific Center in 1989, to work in many research and development projects on knowledge representation, automated reasoning, information integration, and language technologies. Since 2005, he leads the IBM Italy Center for Advanced Studies. He is member of Program Committees of various international conferences on Web Services, Ontologies, Semantic Web. Also, he represents IBM in several joint research programs and standardization activities. He is the author of many scientific publications and regularly collaborates with major Italian newspapers on scientific divulgation. He is co-founder and VP of ‘Senso Comune’, a no-profit organization for building a collaborative knowledge base of Italian.

Talk

“Experience with IBM Watson Question Answering System”

The IBM “Watson” Question Answering system, which won the Jeopardy! contest against human champions last year, is now being applied to real business. Watson integrates Natural Language Processing, Evidence-based Reasoning and Machine Learning in a way that makes it possible to deal with a great variety of information sources and to move beyond some of the most compelling constraints of current IT systems. By developing Watson, IBM Research is focusing on various topics, including language understanding, evidence evaluation, and knowledge acquisition, facing some of the fundamental problems of semantic technologies, which ultimately root in open theoretical questions about linguistic practices and the construction of human knowledge. Experiences with Watson show how it is possible to effectively use a variety of different approaches to such open questions, from exploiting ontologies and encyclopedic knowledge to learning from texts by statistical methods, within a development process that allows evaluating the best heuristics for the use case at hand.

Table of Contents – Part I

OnTheMove 2012 Keynotes

The Coming Age of Ambient Information	XIII
<i>Ed Parsons</i>	
Inconsistency Tolerance in Ontology-Based Data Management	XIV
<i>Maurizio Lenzerini</i>	
Towards Accountable Services in the Cloud	XV
<i>Volkmar Lotz</i>	
Process and Data: Two Sides of the Same Coin?	XVI
<i>Manfred Reichert</i>	
Experiences with IBM Watson Question Answering System	XVIII
<i>Guido Vetere</i>	

Cooperative Information Systems (CoopIS) 2012

CoopIS 2012 PC Co-Chairs Message	1
<i>Stefanie Rinderle-Ma, Xiaofang Zhou, and Peter Dadam</i>	
Process and Data: Two Sides of the Same Coin?	2
<i>Manfred Reichert</i>	

Business Process Design

Configurable Declare: Designing Customisable Flexible Process Models	20
<i>Dennis M.M. Schunselaar, Fabrizio Maria Maggi, Natalia Sidorova, and Wil M.P. van der Aalst</i>	
Efficacy-Aware Business Process Modeling	38
<i>Matthias Lohrmann and Manfred Reichert</i>	
Automated Resource Assignment in BPMN Models Using RACI Matrices	56
<i>Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés</i>	
Semantic Machine Learning for Business Process Content Generation . . .	74
<i>Avi Wasser and Maya Lincoln</i>	

Process Verification and Analysis

Hierarchical Process Verification in a Semi-trusted Environment	92
<i>Ganna Monakova</i>	
Intentional Fragments: Bridging the Gap between Organizational and Intentional Levels in Business Processes	110
<i>Mario Cortes-Cornax, Alexandru Matei, Emmanuel Letier, Sophie Dupuy-Chessa, and Dominique Rieu</i>	
Indexing Process Model Flow Dependencies for Similarity Search	128
<i>Ahmed Gater, Daniela Grigori, and Mokrane Bouzeghoub</i>	

Service-Oriented Architectures and Cloud

A Framework for Cost-Aware Cloud Data Management	146
<i>Verena Kantere</i>	
Event-Driven Actors for Supporting Flexibility and Scalability in Service-Based Integration Architecture	164
<i>Huy Tran and Uwe Zdun</i>	
A Conditional Lexicographic Approach for the Elicitation of QoS Preferences	182
<i>Raluca Iordache and Florica Moldoveanu</i>	
Goal-Based Composition of Stateful Services for Smart Homes	194
<i>Giuseppe De Giacomo, Claudio Di Ciccio, Paolo Felli, Yuxiao Hu, and Massimo Mecella</i>	

Security, Risk, and Prediction

Automated Risk Mitigation in Business Processes	212
<i>Raffaele Conforti, Arthur H.M. ter Hofstede, Marcello La Rosa, and Michael Adams</i>	
Aligning Service-Oriented Architectures with Security Requirements . . .	232
<i>Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini</i>	
Looking Into the Future: Using Timed Automata to Provide a Priori Advice about Timed Declarative Process Models	250
<i>Michael Westergaard and Fabrizio Maria Maggi</i>	
Planlets: Automatically Recovering Dynamic Processes in YAWL	268
<i>Andrea Marrella, Alessandro Russo, and Massimo Mecella</i>	

Discovery and Detection

Discovering Context-Aware Models for Predicting Business Process Performances	287
<i>Francesco Folino, Massimo Guarascio, and Luigi Pontieri</i>	
On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery	305
<i>Joos C.A.M. Buijs, Boudewijn F. van Dongen, and Wil M.P. van der Aalst</i>	
ECO: Event Detection from Click-through Data via Query Clustering	323
<i>Prabhu K. Angajala, Sanjay K. Madria, and Mark Linderman</i>	
Requirements-Driven Qualitative Adaptation	342
<i>Vitor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos</i>	

Collaboration

Analyzing Design Tradeoffs in Large-Scale Socio-technical Systems through Simulation of Dynamic Collaboration Patterns	362
<i>Christoph Dorn, George Edwards, and Nenad Medvidovic</i>	
Semantic and Locality Aware Consistency for Mobile Cooperative Editing	380
<i>André Pessoa Negrão, João Costa, Paulo Ferreira, and Luís Veiga</i>	
Foster an Implicit Community Based on a Newsletter Tracking System	398
<i>Tiago Lopes Ferreira and Alberto Rodrigues da Silva</i>	

Short Papers

Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA	416
<i>Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann, and David Schumm</i>	
BOINC-MR: MapReduce in a Volunteer Environment	425
<i>Fernando Costa, Luís Veiga, and Paulo Ferreira</i>	
Parallel Processing for Business Artifacts with Declarative Lifecycles ...	433
<i>Yutian Sun, Richard Hull, and Roman Vaculín</i>	

Automatically Generating and Updating User Interface Components in Process-Aware Information Systems	444
<i>Jens Kolb, Paul Hübner, and Manfred Reichert</i>	
Embedding 'Break the Glass' into Business Process Models	455
<i>Silvia von Stackelberg, Klemens Böhm, and Matthias Bracht</i>	
Author Index	465

Table of Contents – Part II

Distributed Objects and Applications and Secure Virtual Infrastructures (DOA-SVI) 2012

DOA-SVI 2012 PC Co-Chairs Message.....	469
<i>Alois Ferscha and Siani Pearson</i>	

Towards Accountable Services in the Cloud	470
<i>Volkmar Lotz and Anderson Santana de Oliveira</i>	

Privacy in the Cloud

Protecting Personal Information in Cloud Computing.....	475
<i>Miranda Mowbray and Siani Pearson</i>	

How Not to Be Seen in the Cloud: A Progressive Privacy Solution for Desktop-as-a-Service	492
<i>D. Davide Lamanna, Giorgia Lodi, and Roberto Baldoni</i>	

Privacy Preserving Cloud Transactions	511
<i>Debmalya Biswas and Krishnamurthy Vidyasankar</i>	

A Theoretically-Sound Accuracy/Privacy-Constrained Framework for Computing Privacy Preserving Data Cubes in OLAP Environments	527
<i>Alfredo Cuzzocrea and Domenico Saccá</i>	

Resource Management and Assurance

Consistency in Scalable Systems	549
<i>M.I. Ruiz-Fuertes, M.R. Pallardó-Lozoya, and F.D. Muñoz-Escóí</i>	

QoE-JVM: An Adaptive and Resource-Aware Java Runtime for Cloud Computing.....	566
<i>José Simão and Luís Veiga</i>	

DEDISbench: A Benchmark for Deduplicated Storage Systems.....	584
<i>J. Paulo, P. Reis, J. Pereira, and A. Sousa</i>	

Context, Compliance and Attack

Goal-Oriented Opportunistic Sensor Clouds	602
<i>Marc Kurz, Gerold Hölzl, and Alois Ferscha</i>	

Natural Language Processing of Rules and Regulations for Compliance in the Cloud	620
<i>Nick Papanikolaou</i>	

mOSAIC-Based Intrusion Detection Framework for Cloud Computing	628
<i>Massimo Ficco, Salvatore Venticinquè, and Beniamino Di Martino</i>	

Ontologies, DataBases, and Applications of Semantics (ODBASE) 2012

ODBASE 2012 PC Co-Chairs Message	645
<i>Sonia Bergamaschi and Isabel Cruz</i>	

Using Ontologies and Semantics

Operations over Lightweight Ontologies	646
<i>Marco A. Casanova, José A.F. de Macêdo, Eveline R. Sacramento, Ângela M.A. Pinheiro, Vânia M.P. Vidal, Karin K. Breitman, and Antonio L. Furtado</i>	

Evaluating Ontology Matchers Using Arbitrary Ontologies and Human Generated Heterogeneities	664
<i>Nafisa Afrin Chowdhury and Dejing Dou</i>	

Alignment of Ontology Design Patterns: Class As Property Value, Value Partition and Normalisation	682
<i>Bene Rodriguez-Castro, Mouzhi Ge, and Martin Hepp</i>	

Making Data Meaningful: The Business Intelligence Model and Its Formal Semantics in Description Logics	700
<i>Jennifer Horkoff, Alex Borgida, John Mylopoulos, Daniele Barone, Lei Jiang, Eric Yu, and Daniel Amyot</i>	

Applying Probabilistic Techniques to Semantic Information

Mining RDF Data for Property Axioms	718
<i>Daniel Fleischhacker, Johanna Völker, and Heiner Stuckenschmidt</i>	

A Non-intrusive Movie Recommendation System	736
<i>Tania Farinella, Sonia Bergamaschi, and Laura Po</i>	

Using Random Forests for Data Mining and Drowsy Driver Classification Using FOT Data	752
<i>Cristofer Englund, Jordanka Kovaceva, Magdalena Lindman, and John-Fredrik Grönvall</i>	

Exploiting and Querying Semantic Information

Context-Dependent Fuzzy Queries in SQLf	763
<i>Claudia Jiménez, Hernán Álvarez, and Leonid Tineo</i>	
RDF Keyword Search Query Processing via Tensor Calculus (Short paper)	780
<i>Roberto De Virgilio</i>	
Aggregating Operational Knowledge in Community Settings (Short paper)	789
<i>Srinath Srinivasa</i>	
Processing Heterogeneous RDF Events with Standing SPARQL Update Rules (Short paper)	797
<i>Mikko Rinne, Haris Abdullah, Seppo Törmä, and Esko Nuutila</i>	
Alignment-Based Querying of Linked Open Data	807
<i>Amit Krishna Joshi, Prateek Jain, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, Amit P. Sheth, and Mariana Damova</i>	
An Uncertain Data Integration System	825
<i>Naser Ayat, Hamideh Afsarmanesh, Reza Akbarinia, and Patrick Valduriez</i>	
B ⁺ -Tree Optimized for GPGPU	843
<i>Krzysztof Kaczmarski</i>	
Enhancing Trust-Based Competitive Multi Agent Systems by Certified Reputation (Short paper)	855
<i>Francesco Buccafurri, Antonello Comi, Gianluca Lax, and Domenico Rosaci</i>	
Semantics Enhancing Augmented Reality and Making Our Reality Smarter (Short paper)	863
<i>Lyndon Nixon, Jens Grubert, Gerhard Reitmayr, and James Scicluna</i>	
Liana: A Framework That Utilizes Causality to Schedule Contention Management across Networked Systems (Short Paper)	871
<i>Yousef Abushnagh, Matthew Brook, Craig Sharp, Gary Ushaw, and Graham Morgan</i>	

Managing and Storing Semantic Information

Extending Ontology-Based Databases with Behavioral Semantics	879
<i>Youness Bazhar, Chedlia Chakroun, Yamine Aït-Ameur, Ladjet Bellatreche, and Stéphane Jean</i>	

OntoDBench: Novel Benchmarking System for Ontology-Based Databases	897
<i>Stéphane Jean, Ladjel Bellatreche, Géraud Fokou, Mickaël Baron, and Selma Khouri</i>	
Using OWL and SWRL for the Semantic Analysis of XML Resources . . .	915
<i>Jesús M. Almendros-Jiménez</i>	
Building Virtual Earth Observatories Using Ontologies, Linked Geospatial Data and Knowledge Discovery Algorithms	932
<i>Manolis Koubarakis, Michael Sioutis, George Garbis, Manos Karpathiotakis, Kostis Kyzirakos, Charalampos Nikolaou, Konstantina Bereta, Stavros Vassos, Corneliu Octavian Dumitru, Daniela Espinoza-Molina, Katrín Molch, Gottfried Schwarz, and Mihai Datcu</i>	
Author Index	951

CoopIS 2012 PC Co-Chairs Message

Welcome to the proceedings of CoopIS 2012. It was the 20th conference in the CoopIS conference series and took place in Rome, Italy, in September 2012. This conference series has established itself as a major international forum for exchanging ideas and results on scientific research for practitioners in fields such as computer supported cooperative work (CSCW), middleware, Internet & Web data management, electronic commerce, business process management, agent technologies, and software architectures, to name a few. In addition, the 2012 edition of CoopIS aims at highlighting the increasing need for data- and knowledge-intensive processes. As in previous years, CoopIS'12 was again part of a joint event with other conferences, in the context of the OTM ("OnTheMove") federated conferences, covering different aspects of distributed information systems.

Thanks to the many high-quality submissions we were able to put a strong program together. The selection process was very competitive which resulted in quite a few good papers to be rejected. Each paper received at least three, in a number of cases even four, independent peer reviews. From the 100 submissions we were able to accept only 22 papers as full papers and eight papers as short papers.

We are very grateful to the reviewers who worked hard to meet the tight deadline. We thank also the staff of the OTM secretariat, especially Jan Demey, Daniel Meersman, and Carlos Madariaga and the OTM General Chairs Robert Meersman, Tharam Dillon, and Pilar Herrero for their support.

July 2012

Stefanie Rinderle-Ma
Xiaofang Zhou
Peter Dadam

Process and Data: Two Sides of the Same Coin?

Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Germany
`manfred.reichert@uni-ulm.de`

Abstract. Companies increasingly adopt process management technology which offers promising perspectives for realizing flexible information systems. However, there still exist numerous process scenarios not adequately covered by contemporary information systems. One major reason for this deficiency is the insufficient understanding of the inherent relationships existing between business processes on the one side and business data on the other. Consequently, these two perspectives are not well integrated in many existing process management systems. This paper emphasizes the need for both object- and process-awareness in future information systems, and illustrates it along several examples. Especially, the relation between these two fundamental perspectives will be discussed, and the role of business objects and data as drivers for both process modeling and process enactment be emphasized. In general, any business process support should consider object behavior as well as object interactions, and therefore be based on two levels of granularity. In addition, data-driven process execution and integrated user access to processes and data are needed. Besides giving insights into these fundamental characteristics, an advanced framework supporting them in an integrated manner will be presented and its application to real-world process scenarios be shown. Overall, a holistic and generic framework integrating processes, data, and users will contribute to overcome many of the limitations of existing process management technology.

1 Introduction

Despite the widespread adoption of process management systems [1] there exist many business processes not adequately supported by these systems. In this context, different authors state that many deficiencies of contemporary process management systems (PrMS) can be traced back to the missing integration of processes and data [2–11]. Although processes and data seem to be closely related, a unified understanding of the inherent relationships existing between them is still missing.

In the PHILharmonicFlows project, we analyzed numerous business processes from different domains which require a tight data integration [12–15]. We learned that many of these processes are *data-driven* and that their support requires *object-awareness*; i.e., the progress of these processes depends on the processing of certain business data represented through *business objects*. Objects comprise a set of *object attributes* and are *inter-related*. In this context, business processes

coordinate the processing of business objects among different users enabling them to cooperate and communicate with each other. Most existing PrMS, however, mainly focus on business functions and their flow of control, whereas business objects are "unknown" to them. As a consequence, most PrMS only cover simple data elements needed for control flow routing and for supplying input parameters of activities with values. In turn, business objects are usually stored in external databases and are outside the control of the PrMS. Hence, existing PrMS are unable to adequately support *object-aware processes* [16].

This paper shows that process and data are actually two sides of the same coin. Section 2 introduces the main characteristics of data-driven and object-aware processes, which we gathered in a number of case studies [12, 13, 17] (see [18] for details about the research methodology applied). Following this, Section 3 sketches core components of our PHILharmonicFlows framework, which enables comprehensive support of data-driven and object-aware processes. Section 4 discusses related work and Section 5 concludes the paper with a summary.

2 Data-Driven and Object-Aware Processes

We first discuss fundamental characteristics of data-driven and object-aware processes, we derived from a more detailed property list related to process modeling, execution, and monitoring. The respective properties were discovered in an extensive analysis of processes currently not adequately supported by process management technology [12–14, 16]. To ensure that the processes we considered are not "self-made" examples, but constitute real-world processes of high practical relevance, we further analyzed processes implemented in existing business applications. Further, we have deep insights into the code and process logic of these applications. To justify our findings, we complemented our process analyses by an extensive literature study ensuring relevance and completeness.

2.1 Application Example

We discuss the characteristics of data-driven and object-aware processes along a simple job recruitment scenario (cf. Fig. 1).

Example 1 (Recruitment). *In recruitment, applicants may apply for job vacancies via an Internet online form. Once an application has been submitted, the responsible personnel officer in the human resource department is notified. The overall process goal is to decide which applicant shall get the job. If an application is ineligible the applicant is immediately rejected. Otherwise, personnel officers may request internal reviews for each applicant. In this context, the concrete number of reviews may differ from application to application. Corresponding review forms have to be filled by employees from functional divisions. They make a proposal on how to proceed; i.e., they indicate whether the applicant shall be invited for an interview or be rejected. In the former case an additional appraisal is needed. After the employee has*

filled the *review form* she submits it back to the *personnel officer*. In the meanwhile, additional *applications* might have arrived; i.e., *reviews* relating to the same or to different *applications* may be requested or submitted at different points in time. The processing of the *application*, however, proceeds while corresponding *reviews* are created; e.g., the *personnel officer* may check the *CV* and study the *cover letter* of the *application*. Based on the incoming *reviews* he makes his *decision* on the *application* or initiates further steps (e.g., *interviews* or additional *reviews*). Finally, he does not have to wait for the arrival of all *reviews*; e.g., if a particular *employee* suggests hiring the *applicant* he can immediately follow this recommendation.

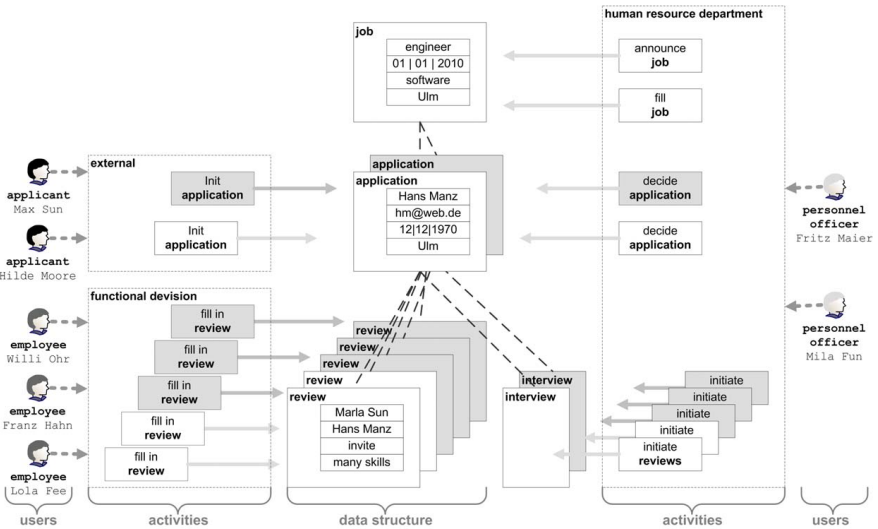


Fig. 1. Example of a recruitment process from the human resource domain

2.2 Basic Characteristics

Basically, data must be manageable in terms of *object types* comprising *object attributes* and *relations* to other object types (cf. Fig. 2a). At run-time, the different object types comprise a varying number of inter-related object instances, whereby the concrete instance number should be restrictable by lower and upper cardinality bounds (cf. Fig. 2b). For each application, for example, at least one and at most five reviews must be initiated. While for one application two reviews are available, another one may comprise three reviews (cf. Fig. 1).

In accordance to data modeling, the modeling and execution of processes can be based on two levels of granularity: *object behavior* and *object interactions*.

Object Behavior. To cover the processing of individual object instances, the first level of process granularity concerns *object behavior*. More precisely, for

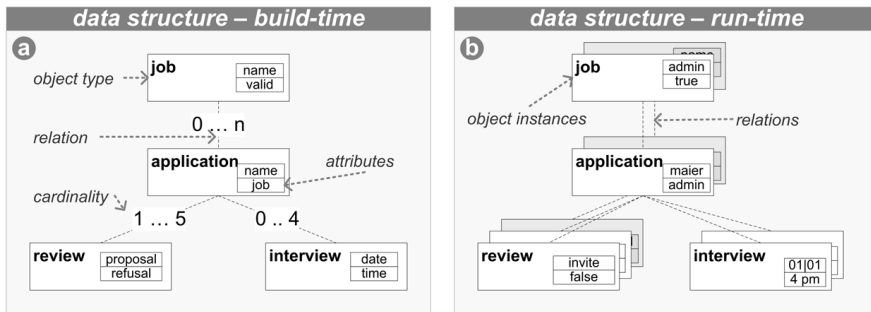


Fig. 2. Data structure at build-time and at run-time

each object type a separate process definition should be provided (cf. Fig. 3a), which can be used for coordinating the processing of an individual object instance among different users. In addition, it should be possible to determine in which order and by whom the attributes of a particular object instance have to be (mandatorily) written, and what valid attribute values are. At run-time, the creation of an object instance is directly coupled with the creation of its corresponding process instance. In this context, it is important to ensure that mandatorily required data is provided during process execution. For this reason, object behavior should be defined in terms of *data conditions* rather than based on black-box activities.

Example 2 (Object behavior). *For requesting a review the responsible personnel officer has to mandatorily provide values for object attributes return date and questionnaire. Following this, the employee being responsible for the review has to mandatorily assign a value to object attribute proposal.*

Object Interactions. Since related object instances may be created or deleted at arbitrary points in time, a complex data structure emerges, which dynamically evolves depending on the types and number of created object instances. In addition, individual object instances (of the same type) may be in different processing states at a certain point in time.

Taking the behavior of individual object instances into account, we obtain a *complex process structure* in correspondence to the given data structure (cf. Fig. 3a). In this context, the second level of process granularity comprises the *interactions* that take place between different object instances. More precisely, it must be possible to execute individual process instances (of which each corresponds to a particular object instance) in a loosely coupled manner, i.e., concurrently to each other and synchronizing their execution where needed. First, it should be possible to make the creation of a particular object instance dependent on the progress of related object instances (*creation dependency*). Second, several object instances of the same object type may be related to one and the same object instance. Hence, it should be possible to aggregate information; amongst

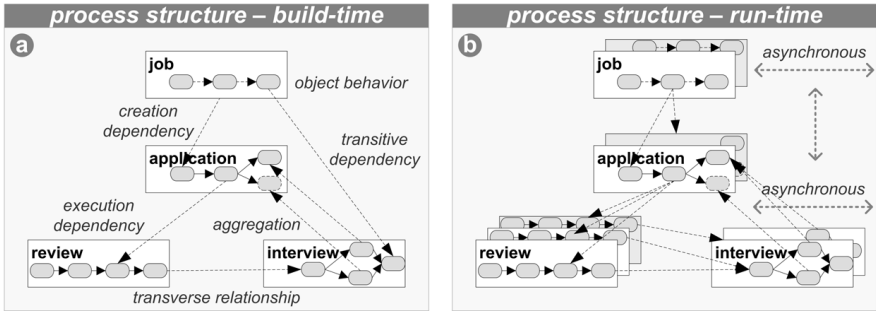


Fig. 3. Process structure at build-time and at run-time

others, this requires the aggregation of attribute values from related object instances (*aggregation*). Third, the executions of different process instances may be mutually dependent; i.e., whether or not an object instance can be further processed may depend on the processing progress of other object instances (*execution dependency*). In this context, interactions must also consider *transitive dependencies* (e.g., reviews depend on the respective job offer) as well as *transverse* ones (e.g., the creation of an interview may depend on the proposal made in a review) between object instances (cf. Fig. 3).

Example 3 (Object interactions). *A personnel officer must not initiate any review as long as the corresponding application has not been finally submitted by the applicant (creation dependency). Furthermore, individual review process instances are executed concurrently to each other as well as concurrently to the application process instances; e.g., the personnel officer may read and change the application, while the corresponding reviews are processed. Further, reviews belonging to a particular application can be initiated and submitted at different points in time. Besides this, a personnel officer should be able to access information about submitted reviews (aggregative information); i.e., if an employee submits her review recommending to invite the applicant for an interview, the personnel officer needs this information immediately. Opposed to this, when proposing rejection of the applicant, the personnel officer should only be informed after all initiated reviews have been submitted. Finally, if the personnel officer decides to hire one of the applicants, all others must be rejected (execution dependency). These dependencies do not necessarily coincide with the object relations. As example consider reviews and interviews corresponding to the same application; i.e., an interview may only be conducted if an employee proposes to invite the applicant during the execution of a review process instance.*

Data-Driven Execution. In order to proceed with the processing of a particular object instance, usually, in a given state certain *attribute values are mandatorily required*. Thus, object attribute values reflect the progress of the corresponding process instance. In particular, the activation of an activity does

not directly depend on the completion of other activities, but on the values set for object attributes. More precisely, *mandatory activities* enforce the setting of certain object attribute values in order to progress with the process. If required data is already available, however, mandatory activities can be *automatically skipped* when being activated. In principle, it should be possible to *set respective attributes also up front*; i.e., before the mandatory activity normally writing this attribute becomes activated. However, users should be allowed to *re-execute a particular activity*, even if all mandatory object attributes have been already set. For this purpose, data-driven execution must be combined with *explicit user commitments* (i.e., activity-centred aspects). Finally, the execution of a mandatory activity may also depend on available attribute values of related object instances. Thus, coordination of process instances must be supported in a data-driven way as well.

Example 4 (Data-driven execution). *During a review request the personnel officer must mandatorily set a return date. If a value for the latter is available, a mandatory activity for filling in the review form is assigned to the responsible employee. Here, in turn, a value for attribute proposal is mandatorily required. However, even if the personnel officer has not completed his review request yet (i.e., no value for attribute return data is available), the employee may optionally edit certain attributes of the review (e.g., the proposal). If a value of attribute proposal is already available when the personnel officer finishes the request, the mandatory activity for providing the review is automatically skipped. Opposed to this, an employee may change his proposal arbitrarily often until he explicitly agrees to submit the review to the personnel officer. Finally, the personnel officer makes his decision (e.g., whether to reject or to accept the applicant) based on the incoming reviews.*

Variable Activity Granularity. For creating object instances and changing object attribute values, *form-based activities* are required. Respective user forms comprise *input fields* (e.g., text-fields or checkboxes) for writing and *data fields* for reading selected attributes of object instances. In this context, however, different users may prefer different work practices. In particular, using *instance-specific activities* (cf. Fig. 5a), all input fields and data fields refer to attributes of one particular object instance, whereas *context-sensitive activities* (cf. Fig. 5b) comprise fields referring to different, but related object instances (of potentially different type). When initiating a review, for example, it is additionally possible to edit the attribute values of the corresponding application. Finally, *batch activities* involve several object instances of the same type (cf. Fig. 5c). Here, the values of the different input fields are assigned to all involved object instances in one go. This enables a personnel officer, for example, to reject a number of application in one go. Depending on their preference, users should be able to freely choose the most suitable activity type for achieving a particular goal. In addition to form-based activities, it must be possible to integrate *black-box activities*. The latter enable complex computations as well as the integration of advanced functionalities (e.g., provided by web services).

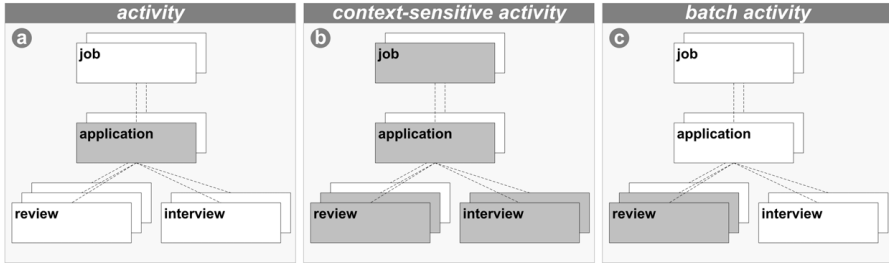


Fig. 4. Different kinds of activities

Moreover, whether certain object attributes are mandatory when processing a particular activity might depend on other object attribute values as well; i.e., when filling a form certain attributes might become mandatory on-the-fly. Such *control flows being specific to a particular form* should be also considered.

Example 5 (Activity Execution). *When an employee fills in a review, additional information about the corresponding application should be provided; i.e., attributes belonging to the application for which the review is requested. For filling in the review form, a value for attribute proposal has to be assigned. If the employee proposes to invite the applicant, additional object attributes will become mandatory; e.g., then he has to set attribute appraisal as well. This is not required if he assigns value reject to attribute proposal. Further, when a personnel officer edits an application, all corresponding reviews should be visible. Finally, as soon as an applicant is hired for a job, for all other applications value reject should be assignable to attribute decision by filling one form.*

Integrated Access. To proceed with the control flow, *mandatory activities* must be executed by responsible users in order to provide required attribute values. Other attribute values, however, may be optionally set. Moreover, users who are usually not involved in process execution should be allowed to optionally execute selected activities. In addition to a *process-oriented view* (e.g. work lists), a *data-oriented view* should be provided enabling users to access and manage data at any point in time. For this purpose, we need to define permissions for creating and deleting object instances as well as for reading/writing their attributes. However, attribute changes contradicting to specified object behavior should be prevented. Which attributes may be (mandatorily or optionally) written or read by a particular form-based activity not only depends on the user invoking this activity, but also on the progress of the corresponding process instances. While certain users must execute an activity mandatorily in the context of a particular object instance, others might be authorized to optionally execute this activity; i.e., *mandatory and optional permissions* should be distinguishable. Moreover, for object-aware processes, the selection of potential actors should not only depend on the activity itself, but also on the object instances processed by this

activity. In this context, it is important to take the relationships between users and object instances into account.

Example 6 (Integrated Access). *A personnel officer may only decide on **applications** for which the name of the **applicants** starts with a letter between 'A' and 'L', while another officer may decide on **applicants** whose name starts with a letter between 'M' and 'Z'. An **employee** must mandatorily write attribute **proposal** when filling in a **review**. However, her **manager** may optionally set this attribute as well. The mandatory activity for filling the **review** form, in turn, should be only assigned to the **employee**. After submitting her **review**, the **employee** still may change her **comment**. In this context, it must be ensured that the **employee** can only access **reviews** she submitted before. However, attribute **proposal**, in turn, must not be changed anymore. The **personnel officer** might have already performed the proposed action.*

3 A Framework Enabling Data-Driven and Object-Aware Processes

In the PHILharmonicFlows project, we developed a framework that enables the characteristic properties of data-driven and object-aware processes and hence contributes to overcome many of the limitations of existing process management technology. The PHILharmonicFlows framework will be described in this section using another illustrating example. In particular, the framework provides advanced concepts and components enabling comprehensive support for data-driven and object-aware processes.

3.1 Illustrating Example

We first present another example of an object-aware process along which we will illustrate basic concepts of the PHILharmonicFlows framework.

The selected scenario deals with proposing extension courses at a university; i.e., courses for professionals that aim at refreshing and updating their knowledge in a certain area of expertise. In order to propose a new extension course, the course coordinator must create a project describing it. The latter must be approved by the faculty coordinator as well as the extension course committee.

Example 7 (Extension course proposal). *The course coordinator creates a new **extension course project** using a form. In this context, he must provide details about the course, like **name**, **start date**, **duration**, and **description**. Following this, **professors** may start creating the **lectures** for the extension course. Each **lecture**, in turn, must have detailed **study plan items**, which describe the activities of the lecture. For each lecture, there may be some (external) **invited speakers**. The latter either may **accept** or **reject** the **invitation**. After receiving the responses for the **invitations** and creating the **lectures**, the **coordinator** may request the approval of the **extension course project**. First, it must be approved by the **faculty director**. If he*

wants to *reject* it, he must provide a *reason* for his decision and the course must not take place. Otherwise, the project is sent to the *extension course committee* for evaluation. If there are more *rejections* than *approvals*, the *extension course project* is *rejected*. Otherwise, it is *approved* and hence may take place.

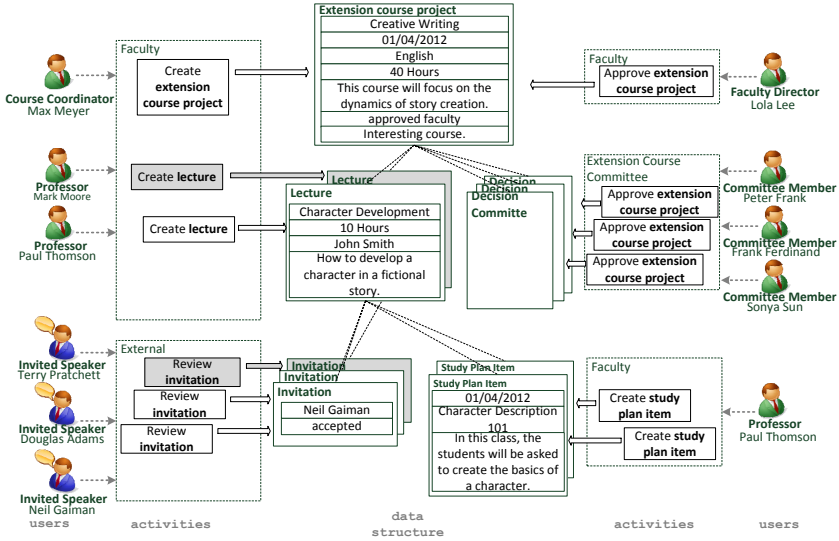


Fig. 5. Extensions course proposal

3.2 Selected Components of the PHILharmonicFlows Framework

The PHILharmonicFlows framework supports object-aware processes focusing on the processing of business data and business objects respectively. More precisely, *object-awareness* means that the overall process model is structured and divided according to the *object types* involved. In turn, these object types are organized in a data model and may be related to other object types. Moreover, for each object type, a separate process type defining its *object behavior* exists. At run-time, each object type then may comprise a varying number of object instances. Since the creation of an object instance is directly coupled with the creation of a corresponding process instance, a complex *process structure* emerges. Thereby, process instances referring to object instances of the same type are executed asynchronously to each other as well as asynchronously to process instances related to objects of different types. However, their execution may have to be synchronized at certain points in time. Overall, PHILharmonicFlows differentiates between *micro* and *macro processes* which allow capturing both *object behavior* and *object interactions*. Furthermore, the execution of micro and macro processes is data-driven and integrated access to processes and data objects is enabled. Finally, different kinds of activity granularities are supported.

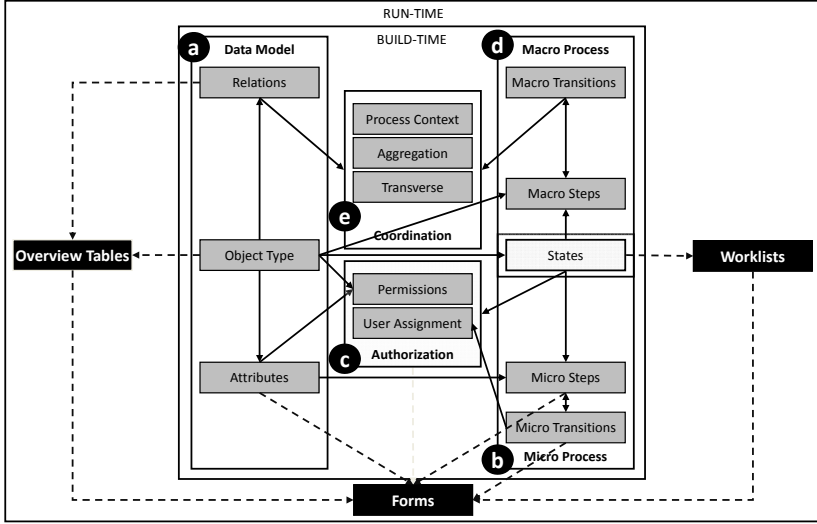


Fig. 6. Overview of the PHILharmonicFlows framework

Data Model. A *data model* defines the object types as well as their corresponding attributes and relations with cardinalities (cf. Fig. 6a).

Example 8 (Data structure). Fig. 7a illustrates the data model relating to our example from Section 3.1. Object types *lecture* and *decision committee* refer to object type extension course project. In turn, object types *invitation* and *study plan item* refer to *lecture*. At run-time, these relations allow for a varying number of inter-related object instances whose processing must be coordinated. Further, cardinality constraints restrict the minimum and maximum number of instances of an object type that may reference the same higher-level object instance. Fig. 7b shows a corresponding data structure at run-time.

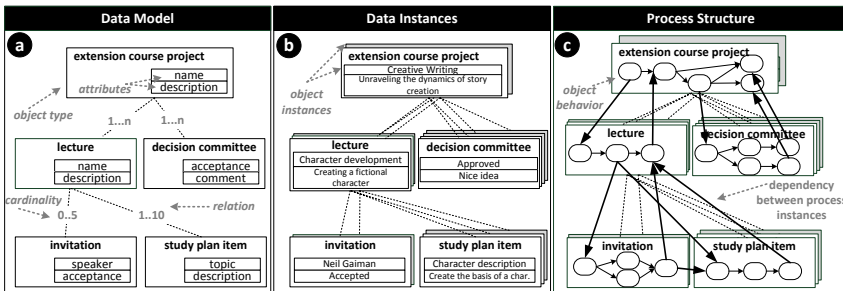


Fig. 7. Data structure (data model and instances) and process structure

Micro Processes. To express *object behavior*, for each object type of a data model, a *micro process type* must be defined (cf. Fig. 6b). At run-time, the creation of object instances is then directly coupled with the creation of a corresponding *micro process instances*. The latter coordinates the processing of the object instance among different users and specifies the order in which object attributes may be written. For this purpose, a micro process type comprises a number of *micro step types* (cf. Fig 6b), of which each refers to one specific object attribute and describes an atomic action for writing it. At run-time, a micro step is reached if a value is set for the corresponding attribute; i.e., a *data-driven execution* is enabled. Micro step types may be inter-connected using *micro transition types* in order to express their default execution order. When using form-based activities, micro transitions define the order in which the input fields of the respective form shall be filled (i.e., the internal processing logic of the form). Finally, to coordinate the processing of individual object instances among different users, several micro step types can be grouped into *state types*. At the instance level, a state may only be left if the values for all attributes associated with the micro steps of the respective state type are set.

Example 9 (Micro process type). Fig. 8a shows the micro process type of object type *extension course project*. While the *extension course project* is in state *under creation*, the *course coordinator* may set the attributes to which the corresponding micro step types refer (e.g., *name*, *start_date*, or *description*). Following this, a user decision is made in state *under approval faculty*; i.e., the *faculty director* either approves or rejects the *extension course project*. If the value of attribute *decision_faculty* is *rejected*, a value for attribute *reason_rejection* is required.

User Authorization. PHILharmonicFlows provides advanced support for user authorization while enabling an integrated access to process and data (cf. Fig. 6c). User roles are associated with the different states of a micro process type.

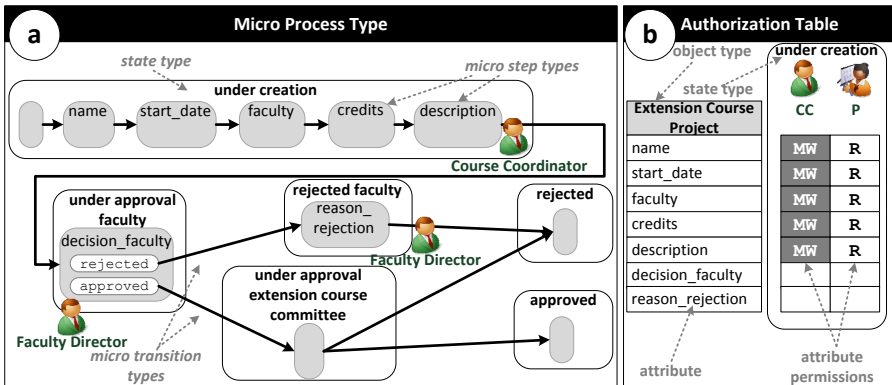


Fig. 8. Micro process type and authorization table for state “under creation”

At run-time, users owning the respective role then must set required attribute values as indicated by the micro steps corresponding to the respective state; i.e., a mandatory activity (i.e., a user form) is created and assigned to the user's work list. To enable optional activities, in addition, PHILharmonicFlows generates an *authorization table* for each object type. More precisely, the framework allows granting different permissions for reading and writing attribute values as well as for creating and deleting object instances to different user roles (cf. Fig. 6d). Furthermore, permissions may vary depending on the state of an object instance. The framework ensures that each user who must execute a mandatory activity also owns corresponding write permissions; i.e., data and process authorization are compliant with each other. The initially generated authorization table may be further adjusted by assigning optional permissions to other users. In this context, we differentiate between *mandatory* and *optional write* permissions.

Attributes, permissions, and the described micro process logic also provide the basis for automatically generating *user forms* at run-time. In particular, when taking the currently activated state of the micro process instance into account, the authorization table specifies which input fields can be read or written by the respective user in this state. Hence, any change directly affecting directly the forms will be transparent to the end-user; i.e., the forms do not need to be manually updated as in existing process-aware information systems.

Example 10 (Authorization table). *In Fig. 8b, in state under creation of micro process type extension course project, the course coordinator (CC) has mandatory write (MW) permission for attributes name, start_date, faculty, credits, and description. In turn, a professor (P) is authorized to read (R) these attributes in the respective state.*

Macro Process Level. At run-time, object instances of the same and different types may be created or deleted at arbitrary points in time; i.e., the data structure dynamically evolves depending on the types and number of created object instances. In particular, whether subsequent states of micro process instances can be reached may depend on other micro process instances as well; i.e., the processing of an object instance may depend on the processing of a variable number of instances of a related object type. Taking these dependencies among objects into account, a complex *process structure* results (cf. Fig. 7c). To enable proper interaction among the micro process instances, a *coordination* mechanism is required to specify the interaction points of the involved processes. For this purpose, PHILharmonicFlows automatically derives a state-based view for each micro process type. This view is then used for modeling so-called *macro process types* defining the respective *object interactions* (cf. Fig. 6d). The latter hides the complexity of emerging process structure from users. Each *macro process type* (cf. Fig. 9) consists of *macro step types* and *macro transitions types* connecting them. As opposed to traditional process modeling approaches, where process steps are defined in terms of black-box activities, a macro step type always refers to an object type together with a corresponding state type; i.e., the latter serve as interface between micro and macro process types.

The activation of a particular macro state might depend on instances of different micro process types. To express this, a respective *macro input type* has to be defined for each macro step types. The latter can be connected to several incoming macro transitions. At run-time, a macro step is enabled if at least one of its macro inputs becomes activated.

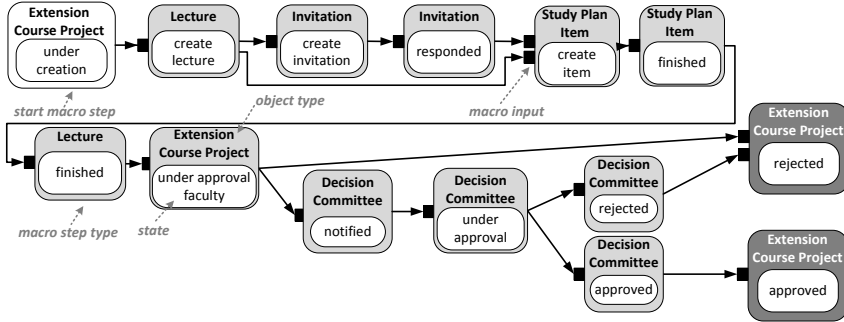


Fig. 9. Example of a Macro Process Type

To take the dynamically evolving number of object instances as well as the asynchronous execution of corresponding micro process instances into account, for each macro transition a corresponding *coordination component* needs to be defined (cf. Fig. 6e). For this purpose, PHILharmonicFlows utilizes object relations from the data model; i.e., takes the relations between the object type of a source macro step type and the one of a target macro step type into account. For this purpose, the framework organizes the data model into different *data levels*. All object types not referring to any other object type are placed on the top level. Generally, any other object type is always assigned to a lower data level as the object type it references. Based on this, PHILharmonicFlows can automatically classify the macro transitions either as *top-down* or as *bottom-up* (cf. Fig. 10a). If the object types of the source and sink macro step types refer to a common higher-level object type, the macro transition is categorized as *transverse*.

For each of these categories of macro transition type, a particular coordination component is required. A *top-down transition* characterizes the interaction from an upper-level object type to a lower-level one. Here, the execution of a varying number of micro process instances depends on one higher-level micro process instance. In this context, a so-called *process context type* must be assigned to the respective macro transition type. Due to lack of space, we do not go into details. We also do not discuss transverse macro transition types here. In turn, a *bottom-up transition* characterizes an interaction from a lower-level object type to an upper-level one. In this case, the execution of one higher-level micro process instance depends on the execution of several lower-level micro process instances of the same type. For this reason, each bottom-up transition requires an *aggregation component* for coordination. To manage the total number of lower-level micro process instances related to the dependent upper-level micro process

instance, PHILharmonicFlows provides *counters*; i.e., the latter permit to control the number of micro process instances that have reached the respective state as well as the ones that have not yet reached the state and the ones that have skipped the same state. These counters can be used for defining aggregation conditions enabling the higher-level micro process instance to activate the state.

Example 11 (Aggregation component). *Fig. 10b shows an aggregation condition ($\#IN < \#ALL/2$) expressing that the *extension course project* will be *approved* if more than half of the *committee members* approve the project. In this example, there are three micro process instances of *decision committee* related to one instance of *extension course project*. The counter of this example indicates that two of the running instances of *decision committee* have already reached state *approved* ($\#IN = 2$), while one instance has not yet reached this state ($\#BEFORE=1$). In this case, the condition is already fulfilled and the upper-level micro process instance may continue its execution.*

4 Related Work

In [16], we have shown why existing imperative, declarative, and data-driven (i.e., Case Handling [2, 19, 20]) process support paradigms are unable to adequately support object-aware processes. However, to enable consistency between process and object states, extensions of these approaches based on object life cycles have been proposed. These extensions include object life cycle compliance [21], object-centric process models [8, 9], business artifacts [4, 22], data-driven process coordination [5, 23], and product-based workflows [6, 24]. However, none of these approaches explicitly maps states to object attribute values. Consequently, if certain pre-conditions cannot be met during run-time, it is not possible to dynamically react to this. In addition, generic form logic is not provided in a

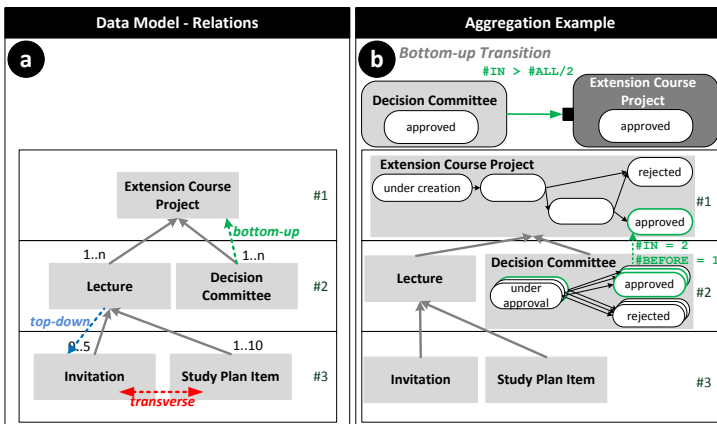


Fig. 10. a) Kinds of relations between object Types; b) Aggregation example

flexible way; i.e., there is no automatic generation of forms taking the individual attribute permissions of a user as well as the progress of the corresponding process into account. Finally, opposed to these approaches, PHILharmonicFlows captures the internal logic of an activity as well.

As illustrated in Fig. 11, each characteristic from Section 2 is addressed by at least one existing approach. Although the mentioned approaches have deficiencies (see footnotes in Fig. 11), they can be considered as pioneering work towards data-driven and object-aware process support. As opposed to PHILharmonicFlows, however, none of them covers all characteristics in a comprehensive and integrated way. Also note that Fig. 11 does not make a difference between process modeling and process execution. Though some approaches (e.g., the business artifacts framework [4]) provide rich capabilities for process modeling, they do not cover run-time issues (or at least do not treat them explicitly).

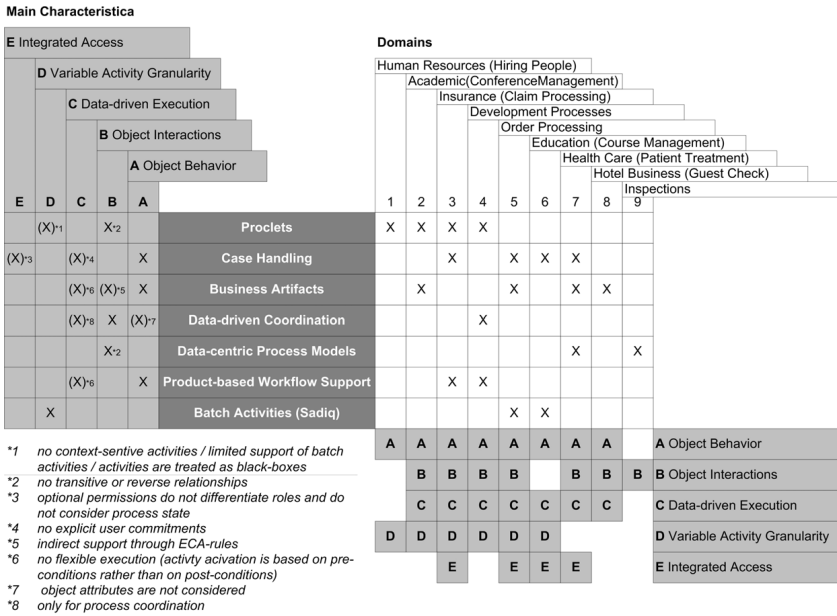


Fig. 11. Evaluation of existing approaches

Interestingly, existing approaches partially consider similar scenarios, while addressing different characteristics (see the grey boxes on the bottom of Fig. 11). For example, *order processing* was taken as illustrating scenario by Case Handling [2], Batch Activities [25], and Business Artifacts [4]. Case Handling addresses the need for enabling object behavior, data-driven execution, and integrated access. In turn, Business Artifacts consider data-driven execution, object behavior and object interactions. Finally, [25] describes the need for executing several activities in one go (i.e., the execution of batch-activities). Hence, the integrated support of all characteristics described is urgently needed to adequately cope with *order processes*.

5 Summary

Through elaborating the main characteristics of object-aware processes, this paper has shown that process and data more or less constitute two sides of the same coin, which must be tightly integrated. Hence, data-driven and object-aware process support will provide an important contribution towards the realization of a more flexible process management for which daily work can be accomplished in a more natural way than in traditional process-aware information systems. As illustrated in Fig. 12, a comprehensive integration of processes and data entails three major benefits:

1. Flexible execution of unstructured, knowledge-intensive processes.
2. Integrated view on processes, data, and functions to users.
3. Generic business functions, e.g., automatically generated form-based activities.

PHILharmonicFlows offers a comprehensive solution framework to adequately support data-driven and object-aware processes. In particular, this framework supports the definition of data and process in separate, but well integrated models. So far, PHILharmonicFlows has addressed process modeling, execution and monitoring, and it provides generic functions for the model-driven generation of end user components (e.g., form-based activities). Furthermore, PHILharmonicFlows considers all components of the underlying data structure; i.e., objects, relations and attributes. For this purpose, it enables the modeling of processes at different levels of granularity. In particular, it combines object behavior based on states with data-driven process execution. Further, it provides advanced support for process coordination as well as for the integrated access to business processes, business functions and business data. We believe that the described framework offers promising perspectives for overcoming many of the limitations of contemporary PrMS.

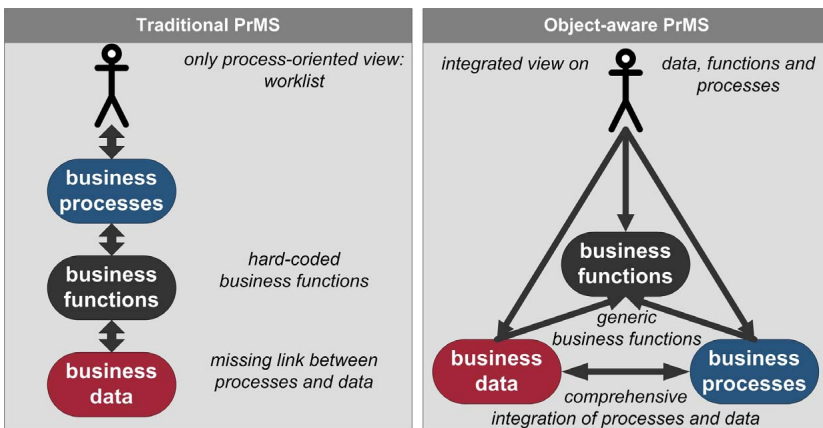


Fig. 12. Object-aware Process Management

References

1. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer (2012)
2. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A new Paradigm for Business Process Support. *DKE* 53, 129–162 (2005)
3. van der Aalst, W.M.P., Barthelmeß, P., Ellis, C.A., Wainer, J.: Workflow Modeling using Proclets. In: Scheuermann, P., Etzion, O. (eds.) *CoopIS 2000*. LNCS, vol. 1901, pp. 198–209. Springer, Heidelberg (2000)
4. Bhattacharya, K., Hull, R., Su, J.: A Data-Centric Design Methodology for Business Processes, pp. 503–531. IGI Global (2009)
5. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) *OTM 2007, Part I*. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
6. Reijers, H.A., Liman, S., van der Aalst, W.M.P.: Product-Based Workflow Design. *Management Information Systems* 20, 229–262 (2003)
7. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based Workflow Support. *Information Systems* 36, 517–535 (2011)
8. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: Transforming Object-Oriented Models to Process-Oriented Models. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) *BPM Workshops 2007*. LNCS, vol. 4928, pp. 132–143. Springer, Heidelberg (2008)
9. Redding, G.M., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, 1–11 (2009)
10. Künzle, V., Reichert, M.: Striving for object-aware process support: How existing approaches fit together. In: 1st Int'l Symposium on Data-driven Process Discovery and Analysis, *SIMPDA 2011* (2011)
11. Rinderle, S., Reichert, M.: Data-Driven Process Control and Exception Handling in Process Management Systems. In: Martinez, F.H., Pohl, K. (eds.) *CAISE 2006*. LNCS, vol. 4001, pp. 273–287. Springer, Heidelberg (2006)
12. Künzle, V., Reichert, M.: Towards Object-Aware Process Management Systems: Issues, Challenges, Benefits. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) *BPMS 2009 and EMMSAD 2009*. LNBIP, vol. 29, pp. 197–210. Springer, Heidelberg (2009)
13. Künzle, V., Reichert, M.: Integrating Users in Object-Aware Process Management Systems: Issues and Challenges. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) *BPM 2009 Workshops*. LNBIP, vol. 43, pp. 29–41. Springer, Heidelberg (2010)
14. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. *Journal of Software Maintenance and Evolution: Research and Practice* 23, 205–244 (2011)
15. Künzle, V., Reichert, M.: A Modeling Paradigm for Integrating Processes and Data at the Micro Level. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) *BPMS 2011 and EMMSAD 2011*. LNBIP, vol. 81, pp. 201–215. Springer, Heidelberg (2011)
16. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. *International Journal of Information System Modeling and Design (IJISMD)* 2, 19–46 (2011)
17. Chiao, C.M., Künzle, V., Reichert, M.: Towards object-aware process support in healthcare information systems. In: 4th International Conference on eHealth, Telemedicine, and Social Medicine, *eTELEMED 2012* (2012)

18. Künzle, V., Reichert, M.: PHILharmonicFlows: Research and Design Methodology. Technical Report UIB-2011-05, University of Ulm, Ulm, Germany (2011)
19. Weber, B., Mutschler, B., Reichert, M.: Investigating the effort of using business process management technology: Results from a controlled experiment. *Science of Computer Programming* 75, 292–310 (2010)
20. Guenther, C.W., Reichert, M., van der Aalst, W.M.: Supporting flexible processes with adaptive workflow and case handling. In: Proc. WETICE 2008, 3rd IEEE Workshop on Agile Cooperative Process-aware Information Systems (ProGility 2008), pp. 229–234. IEEE Computer Society Press (2008)
21. Küster, J.M., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
22. Gerede, C.E., Su, J.: Specification and Verification of Artifact Behaviors in Business Process Models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
23. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures. In: Bellahsene, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 48–63. Springer, Heidelberg (2008)
24. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product Based Workflow Support: Dynamic Workflow Execution. In: Bellahsene, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 571–574. Springer, Heidelberg (2008)
25. Sadiq, S.W., Orłowska, M.E., Sadiq, W., Schulz, K.: When workflows will not deliver: The case of contradicting work practice. In: Proc. BIS 2005 (2005)

Configurable Declare: Designing Customisable Flexible Process Models^{*,**}

Dennis M.M. Schunselaar, Fabrizio Maria Maggi,
Natalia Sidorova, and Wil M.P. van der Aalst

Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar,f.m.maggi,n.sidorova,w.m.p.v.d.aalst}@tue.nl

Abstract. Declarative languages are becoming more popular for modelling business processes with a high degree of variability. Unlike procedural languages, where the models define what is to be done, a declarative model specifies what behaviour is not allowed, using constraints on process events. In this paper, we study how to support configurability in such a declarative setting. We take *Declare* as an example of a declarative process modelling language and introduce *Configurable Declare*. Configurability is achieved by using configuration options for event hiding and constraint omission. We illustrate our approach using a case study, based on process models of ten Dutch municipalities. A Configurable *Declare* model is constructed supporting the variations within these municipalities.

Keywords: business process modelling, configurable process models, declarative process models, Declare.

1 Introduction

Process-aware information systems [4], such as workflow management systems, case-handling systems and enterprise information systems, are used in many branches of industry and governmental organisations. *Process models* form the heart of such systems since they define the flow of task executions. Traditionally, the languages used for process modelling are *procedural* languages, since they are very appropriate for describing well-structured processes with a predefined flow. At the same time, procedural models become very complex for environments with high variability, since every possible execution path has to be encoded in the model. In the most extreme cases, like specifications of some medical

* This research has been carried out as part of the Configurable Services for Local Governments (*CoSeLoG*) project (<http://www.win.tue.nl/coselog/>).

** This research has been carried out as a part of the Poseidon project at Thales under the responsibilities of the Embedded Systems Institute (ESI). The project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

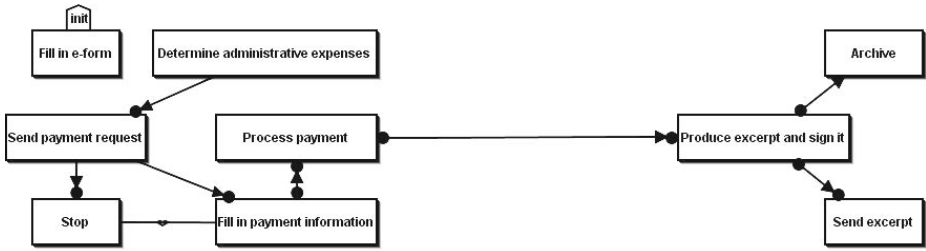


Fig. 1. Example of Declare model describing the process for requesting an excerpt from the civil registration (Mun. A)

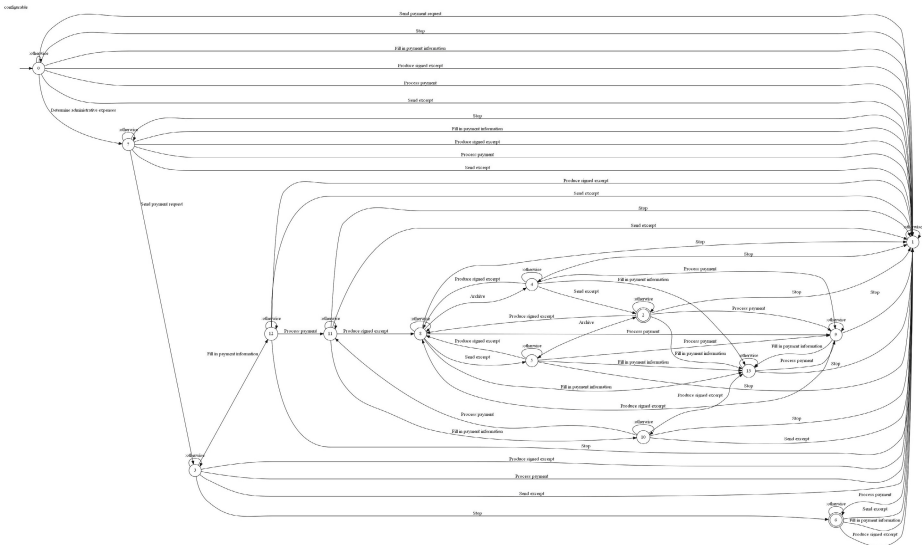


Fig. 2. Automaton obtained from the translation of the Declare model in Fig. 1

protocols, the process flow cannot be completely predefined, and the procedural way of modelling becomes impossible.

Unlike procedural languages, specifying what should be done, *declarative* languages specify which constraints may not be violated, and therefore they allow for comprehensible descriptions of processes with a high degree of variability. Declarative languages are also very appropriate for defining *compliance models*, which specify *what* should (not) be done instead of saying how it should be done. Consider, for instance, the set of rules in Fig. 1 from our case study. By translating this simple set of rules into an automaton¹, we obtain the procedural model in Fig. 2 that is much more complex.

¹ We have used the Declare tool (<http://www.win.tue.nl/declare/>) for the translation of Declare models to automata.

In recent years, a number of declarative process modelling languages were developed [1,10,11,14] and proven to be more suitable for certain application domains than procedural languages [6,12,19]. Nevertheless, since declarative process modelling languages attracted the attention of the research community at the later stage, when procedural languages were already massively used, there are still serious gaps in the domain of declarative process modelling which are still to be filled in. In this paper, we address one such a gap: *configurability of declarative process models*.

Nowadays, many branches of industry have semi-standardised collections of process models. Within one branch, process models of different organisations are often very similar due to legislation and (partial) standardisation, e.g., processes for registering a birth or extending a driving license would be very similar to each other for different municipalities. A one-size-fits-all approach with full standardisation of processes is however often inappropriate, as these organisations have good reasons for using specific variants of these common processes.

Configurable process models were introduced to solve the aforementioned problems [8,9]. They allow the user to change some parts of the model towards the user's preferences. This solves the one-size-fits-all problem and improves maintainability of processes since it becomes possible to describe several slightly different models by a single configurable model. When the configurable model is changed, all process models are updated automatically. To the best of our knowledge, the only kind of existing configurable process models are *procedural* models.

In this paper, we study in which way configurability in the declarative context is different from configurability in the procedural context. For this purpose, we consider the example declarative language *Declare* [11], in which constraints are LTL-formulas evaluated on traces of events executed in a process, and we define *Configurable Declare*. Since a declarative process model is a set of constraints over a set of events (representing the completion of a specific task in a business process), the *configuration options* we include in *Configurable Declare* are (1) *hiding an event* and (2) *omitting a constraint*. Through a *configuration*, it is possible to specify, for each configuration option, a boolean value that indicates whether a hideable event must be hidden or an omissible constraint must be omitted in the configured model.

Like most declarative languages, *Declare* works under the open world assumption, and, therefore, hiding an event does not mean forbidding this event to be executed. As the name suggests, hiding means allowing some event to become unobservable, unmonitored, unlogged. The behaviour of a model in which an event is hidden should remain the same as it was modulo this event. In the case of a *Declare* model, where process behaviour is considered to be defined by the set of traces (language) compliant with the model, hiding an event should result in a model with the same language modulo the hidden event.

To achieve language preservation, we take into account implicit constraints that would be lost if we simply removed from the model the event and the constraints connected to the event. For example, consider a model with constraints

“every paper submission is followed by a review” and “every review is followed by sending a notification letter” in which the event “review” gets hidden. This implies that the two constraints we have will be removed together with the event. To preserve the language modulo the hidden event “review”, we have to include the implicit constraint “every paper submission is followed by sending a notification letter” into the configured model. We define a derivation scheme for implicit constraints that allows us to have a sound transformation of a configurable model and a configuration to a configured model.

Since some configurations might lead to uninteresting or undesirable process variants, we introduce meta-constraints, which are defined as logical expressions over configuration choices, e.g., “if event A is hidden, then event B is not hidden”. Meta-constraints, in fact, restrict possible configurations to configurations that make sense from the content-wise perspective.

To support process modelling with *Configurable Declare*, we have developed **ConfDeclare** [16]. We have used this tool in the context of the *CoSeLoG²* project for a case study based on process models from ten Dutch municipalities. These models represent the production of an excerpt from the civil registration. In particular, starting from the different variants of the process (one for each municipality) we build a *Configurable Declare* model from which these variants can be derived.

Related Work. Configurable process models have been defined for a number of procedural modelling languages, e.g., *C-SAP WebFlow*, *C-BPEL*, *C-YAWL* [8], *CoSeNets* [18], and *C-EPC* [13]. Imperative configurable process models support a number of standard operations. Some patterns for procedural configurable models have been identified to support these operations [2,3,8,15]. *Configurable Declare* supports those patterns that can be mirrored to the declarative approach.

The paper is structured as follows: Section 2 gives a brief introduction to *Declare*. In Section 3, we introduce *Configurable Declare*. Section 4 explains the configuration steps: hiding an event and omitting a constraint. In Section 5, we show how to derive a *Declare* model from a *Configurable Declare* model and a configuration. Finally, in Section 6, we draw some conclusions and discuss directions for future work.

2 Declare: A User-Friendly Declarative Language

A process can be described by using different types of modelling languages. Process modelling languages can be classified according to two categories: procedural and declarative. A *procedural* model works with a “closed world” assumption, i.e., it explicitly specifies all the acceptable sequences of events in the process and everything that is not mentioned in the model is forbidden. Procedural process models can be used to provide a high level of operating support to participants

² <http://www.win.tue.nl/coselog/>

Table 1. FLTL operators semantics

<i>operator</i>	<i>semantics</i>
$\bigcirc\varphi$	φ has to hold in the next position of a path.
$\square\varphi$	φ has to hold in all the subsequent positions of a path.
$\diamond\varphi$	φ has to hold eventually (somewhere) in the subsequent positions of a path.
$\varphi U\psi$	φ has to hold in a path at least until ψ holds and ψ must hold in the current or in some future position.

who simply follow one of the allowed sequences in the model during the process execution. Therefore, this type of models is optimal in environments that are stable and where the process flow can be fully described in the model.

In contrast, a *declarative* model describes a process through constraints that should not be violated by the process execution. A declarative process model works with an “open world” assumption, i.e., any event is allowed unless it is explicitly forbidden by some constraint. This type of models can be used in highly dynamic environments where processes have a low degree of predictability. This is optimal when participants make decisions themselves and adapt the process flow accordingly (e.g., a doctor in a procedure to treat a fracture). Using declarative models for such processes allows for compact, readable representations.

In this paper, we study configurability of declarative process modelling languages taking *Declare* as an example of these languages. We explain here the basics of *Declare* necessary in the context of configurability and we refer the reader to [11] for a complete description of the language. *Declare* has formal semantics based on the use of a temporal logic, but the modeler is not confronted with this formal side, since the language has a user-friendly graphical notation capturing behavioural patterns expressible as temporal logic formulas. These patterns are a superset of the ones identified by Dwyer et al. in [5] and each of them has a specific graphical notation and semantics.

Given that business processes eventually terminate, *Declare* reasons on finite traces of events and uses a variant of LTL for finite traces called FLTL [7]. Table 1 contains the main FLTL operators and their semantics. Fig. 3 shows the graphical notation for the *response* constraint ($response(A, B)$) in *Declare*. The semantics of this constraint is captured in FLTL by $\square(A \Rightarrow \diamond B)$ (“every occurrence of event A is eventually followed by an occurrence of event B ”). In Table 2, we summarise the graphical notation and the FLTL semantics of the *Declare* constraints used in this paper.

A *Declare* model can be seen as a set of constraints, i.e., a conjunction of FLTL formulas over events. Formally, a *Declare* model can be defined as follows:

**Fig. 3.** The response constraint between A and B

Table 2. Graphical notation and FLTL semantics of the *Declare* constraints used in this paper

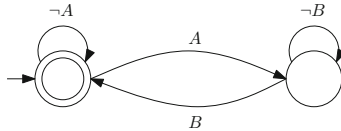
<i>constraint</i>	<i>FLTL semantics</i>	<i>graphical notation</i>
<i>response</i> (A, B)	$\Box(A \Rightarrow \Diamond B)$	
<i>precedence</i> (A, B)	$(\neg B \text{ U } A) \vee \Box(\neg B)$	
<i>succession</i> (A, B)	$\text{response}(A, B) \wedge \text{precedence}(A, B)$	
<i>alternate response</i> (A, B)	$\Box(A \Rightarrow \bigcirc(\neg A \text{ U } B))$	
<i>exclusive 1 of 2</i> (A, B)	$(\Diamond A \wedge \neg \Diamond B) \vee (\neg \Diamond A \wedge \Diamond B)$	
<i>init</i> (A)	A	

Definition 1 (Declare Model). A *Declare model* is a pair $M=(E, C)$, where E is a set of events and C is a set of FLTL formulas over events in E .

A trace of events belongs to the language of a *Declare* model, if it satisfies all the constraints of this model:

Definition 2 (Language). The language of a *Declare* model $M = (E, C)$, namely $\mathcal{L}(M)$, is the set of all traces satisfying all the constraints from C .

The formal semantics allows every *Declare* model to be executable and verifiable. To verify the validity of a constraint on a trace, the corresponding FLTL formula can be translated to a finite state automaton that accepts those and only those traces on which the formula is satisfied. The automaton for the response constraint in Fig. 3 is shown in Fig. 4; the initial state (marked by an edge with no origin) is here also an accepting state (indicated using a double outline). When event A happens, the state of the automaton is changed to a non-accepting state; it changes back to the initial accepting state only when a event B happens. Next to the positive labels, we also have negative labels (e.g., $\neg A$). They indicate that we can follow the transition for any event not mentioned (e.g., we can execute event C from the initial state and remain in the

**Fig. 4.** Automaton for $\text{response}(A, B)$

same state). This allows us to use the same automaton regardless of the input language, taking into account the open-world assumption. To verify the validity of a *Declare* model $M = (E, C)$ for a trace, we consider the automaton obtained from the conjunction of the constraints in the model.

Fig. 1 shows a simple *Declare* model describing the process for requesting an excerpt from the civil registration in a Dutch municipality. This model is part of a case study conducted in collaboration with ten Dutch municipalities in the *CoSeLoG* project. These municipalities model (and execute) this process in different, but still very similar ways, which makes it interesting in the context of configurability.

The *Declare* model in Fig. 1 involves nine *events*, depicted as rectangles, (e.g., *Send payment request*) and nine *constraints*, showed as connectors between the events (e.g., *succession*). Events represent the completion of a specific task in the business process. Constraints highlight mandatory and forbidden behaviours, implicitly identifying the acceptable sequences of events that comply with the model.

The constraint *init* shows that *Fill in e-form* must be the first task performed in the process; note that it is not forbidden to execute this task again later on. Since it is necessary to exactly evaluate the total amount of administrative expenses before formulating a payment request, *Send payment request* can be performed only after having performed *Determine administrative expenses*, as indicated by the *precedence* constraint between the two events. *Fill in payment information* must eventually be followed by *Process payment*, and *Process payment* cannot complete before that *Fill in payment information* is completed, which is captured by the *succession* constraint, being the combination of *precedence* and *response* constraints. The *exclusive choice* between events *Stop* (that stops the procedure) and *Fill in payment information*, depicted as a line with a black diamond, indicates that one of these two events must be performed in the process but not both. The *response* constraint between *Produce excerpt and sign it* and *Archive* indicates that whenever *Produce excerpt and sign it* is executed, *Archive* must eventually follow.

3 Configurable Declare

In this section, we introduce *Configurable Declare*, an extension of *Declare*. While in imperative languages, where the focus is on modelling the allowed behaviour, configuration options aim at making some behaviour optional (e.g., by blocking an event), in declarative languages the focus is on modelling restrictions on the behaviour, and therefore our configuration options will aim at making the restrictions optional. One reason to make some restrictions in the model optional can come from the inability to execute, control or monitor an event in some context. In this case, we want to allow for hiding an event depending on the configuration chosen. Another way to remove some restriction is by removing constraints from the model. Therefore, *Configurable Declare* introduces the possibility of annotating some constraints as omissible. Finally, we use meta-constraints to define which options for configuring the model are combinable and which not.

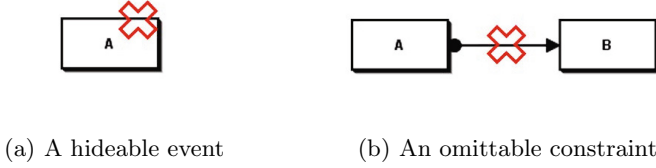


Fig. 5. Graphical representation of the configuration options

Formally, we define a *Configurable Declare* model as:

Definition 3 (Configurable Declare Model). A *Configurable Declare* model is a tuple (M, E_h, C_o, MC) where:

- $M = (E, C)$ is a *Declare* model,
- $E_h \subseteq E$ is a set of hideable events (graphically represented as in Fig. 5(a)),
- $C_o \subseteq C$ is a set of omissible constraints (graphically represented as in Fig. 5(b)), and
- MC is a set of meta-constraints and each meta-constraint defines a narrowing of the function space $(E_h \cup C_o) \rightarrow \mathbb{B}$. We call this narrowing the configuration space of the *Configurable Declare* model.

For the sake of readability, we overload the notation used and write, for example, $\neg e \Rightarrow (\neg c_1 \wedge \neg c_2)$ for the meta-constraint saying that if hideable event e is hidden, then omissible constraints c_1 and c_2 must be omitted, which implies that the configuration space only include functions, whose values on e, c_1 and c_2 obey the meta-constraint.

To configure a *Configurable Declare* model, the user has to make, for each hidable event and omissible constraint, a configuration choice specifying whether a hideable event should be hidden, and whether an omissible constraint should be omitted in the configured model.

Definition 4 (Configuration). Let $M_{conf} = (M, E_h, C_o, MC)$ be a *Configurable Declare* model. A configuration of M_{conf} is a function $conf : (E_h \cup C_o) \rightarrow \mathbb{B}$ from the configuration space of M_{conf} .

By applying a configuration to a *Configurable Declare* model, we obtain a configured model, which is one of the possible variants deducible from the given *Configurable Declare* model.

4 Configuration Steps

We choose a two-step approach for defining a configuration of a *Configurable Declare* model. In the first step –*abstraction*– the user defines the part of the configuration that specifies which hideable events must be hidden in the configured model, thus setting the context for this model. The *Configurable Declare*



Fig. 6. An implicit constraint between A and C can disappear when removing B

model is, then, transformed into a modified *Configurable Declare* model obtained by hiding the events that must be hidden according to the configuration. In the second step –*configuring constraints*– the user defines the part of the configuration that specifies which omissible constraints must be omitted in the configured model. Starting from the modified *Configurable Declare* model obtained in the first step, the configured model is derived by omitting the constraints that must be omitted according to the defined configuration.

4.1 Abstraction

In the abstraction step, the user defines a configuration over the hideable events by choosing (not) to hide events that are hideable in the *Configurable Declare* model. Note that hiding an event in a *Declare* model does not mean that it cannot occur, but it means that it is not monitored, implying that there can be no constraints restricting the execution of that event.

In this step, the user can possibly hide events which are involved in *implicit constraints*, i.e., constraints that can be deduced from other (explicit) constraints. Consider, for instance, the model in Fig. 6 where every A is eventually followed by B , and every B is eventually followed by C . By transitivity, also every A is eventually followed by C , which is an implicit constraint. Similarly to hiding in other settings (e.g., in the process algebra context), we want a model derived by hiding B to be *visible-language-equivalent* to the model where B is not hidden (with the hidden event considered to be invisible, like a τ -event). When hiding B , we cannot simply remove B together with all the constraints connected to B , since this would also remove the implicit constraint between A and C . To maintain the language equivalence, we have to take the implicit constraints into account, and, when necessary, make implicit constraints explicit.

Table 3 presents an excerpt from the list of constraint combinations connecting events A , B and C , and the corresponding implicit constraints that should be

Table 3. An excerpt of the language equivalences after the τ -abstraction of B

$$\begin{aligned}
 \mathcal{L}^{B \leftarrow \tau}(\text{response}(A, B) \wedge \text{response}(B, C)) &= \mathcal{L}(\text{response}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{response}(A, B) \wedge \text{succession}(B, C)) &= \mathcal{L}(\text{response}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{precedence}(A, B) \wedge \text{precedence}(B, C)) &= \mathcal{L}(\text{precedence}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{precedence}(A, B) \wedge \text{succession}(B, C)) &= \mathcal{L}(\text{precedence}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{succession}(A, B) \wedge \text{response}(B, C)) &= \mathcal{L}(\text{response}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{succession}(A, B) \wedge \text{precedence}(B, C)) &= \mathcal{L}(\text{precedence}(A, C)) \\
 \mathcal{L}^{B \leftarrow \tau}(\text{succession}(A, B) \wedge \text{succession}(B, C)) &= \mathcal{L}(\text{succession}(A, C))
 \end{aligned}$$

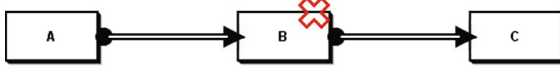


Fig. 7. The implicit constraint between A and C is not expressible in standard *Declare*

added when hiding B . In [16], the reader can find the full list of combinations of standard *Declare* constraints with the strongest implicit constraint (expressable in the standard *Declare*) corresponding to each combination. Several of these combinations do not allow for maintaining the language equivalence and the best we can do is to approximate the implicit constraint. Consider, for instance, the model in Fig. 7 consisting of two *alternate response* constraints. When B is not hidden, the configured model accepts (among others) traces $ABCABC$ and $ABACBC$, which are abstracted to $ACAC$ and $AACC$ when hiding B in the traces. Therefore, when B is hidden, the configured model does not accept any trace in which more than two occurrences of A happen without a C in between. By removing B from the model (i.e., by substituting B by τ), we would need to obtain a *Declare* model which accepts exactly such traces, but such a constraint is not expressible in *Declare*.

The approach provided in [16] considers as implicit constraint the closest constraint that is stronger than the necessary (inexpressible) constraint, which is *alternate response*(A, C) for the example considered (this constraint does not accept the trace $AACC$.) An alternative approach is to choose for the closest weaker constraint. This issue can be fully overcome by extending *Declare* with new constraints (in this case with a constraint that forces the occurrence of C after two occurrences of A). When using only the standard *Declare* constraints, the user should be notified in case the language preservation is violated.

Below we sketch the proof idea, supporting our method for hiding events. The proof is done by comparing the languages of the models obtained from the same *Configurable Declare* model using different configurations.

Theorem 1. *Let $M = (E, C)$ and $M' = (E', C')$ be *Declare* models obtained by not hiding/hiding an event $e \in E$ in the *Configurable Declare* model $((E, C), \{e\}, \emptyset, \emptyset)$, respectively. Then, $\mathcal{L}^{e \leftarrow \tau}(M) = \mathcal{L}(M')$, i.e., the language of model M with event e considered as invisible is the same as the language of model M' .*

Proof. (Idea) We can transform M to an equivalent model M'' where the implicit constraints in which e is involved are made explicit, maintaining the language equivalence between M and M'' . Considering that implicit constraints are deducible from the explicit constraints in the model, this implies that we do not constrain the behaviour any further. Afterwards, we transform M'' to M' by removing event e and all constraints associated with e . Showing that $\mathcal{L}^{e \leftarrow \tau}(M'') = \mathcal{L}^{e \leftarrow \tau}(M')$ is straightforward, using the equivalences as shown in Table 3. Therefore, we can conclude that model M is visible-language equivalent to the model M' (with e abstracted to τ). \square

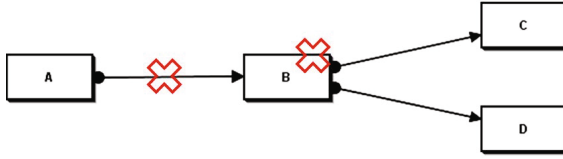


Fig. 8. Implicit constraints between A and C and between A and D are omittable

When a hidden event is involved in omittable constraints, the situation becomes slightly more complicated. If an implicit constraint to be added to the model is derived using an omittable constraint, the implicit constraint should be added as an omittable constraint. Consider, for instance, the model in Fig. 8. The constraint between A and B can be omitted and B can be hidden. If we choose to hide B , we have to make explicit the implicit constraints between A and C (c_1) and between A and D (c_2). However, since the constraint between A and B is omittable, we have to make the implicit constraints c_1 and c_2 also omittable.

At the same time, we need to preserve correlations between implicit constraints. Consider, again, the model in Fig. 8. If the constraint between A and B is omitted, then both the implicit constraints c_1 and c_2 disappear. This introduces a correlation between the implicit constraints c_1 and c_2 : they should be either both present or both omitted in the configured model. This can be encoded through meta-constraints as $c_1 \Leftrightarrow c_2$.

Deducing which correlations have to be maintained between pairs of implicit constraints is straightforward. Consider, for instance, the models in Fig. 9, in which all implicit constraints have been made explicit. Constraint c_4 is deduced from c_1 and c_3 , and c_5 is deduced from c_2 and c_3 . In the model in Fig. 9(a), explicit constraints c_1 and c_2 , and c_3 are all omittable. Starting from the omittable (explicit) constraints, we can build the deduction graph in Fig. 10(a). In this graph, if explicit constraints are used to deduce an implicit constraint, we include an hyperarc between the explicit constraints and the implicit constraint. For instance, c_4 is deduced from c_1 and c_3 , hence, we include a hyperarc between c_1 , c_3 and c_4 . Using the hyperarcs, we can induce meta-constraints like $(c_1 \wedge c_3) \Leftrightarrow c_4$.

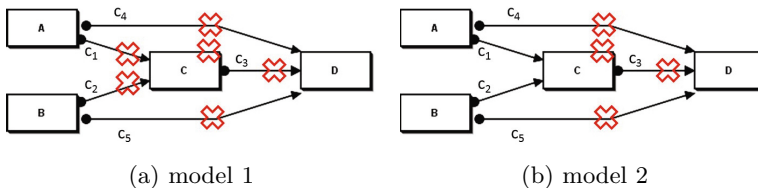


Fig. 9. Configurable Declare models with the implicit constraints (c_4, c_5) made explicit

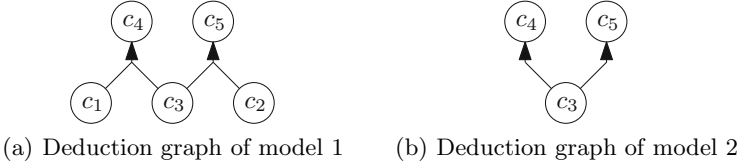


Fig. 10. Deduction graphs of the *Configurable Declare* models in Fig. 9

Using the deduction graphs, we can easily induce correlations between implicit constraints. Consider the *Configurable Declare* model in Fig. 9(b) and the corresponding deduction graph in Fig. 10(b). From Fig. 10(b), we can induce the meta-constraints $c_3 \Leftrightarrow c_4$ and $c_3 \Leftrightarrow c_5$. By transitivity, we obtain the meta-constraint $c_4 \Leftrightarrow c_5$. Therefore, when hiding C , c_4 and c_5 must be both omitted or both not omitted.

From the technical perspective, it would be easier to let the user first define which constraints should be omitted and then choose which events should be hidden. In this case, omittability of implicit constraints and correlations between them would be irrelevant. This would, however, require the user to make choices about constraints that are defined on events which are not relevant for her practice.

4.2 Configuring Constraints

In the second step, the user can decide which omittable constraints should be omitted in the configured model. It is also possible to include the option of substituting a constraint by a different constraint into *Configurable Declare*. However, this option can be considered as syntactic sugar. Indeed, substituting a (default) constraint with different constraints can be obtained by considering these constraints as omittable and providing a meta-constraint specifying that exactly one of this constraints can be kept in the configured model.

In Fig. 11(a), for instance, we show a model in which the $response(A, B)$ can be substituted by either $precedence(A, B)$ or by $alternate\ response(A, B)$. This can also be encoded through omittable constraints as depicted in Fig. 11(b) with a meta-constraint enforcing that exactly one of the omittable constraints is kept in the configured model. This is encoded as $c_1 \underline{\vee} c_2 \underline{\vee} c_3$, where $\underline{\vee}$ is the exclusive or.

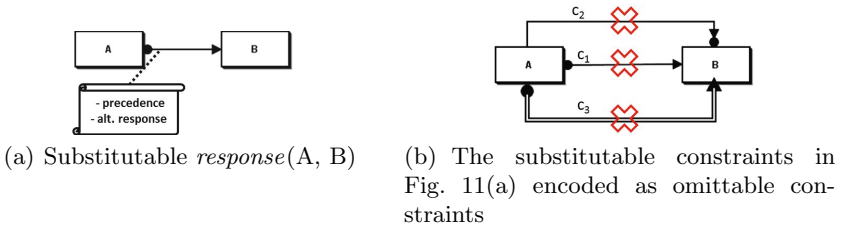


Fig. 11. Substitutable constraints in *Configurable Declare*

5 Methodology and Case Study

In this section, we present the deduction of a *Declare* model from a *Configurable Declare* model. As mentioned before, this is done through a two-step approach. First the context is set (Subsection 5.1), and then, some constraints are selected to be omitted in the configured model (Subsection 5.2). The methodology is presented by using an example from our case study. Further results about the case study are given in Section 5.3.

5.1 Setting the Context

The first step for configuring a *Configurable Declare* model consists in selecting which events should be controlled and which events are uncontrolled in the configured model. Using Algorithm 1, the *Configurable Declare* model is transformed into a (modified) *Configurable Declare* model. Here, the hideable events that are chosen to be hidden (denoted as set S_h) are removed from the model. It can be the case that hiding an event invalidates some meta-constraints, or even that the event to be hidden is not hideable. Therefore, we first check whether all meta-constraints are satisfied and whether all events in S_h are hideable ($S_h \subseteq E_h$). If this is the case, the events in S_h are removed from the *Configurable Declare*

Algorithm 1: Setting the context for a *Configurable Declare* model

SETTHECONTEXT(M_{conf} , S_h , $M_{conf'}$)

Input: M_{conf} the *Configurable Declare* model, $S_h \subseteq E_h$ the set of hidden events

Output: $M_{conf'}$ the *Configurable Declare* model after abstraction

- (1) **if** the meta-constraints MC are not satisfied
- (2) **return**
- (3) **else**
- (4) $M_{temp} \leftarrow M_{conf}$
- (5) $\mathcal{T} \leftarrow$ all implicit constraints (using the transitive closure based on the rules in [16])
- (6) **foreach** event $e \in M_{temp}$
- (7) **if** $e \in S_h$
- (8) $\mathcal{C} \leftarrow$ all implicit constraints which have to be made explicit after hiding e using the defined rules (see [16])
- (9) add all constraints from \mathcal{C} to M_{temp}
- (10) add all meta-constraints related to the removal of e (based on \mathcal{T}) to M_{temp}
- (11) Remove all events from M_{temp} which are hidden
- (12) Remove all constraints from M_{temp} which are related to hidden events
- (13) Update all meta-constraints in M_{temp} which are related to hidden events
- (14) **return** M_{temp}

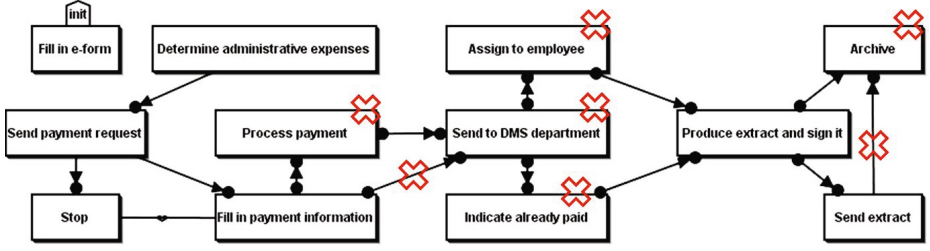


Fig. 12. The *Configurable Declare* model

model, the implicit constraints are made explicit if needed, and meta-constraints are updated accordingly. Otherwise, the empty model is returned.

Consider the *Configurable Declare* model in Fig. 12 (without any specified meta-constraint at the beginning). Suppose that the user chooses to hide events *Assign to employee*, *Send to DMS department*, and *Indicate already paid*.

If we hide *Send to DMS department*, we need to add to the modified model ($M_{conf'}$) a *succession* constraint between *Process payment* and *Assign to employee* (c_1), and between *Process payment* and *Indicate already paid* (c_2). Furthermore, we have to include in $M_{conf'}$ a *succession* constraint between *Fill in payment information* and *Assign to employee* (c_3), and between *Fill in payment information* and *Indicate already paid* (c_4). Since the *succession* between *Fill in payment information* and *Send to DMS department* can be omitted, we have to maintain the correlation between c_3 and c_4 , i.e., the meta-constraint $m_1 = c_3 \Leftrightarrow c_4$ is added to $M_{conf'}$.

In the second iteration, we process *Assign to employee*. This introduces a *succession* between *Process payment* and *Produce extract and sign* (c_5) in $M_{conf'}$. Furthermore, the *succession* between *Fill in payment information* and *Produce extract and sign it* (c_6) has to be made explicit, and we have to include the meta-constraint $m_2 = c_4 \Leftrightarrow c_6$ in $M_{conf'}$.

If we hide *Indicate already paid*, we need to add to $M_{conf'}$ the constraints *succession*(*Process payment*, *Produce extract and sign it*) and *succession*(*Fill in payment information*, *Produce extract and sign it*). Note that no new meta-constraints have to be included at this iteration. Finally, we have to remove the hidden events from the model, and remove the constraints and update the meta-constraints related to any of those events. This yields the *Configurable Declare* model depicted in Fig. 13 (with meta-constraints m_1 and m_2).

5.2 Configuring Constraints

The second step for configuring a *Configurable Declare* model consists of selecting which constraints have to be omitted in the configured model (we indicate this set of constraints with S_o). Omitting a constraint might invalidate some meta-constraints, or it can be the case that the constraint to be omitted is not omissible. Therefore, we first check whether all meta-constraints are satisfied

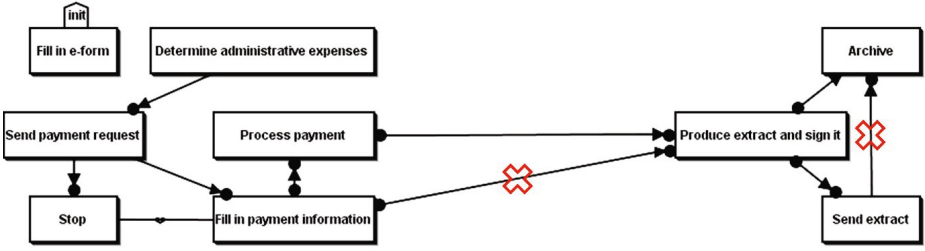


Fig. 13. The *Configurable Declare* model for municipality A after setting the context

and whether all constraints in S_o are omissible ($S_o \subseteq C_o$). If this is the case, the constraints in S_o are removed from the *Configurable Declare* model. Otherwise, the empty model is returned (Algorithm 2).

Suppose that we start from the *Configurable Declare* model depicted in Fig. 13 (with meta-constraints m_1 and m_2). Suppose that the user chooses to remove the *succession* between *Fill in payment information* and *Produce extract and sign it*, and the *precedence* between *Archive* and *Send excerpt*. Since m_1 and m_2 are satisfied, *succession* and *precedence* can be removed from the model yielding the model depicted in Fig. 1, which belongs to municipality A.

5.3 Case Study

For the case study, we have used models from the *CoSeLoG* project adopted by ten different Dutch municipalities. We have used the model for municipality A (depicted in Fig. 1) as running example to present our proposed approach. To obtain the models depicted in Fig. 14 and in Fig. 15 for municipalities B and C, we use the configurations showed in Table 4 and in Table 5.

Algorithm 2: Removing omitted constraints from the *Configurable Declare* model

CONFIGURABILITY($((E, C), E_h, C_o, MC), S_o, M_{conf'}$)

Input: $((E, C), E_h, C_o, MC)$ the *Configurable Declare* model, $S_o \subseteq C_o$ the set of omitted constraints

Output: M_{decl} the *Declare* model after omitting the constraints

- (1) **if** the meta-constraints MC are not satisfied
- (2) **return**
- (3) **else**
- (4) **return** $(E, C \setminus S_o)$

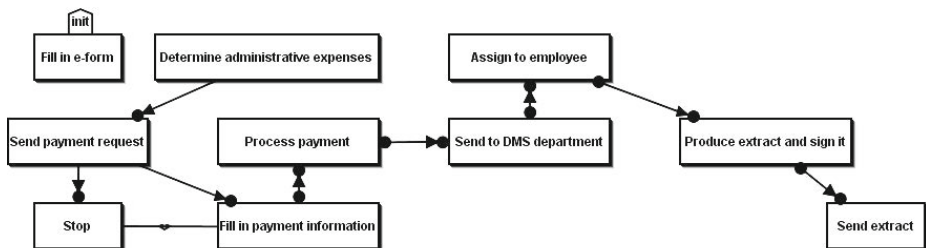
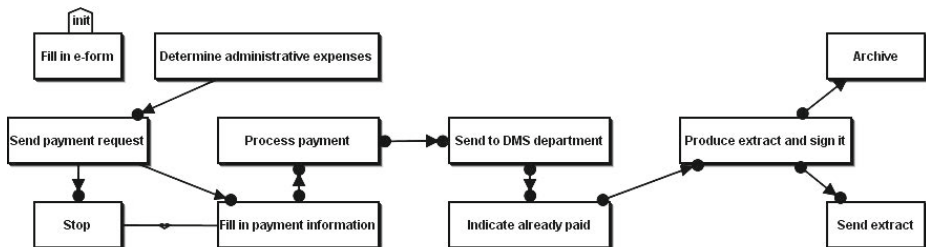
Fig. 14. The *Declare* model for municipality BFig. 15. The *Declare* model for municipality C

Table 4. The context for municipalities B and C

Event	Mun. B	Mun. C
<i>Archive</i>	hidden	not hidden
<i>Assign to employee</i>	not hidden	hidden
<i>Indicate already paid</i>	hidden	not hidden
<i>Process payment</i>	not hidden	hidden
<i>Send to DMS department</i>	not hidden	not hidden

Table 5. The configurability for municipalities B and C

Constraint	Mun. B	Mun. C
$precedence(Archive, Send\ Extract)$	omitted	omitted
$succession(Fill\ in\ payment\ information, Send\ to\ DMS\ department)$	omitted	omitted

6 Conclusion

In this paper, we defined *Configurable Declare*, a configurable declarative language. The configurability setting for declarative languages differs from the setting for procedural languages. Indeed, while adding configurability options for procedural languages implies that *more options for allowed behaviour* get included in the model, adding configurability options for declarative languages results in the inclusion of *more options for restricting behaviour*.

We have defined an approach to transform a *Configurable Declare* model and a given configuration into a *Declare* model. While in the declarative setting removing a constraint turned out to be a trivial transformation, in the procedural

setting removing a dependency between two events without influencing dependencies between other events is far from being trivial. On the other hand, hiding an event is easy to implement in the procedural setting, whereas it requires a dedicated mechanism to maintain implicit constraints in the declarative setting.

We have applied our approach as a proof of concept to a case study and we have been able to capture processes of ten Dutch municipalities in one readable *Configurable Declare* model. This paper must be considered as a starting point for *Configurable Declare* and there are several research directions we want to investigate concerning this topic. Below we elaborate on some of them.

Outlook. Building a *Configurable Declare* model from scratch is a logical option when a completely new process needs to be designed. However, in many cases (like in our case study) organisations already have models of their processes available and the configurable model should be built based on some existing knowledge. To make it possible, we are working on an approach for automatic generation of a *Configurable Declare* model from a given set of *Declare* models in such a way that the original *Declare* models are derivable from the generated *Configurable Declare* model (by applying some configurations). A related question is how to derive automatically a configuration for a given *Configurable Declare* model resulting in a model that is similar to a given *Declare* model.

When an organisation wants to start configuring a configurable model for some existing process for which no model is available, event logs can be used for deriving an appropriate configuration.

Finally, we would like to introduce patterns for meta-constraints in order to ease the design process. In particular, we want to develop a method for the automated deduction of meta-constraints to forbid configurations that lead to unsatisfiable models (models with no behaviour), or to models in which some important events become not executable or some important constraints become trivially true [17].

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: The Role of Business Processes in Service Oriented Architectures. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
2. Becker, J., Delfmann, P., Knackstedt, R., Kuropka, D.: Configurative process modeling - outlining an approach to increased business process model usability. In: Proceedings of the 15th Information Resources Management Association Information Conference (2004)
3. Dreiling, A., Rosemann, M., van der Aalst, W.M.P., Heuser, L., Schulz, K.: Model-based software configuration: patterns and languages. EJIS 15(6), 583–600 (2006)
4. 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 Interscience (2005)
5. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering, ICSE 1999, pp. 411–420. ACM (1999)

6. Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S.: Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) *BPMDs 2009 and EMMSAD 2009*. LNBIP, vol. 29, pp. 353–366. Springer, Heidelberg (2009)
7. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: *ASE*, pp. 412–416. IEEE Computer Society (2001)
8. Gottschalk, F.: *Configurable Process Models*. Ph.D. thesis, Eindhoven University of Technology, The Netherlands (December 2009)
9. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M., La Rosa, M.: Configurable Workflow Models. *International Journal on Cooperative Information Systems* 17(2) (2008)
10. Lamport, L.: The temporal logic of actions. *ACM Trans. Program. Lang. Syst.* 16(3), 872–923 (1994)
11. Pesic, M.: *Constraint-Based Workflow Management Systems: Shifting Controls to Users*. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
12. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) *BPM Workshops 2011, Part I*. LNBIP, vol. 99, pp. 383–394. Springer, Heidelberg (2012)
13. Rosemann, M., van der Aalst, W.M.P.: A configurable reference modelling language. *Information Systems* 32, 1–23 (2007)
14. Sadiq, S.W., Orłowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. *Information Systems* 30(5), 349–378 (2005)
15. Schumm, D., Leymann, F., Streule, A.: Process viewing patterns. In: *EDOC*, pp. 89–98 (2010)
16. Schunselaar, D.M.M.: *Configurable Declare*. Master’s thesis, Eindhoven University of Technology (2011), <http://alexandria.tue.nl/extra1/afstvers1/wsk-i/schunselaar2011.pdf>
17. Schunselaar, D.M.M., Maggi, F.M., Sidorova, N.: Patterns for a Log-Based Strengthening of Declarative Compliance Models. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) *IFM 2012*. LNCS, vol. 7321, pp. 327–342. Springer, Heidelberg (2012)
18. Schunselaar, D.M.M., Verbeek, E., van der Aalst, W.M.P., Raijers, H.A.: Creating Sound and Reversible Configurable Process Models Using CoSeNets. In: Abramowicz, W., Kriksuniene, D., Sakalauskas, V. (eds.) *BIS 2012*. LNBIP, vol. 117, pp. 24–35. Springer, Heidelberg (2012)
19. Zugal, S., Pinggera, J., Weber, B.: The Impact of Testcases on the Maintainability of Declarative Process Models. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) *BPMDs 2011 and EMMSAD 2011*. LNBIP, vol. 81, pp. 163–177. Springer, Heidelberg (2011)

Efficacy-Aware Business Process Modeling

Matthias Lohrmann and Manfred Reichert

Ulm University,
Databases and Information Systems Institute
{matthias.lohrmann,manfred.reichert}@uni-ulm.de

Abstract. In business process design, business objective models can fulfill the role of formal requirement definitions. Matching process models against objective models would, for instance, enable sound comparison of implementation alternatives. For that purpose, objective models should be available independently of their concrete implementation in a business process. This issue is not addressed by common business process management concepts yet. Moreover, process models are currently not sufficiently expressive to determine business process efficacy in the sense of fulfilling a business objective. Therefore, this paper develops and integrates constructs required for efficacy-aware process modeling and apt to extend common modeling approaches. The concept is illustrated with a sample scenario. Overall, it serves as an enabler for progressive applications like automated process optimization.

Keywords: Business Process Modeling and Analysis, Business Process Design, Business Objectives and Goals.

1 Introduction

The notion of business goals, objectives, or similar concepts has been widely used to define the term *business process* (e.g., [1]). At the same time, semantic quality of process models has been recognized as an important prerequisite for successful adoption [2]. Nevertheless, objectives are still a notable exception to the progress towards formal business process semantics, and are only rudimentarily considered in common modeling approaches [3, 4]. The effectiveness of processes in regard to achieving business objectives can be subsumed as *business process efficacy*. The case for assessing and controlling business process efficacy with formal business objective models can be illustrated by considering exemplary application scenarios:

Scenario 1 (Automated Business Process Optimization). Process-aware information systems (PAISs) collect data on process execution that could be leveraged for automated business process optimization [5]. Consider, for instance, process abortions: if a process instance cannot be completed, it should abort as early as possible to avoid unnecessary consumption of resources. Next-generation PAISs might re-arrange control flow to foster this behavior based on the execution logs of past instances. However, this must be done in a way to maintain the overall efficacy of the business process. Thus, a semantic link between business objectives and business process models is required.

Scenario 2 (Identification of Business Process Variants). The management of business process variants has emerged as an important business process management (BPM) issue [6–8]. However, criteria to determine whether two process models are variants of the same reference process remain a “missing link”. In this respect, modeling business processes in a way that enables tracing to common business objectives can provide an effective characteristic to assess the “equivalence” of process variants.

Scenario 3 (Benchmarking). Qualitative benchmarking deals with good practices to identify opportunities for process improvement [9]. This often meets the resistance of practitioners as the equivalence of process alternatives regarding their outcome is doubted. Formalizing efficacy can help to alleviate this issue. Similar considerations apply to more recent approaches like process performance management [10].

As depicted in Figure 1, our approach contributes capabilities which are required to address the scenarios lined out, but not provided by the state of the art. This includes a clear distinction between business objectives (as a formal requirements definition) and business processes (as an implementation), and the ability to formally determine whether and under which circumstances a business objective is achieved, i.e. whether a business process is *efficacious*. Proper formalization will, in the end, be key to efficient automation. Seamless integration of new concepts with existing BPM approaches fosters applicability.

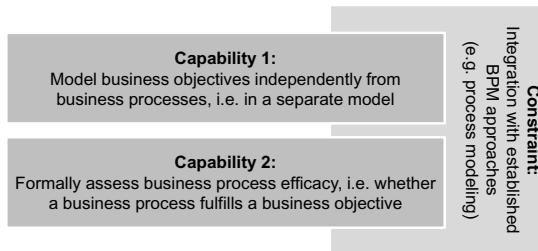


Fig. 1. Contribution Overview

2 Methodology and Outline

In general, business objectives exist independently of business processes. A certain process constitutes just one of many potential alternatives to achieve its objective, e.g. by using another IT system or re-arranging the order of activities. In other words, a business objective is achieved by inducing a state that satisfies particular criteria – no matter how this is done. Therefore, assessment and control of business process efficacy generally require two modeling facilities:

1. A *business objective meta-model* that is sufficiently expressive to model business objectives independently of corresponding business processes in the sense of a formal requirements definition.

2. An *efficacy-aware business process meta-model* that is sufficiently expressive to determine whether a corresponding business process model fulfills a business objective, i.e. whether it is efficacious.

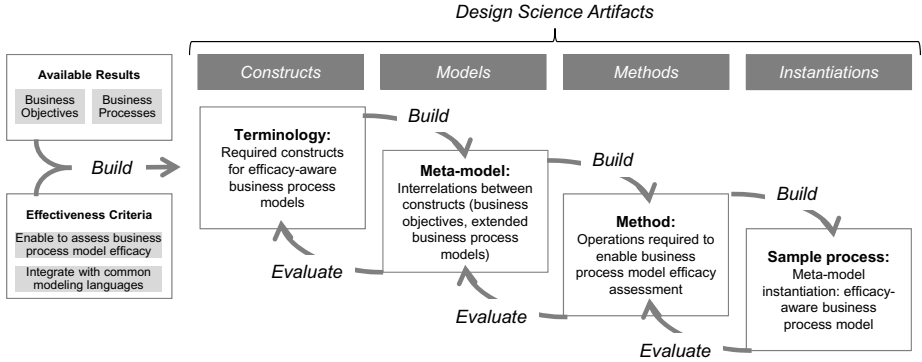


Fig. 2. Methodology to Conceive an Efficacy-aware Business Process Meta-model

As we presented results towards business objective modeling in [11], the focus of this paper lies on the second modeling facility, the *efficacy-aware business process meta-model*. Since this is a *goal-bound artificial construct*, we orient our methodology at the design science paradigm [12, 13]. Figure 2 presents an overview reflecting the design procedures *build* and *evaluate*, and the design artifacts *constructs*, *models*, *methods*, and *instantiations* [13]. As a first step, we *build* a set of required *constructs* based on available results. This provides us with a terminology for further considerations. We then describe the relations between the constructs identified, thus *building* a meta-model.¹ The meta-model is amended with operations required for efficacy assessment in the sense of a *method*. On that basis, efficacy-aware business process models can be *built*. *Evaluation* of results then occurs in reversed order: a sample process is used to evaluate the effectiveness of the method, the meta-model and, in turn, the set of constructs.

In terms of functionality, the resulting artifacts can be considered as effective if efficacy assessment has been enabled. As an additional effectiveness criterion to ensure applicability, we also demand that results should integrate well with existing business process modeling languages. This means that new constructs and meta-model elements should only be used where required due to limitations in well-established languages, and all new constructs should be well-connected to existing terminology. Note that these effectiveness criteria correspond to the second capability and the constraint stipulated in Figure 1. The first capability, separating objective from process models, has been addressed in [11] which provides the basis for this paper.

¹ Note the shift in terminology: in terms of design science artifacts, our meta-model constitutes a *model*.

Accordingly, Section 3 reflects existing results and related work as a starting point to build our approach. Section 4 derives terminology required for efficacy assessment. Section 5 builds a meta-model for efficacy-aware business processes. Section 6 presents operations required for efficacy assessment as well as an exemplary instantiation of the meta-model to discuss the validity of our results.

3 Background

This section discusses related approaches and summarizes our previous work.

3.1 Related Work

Since the notion of goals or objectives is part of most definitions of the term “business process”, there have been a number of approaches towards integrating objectives into process modeling. Table 1 summarizes related approaches along a set of semantic requirements relevant in the context of business objectives. A more detailed analysis is presented in [11].

Table 1. Semantic Requirements and Related Work

Semantic Requirements	Reflection in Related Work	Conclusions
<i>Consideration of the affecting environment:</i> Objectives must consider the state of both target artifacts to be created and altered <i>and</i> environmental conditions (e.g. in decision processes).	Mostly no formal, state-based concept of objectives achievement (e.g., [14–16]); objectives may be viewed not as state to be achieved, but as set of tasks to be executed (e.g., [17, 18]).	The requirement to delineate objectives from activities and to sufficiently consider environmental conditions are still open issues.
<i>Varying target environment:</i> Depending on environmental conditions, the set of artifacts to be created or altered and the set of operations to be carried out may vary.	Goals are mostly discussed on an abstract level without referring to single target artifacts (e.g., [14, 15]) or without an environmental conditions concept (e.g., [16, 18]).	Flexible adaptation to environmental conditions in terms of actually required process results is still an open issue.
<i>Order constraints:</i> Constraints to the order of activities to be carried out must be representable).	Constraints are partially considered as an abstract construct [18] or via consistency of paths [17]. Other approaches omit order constraints (e.g. [14–16]).	The notion of constraints is partly available, but needs to be refined.

The gap of existing approaches regarding the coverage of semantic requirements is not surprising considering that goal structures are mostly used as an auxiliary tool in process design, but not as a means to formally manage efficacy. Typically, this leads to the absence of separate business *objective* models. Instead, business objectives are integrated into business *process* models.

This view is also reflected in a number of general process modeling formalisms (e.g., EPCs [4]) as well as, in a broader context, enterprise architecture approaches (e.g., [19]). It corresponds to the *methods* category of design science

artifacts, and the merits of related concepts should be evaluated in this context. In contrast, this paper is mainly focused on *constructs* and *models*.

Beyond the related work discussed in Table 1, it is instructive to consider i^* [20] and KAOS [21] from the requirements engineering field. The i^* framework aims at documenting actors’ goals and dependencies in early-phase requirements engineering, but not on formalizing objectives. Accordingly, i^* addresses a different lifecycle stage and cannot be matched against the semantic requirements for our approach. In turn, KAOS provides a framework for capturing aspects relevant to information systems requirements engineering via an “acquisition strategy”. The *constraints* construct in KAOS corresponds to the concept of business objectives in [11] and could be extended by the aspects relevant to BPM as discussed there (cf. Section 3.2). The focus of this paper, however, is on integrating with common BPM approaches instead of requirements engineering. To avoid unnecessary complexity, we thus settle for our approach which is more specific in this respect.

Approaches towards the compliance of process models to given rules (e.g., [22]) are aimed at ensuring the compliance of process execution with constraints imposed (e.g., legal requirements), but do not address issues such as deriving required resources from a process model. They are thus not sufficient to enable efficacy assessment.

3.2 Available Results from Previous Work

Addressing the first capability stipulated in Section 1, we suggested and evaluated a meta-model for formal business *objectives* modeling in [11]. This paper focuses on its integration with common business *process* modeling concepts to enable efficacy assessment. Consequently, this section provides an overview on relevant business objective concepts needed for the understanding of our results.

Business processes are enacted to induce a change to their environment. The intended change constitutes a *business objective*. A naïve approach might be to simply model a set of actions required to fulfill the business objective. However, this is not sufficient, as business processes need to *interact* with their environment. This means that the intended final state must be derived from the initial state of the environment. As an example, consider the approval of loans. The loan decision must consider the customer’s credit history. Thus “approve loan” is not sufficient to describe the business objective. This challenge lies behind most of the constructs shortly presented in the following.

For business objective modeling, we discern *target elements* as characteristics of the business process’s environment to be altered, and *conditional elements* as characteristics that need to be considered to determine the intended final state. Both make up the *environmental elements* of a business process. In our loan approval example, the loan decision constitutes a target element, and its aspired value depends on the state of the customer’s credit history as a conditional element. To describe the state of both target and conditional elements, we use the concept of *binary state determinants (BSDs)*.

We discern between *target BSDs* and *conditional BSDs*, which relate the state of target and conditional elements, respectively, to a value range, either absolutely or in terms of other conditional elements. If the respective relation holds, the BSD is *fulfilled*. Regarding our loan example, “loan decision = approved” might be a target BSD, and “overdues < 5% of credit volume” might be a conditional BSD. Target BSDs are classified according to types reflecting their semantic interrelation with environmental conditions which can, in turn, be expressed through sets of conditional BSDs. Figure 3 provides an overview.

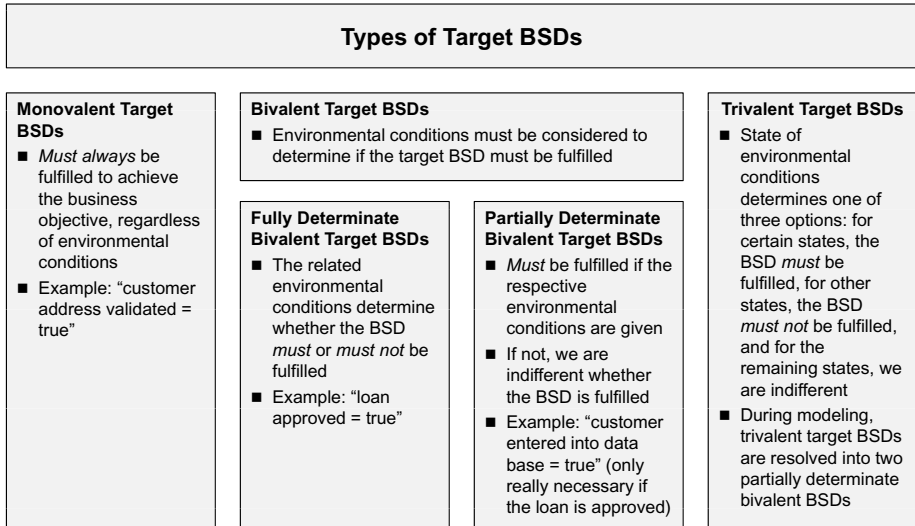


Fig. 3. Types of Target BSDs

To model the environmental conditions that determine whether a bivalent target BSD must be fulfilled, each bivalent target BSD is linked to a *conditional proposition*. Conditional propositions consist of conditional BSDs that are arranged into sets of necessary and sufficient sub-conditions (cf. Example 1). The sub-conditions allow minimizing the number of required checking actions by following a strategy to quickly approve or disapprove the proposition.

Thus, a *business objective* bundles a set of target BSDs. It is achieved if and only if each target BSD comprised has assumed a state reflecting its conditional propositions. An *efficacious business process* has to approve or reject each conditional proposition, and manipulate target elements accordingly. Figure 4 shows an exemplary business objective model. The corresponding semantics are described in Example 1.

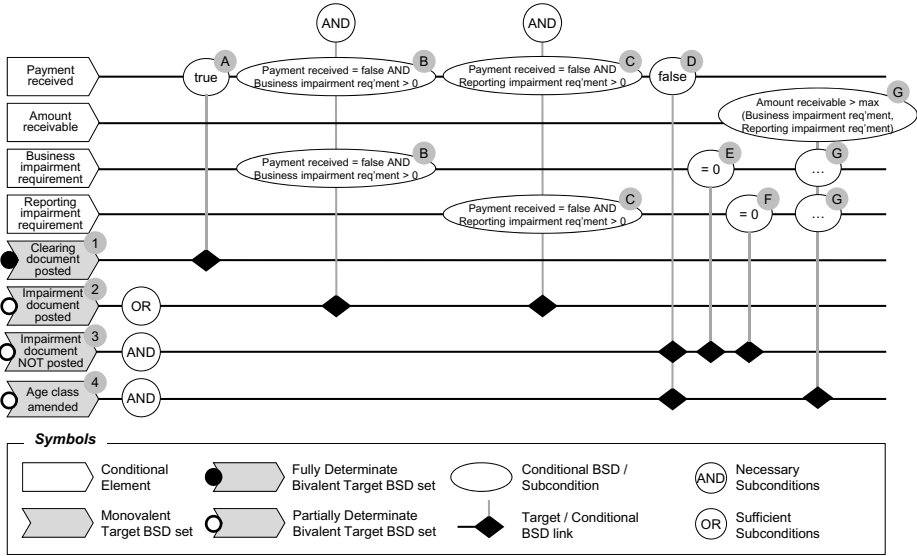


Fig. 4. Exemplary Business Objective: Year-end Receivables Processing

Example 1 (Business Objective: Year-end Receivables Processing). Properly processing receivables, e.g. open customer invoices, constitutes a business objective during year-end closing in accounting. Figure 4 informally presents the respective objective model described in the following. The top four horizontal lines in the model correspond to the relevant conditional elements, and the bottom four lines correspond to target BSDs. Vertical lines and nodes are used to link conditional elements and target BSDs by way of conditional propositions. For reference, target BSDs and sub-conditions have been amended with numbers and literals, respectively. The individual target BSDs are modeled as follows:

- Target BSD *Clearing document posted* (1): If payment has been received for the receivable, it must be cleared, i.e. removed from the list of open items. If not, a clearing document must not be posted. Accordingly, *Clearing document posted* constitutes a fully determinate bivalent target BSD linked to *Payment received* (A) as a conditional BSD. Since there are no other conditional BSDs to be considered, there is no need to discern sufficient and necessary sub-conditions.
- Target BSDs *Impairment document posted* / *Impairment document NOT posted*: An open receivable must be impaired (i.e., its book value as an asset must be reduced) in certain cases, but it must not be impaired in others. Moreover, there may be circumstances where we are indifferent. Accordingly, *Impairment document posted* constitutes a trivalent target BSD which we resolve into two partially determinate bivalent ones: *Impairment document posted* and *Impairment document NOT posted*.

- Target BSD *Impairment document posted* (2): Business and reporting impairment requirements are needed if the receivable is not being fully recoverable according to management’s judgment or having to be impaired for (legal) reporting guidelines, respectively. If there is no payment but a business impairment requirement (B), or if there is no payment but a reporting impairment requirement (C), the impairment must be posted. Accordingly, the OR label associated with the target BSD indicates that there are two sufficient sub-conditions, each consisting of two conditional BSDs.
- Target BSD *Impairment document NOT posted* (3): On the other hand, if no payment has been received (D), and there is neither a business nor a reporting impairment requirement (E and F), an impairment must not be posted. Thus, there are three necessary sub-conditions as indicated by the AND label going with the target BSD.
- Target BSD *Age class amended* (4): If an open receivable has not been cleared (D) and its amount is greater than the amount to be impaired (G), it must be amended with an age class for correct balance sheet reporting (e.g., “> 12 months”). If the receivable has been reduced to zero through clearing or impairment, we are indifferent whether an age class is amended. Therefore, *Age class amended* constitutes a partially determinate target BSD with two necessary sub-conditions.

Note that for target BSDs with more than one conditional BSD, the notation allows showing either necessary or sufficient sub-conditions, depending on the modeler’s choice, and that monovalent Target BSDs do not occur in our example.

The business objectives modeling approach has been derived from semantic requirements (cf. Table 1) and tested against usability criteria by applying it to an exemplary real-world case with a modeling *method* [11]. In contrast to related work (cf. Section 3.1), it provides the ability to formally describe business objectives as intended states of the environment. It also provides a convention to model order constraints. However, this topic is not covered by our example, as it is more of a challenge when modeling business objectives *without* referring to a concrete process model.

4 Business Process Model Efficacy

In Section 3.2, we discussed how business objectives can be modeled independently from business processes, thus addressing the first capability in Figure 1. This section focuses on the second capability, enabling efficacy assessment. We first deepen our understanding of what constitutes an efficacious business process. Then, we discuss what information is required to assess efficacy.

Considering the notion of business objectives in [11], we can define:

Definition 1 (Business Process Model Efficacy). *A business process model is formally efficacious iff no target BSD in its business objective can be fulfilled unless the respective conditions defined by the business objective are fulfilled.*

A business process model is fully efficacious iff it is formally efficacious and all conditions which the model poses to target BSDs, but which are not defined by the business objective, are considered as reasonable by subject matter experts.

A business process model is ideally efficacious iff it is formally efficacious and there are no additional conditions posed to target BSDs beyond those defined by the business objective.

Note that *ideally efficacious business processes* do not occur in practice as each business process requires resources which are not part of the business objective (e.g., expenditure of labor).

According to Def. 1, assessing efficacy requires to analyze process models regarding the conditions they pose towards the fulfillment of target BSDs. To assess formal efficiency, the conditions obtained are then compared to the conditions posed by the objective model. Moreover, to assess full efficacy, they are in matched against subject matter experts' expectations.

Example 2 (Efficacy Assessment). As an example, reconsider the loan approval process. Comparing it with the business objective, in this case, will clarify whether decision criteria for loan approval such as the credit history are observed, i.e. whether the process is formally efficacious. For full efficacy, however, we also need to consider whether an unreasonable amount of working time is required for the process. This is not documented in the business objective, but requires the judgment of subject matter experts.

Efficacy assessment can be supported by consolidating and properly structuring the conditions a business process poses to individual target BSDs. Similar to the modeling of business objectives in [11], this can be achieved by amending target BSDs with conditional propositions consisting of conditional BSDs. In contrast to business objectives, business processes pose a conditional proposition even to monovalent target BSDs, since each business process requires resources to be available (i.e., there are no ideally efficacious processes, cf. Def. 1). As discussed in Section 3.2, assessment can be simplified by structuring conditional propositions into necessary and sufficient sub-conditions.

Example 3 (Necessary and Sufficient Sub-conditions). Again, consider the approval of loans. The availability of customer master data and the responsible manager both constitute necessary sub-conditions. Assuming that the customer's credit history is usually available in the data base, but may also have to be obtained manually, we have two sufficient sub-conditions: the described necessary sub-conditions plus the data base entry, and the described necessary sub-conditions plus the availability of a clerk for manual evaluation.²

Moreover, to achieve formal efficacy (cf. Def. 1), the environmental conditions resulting from a process model with respect to a target BSD must “encompass” the environmental conditions specified in the objective model. More precisely, each necessary sub-condition of the target BSD in the business objective should be a necessary sub-condition in the process model as well. Therefore, we discern between the *outer conditional environment* and the *inner conditional environment* defined by process and objective model, respectively. Figure 5 summarizes the constituents of the outer conditional environment. We use the *Referent Modeling Language* (RML) described in [23], since it provides a concise means of describing set relations. The concept of *target presumptions* will be illustrated in Section 6.

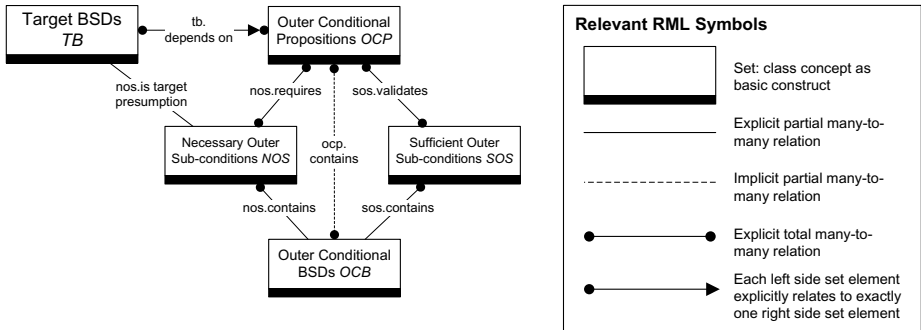


Fig. 5. Relating Target BSDs and the Outer Conditional Environment


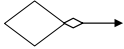
5 Efficacy-Aware Business Process Models

Business process modeling languages are mostly oriented at execution semantics of business processes, for instance because this is required for computerized workflow implementation. In terms of semantics, this requires modeling possible task sequences, but not the formalization of the impact on target BSDs or enactment preconditions (e.g., the availability of labor). We thus need to extend existing approaches towards *efficacy-aware process models*.

The Business Process Model and Notation (BPMN) constitutes a broadly applied language covering common process modeling concepts [3]. Since we aim at seamless integration with well-established concepts (cf. Figure 1), we use BPMN as a basis for extension instead of defining an entirely new formalism. Table 2 summarizes the additional terminology required. The meta-model presented in Figure 6 shows how the necessary terms are interrelated, and how they integrate with BPMN concepts.

In BPMN, the modeler is generally free with respect to the level of granularity regarding tasks and activities as atomic or aggregate constructs. In our context, however, we need to limit this degree of freedom to obtain stricter execution semantics. Accordingly, we require tasks to be enacted atomically, i.e., either

Table 2. Required Terminology

BPMN Terms	Efficacy-aware Meta-model Terms	Semantic Adaptations
Data objects 	Environmental elements: affecting and affected elements, target and conditional elements	Environmental elements replace data objects. From the perspective of the business process, they comprise the overlapping sub-sets of affecting elements (e.g., data fields altered) and affected elements (e.g., resources spent). From the perspective of the business objective, they comprise target elements and conditional elements (cf. Section 3.2). Both perspectives overlap. For example, a target element can be an affected element and an affecting element.
Conditions attached to split gateways 	Branches and branch-conditional BSDs	A branch is a sequence flow succeeding a conditional split gateway. Branch-conditional BSDs take up the concept presented in [11]. They are used to describe split gateway conditions by relating affecting elements to absolute or relative conditions (e.g., “A = 5” or “A < B”). Thus, branch-conditional BSDs represent environmental conditions that co-determine which tasks are enacted.
[none]	Task-requisite BSDs	The BSD concept is also used to describe enactment preconditions attached to tasks. Semantically, we assume that an enabled task is enacted if and only if <i>all</i> task-requisite BSDs are fulfilled. Task-requisite BSDs may relate to resources that just need to be available (e.g., “information system available = true”), or to resources actually spent (e.g., “working time available > 1h”).
[none]	State operations	State operations related to tasks are used to model effects on affected elements as functions (e.g., “A = A + B”). We assume that if a task is enacted, <i>all</i> related state operations are executed, and that state operations related to tasks are the only elements of business process models with an impact on affected elements.

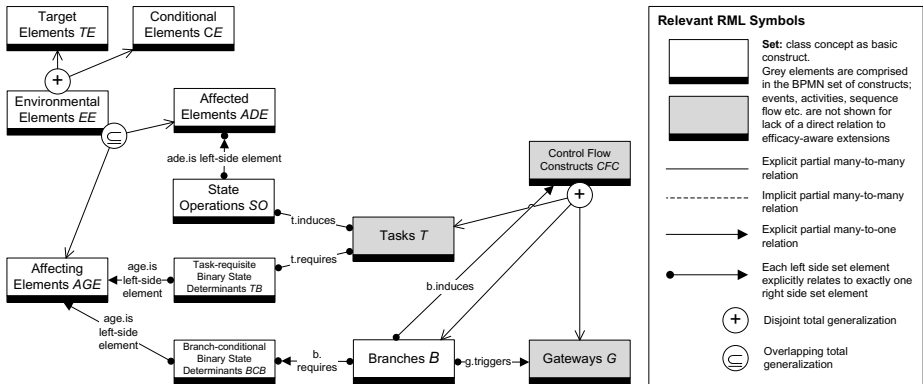


Fig. 6. Efficacy-aware Business Process Meta-model

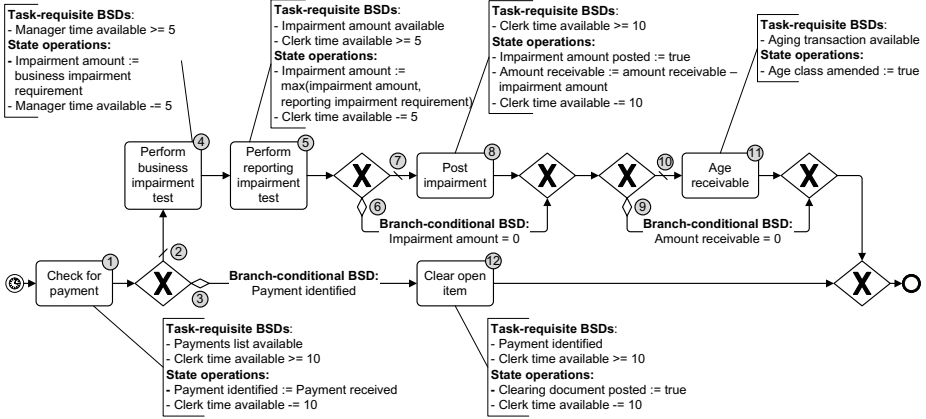


Fig. 7. Exemplary Business Process: Year-end Receivables Processing

in total or not at all. Thus, tasks do not have internal execution semantics. Trivially, this can be achieved by sufficiently refining tasks during modeling.

Figure 7 illustrates a process modeled in terms of BPMN and amended with additional information according to Table 2. Its semantics are described in Example 4.

Example 4 (Sample Business Process). Figure 7 depicts a sample process model which corresponds to the business objective model from Figure 4. Relevant control flow elements have been annotated with reference numbers 1-12.

Business objective and business process relate to the management of receivables during year-end closing. Receivables are first matched against unallocated payments (1). If payment has been identified (3), the receivable is cleared (12). Otherwise (2), it is assessed in an impairment test based on management’s appraisal (4) and formal criteria (5). If an impairment amount has been identified (7), the impairment is posted (8). If an open item remains (10), it is allocated to an age class (11). The latter task, for instance, can be enacted if it is enabled and the aging transaction is available (task-requisite BSD). If it is enacted, the age class is amended (state operation).

6 Efficacy Assessment Method and Sample Validation

Since our approach towards an efficacy-aware business process models extends BPMN with a small set of additional terms, we may assume the second effectiveness criterion (i.e., integration with common modeling languages) to be fulfilled. Accordingly, our validation focuses on the functional requirement of enabling to assess business process efficacy. Figure 5 shows which information must be available to allow assessing efficacy. This section discusses, by means of the sample

Table 3. Target BSDs, State Operations, and Control Flow Paths

Target BSD (cf. Fig. 4)	Relevant State Operation (Task No.)	Control Flow Path Alternatives (cf. Fig. 7 for reference numbers)
Clearing document posted	Clearing document posted = true (12)	(1-3-12)
Impairment document posted	Impairment document posted = true (8)	(1-2-4-5-7-8)
Impairment document NOT posted	Impairment document posted = true (8)	NOT (1-2-4-5-7-8)
Age class amended	Age class amended = true (11)	(1-2-4-5-7-8-10-11) OR (1-2-4-5-6-10-11)

process described in Example 4, how this information can be derived from an efficacy-aware business process model and used for efficacy assessment. It thus demonstrates a *method* as discussed in Section 2.

To enable efficacy assessment for a business process model, we require information about the outer conditional environment of target BSDs as described in Figure 5. This information is obtained by executing three steps presented in the following. The fourth step constitutes the actual efficacy assessment.

Step 1 (Matching Target BSDs, State Operations, and Control Flow Paths). State operations describe the actions carried out on environmental elements when enacting tasks (cf. Table 2), and are required to fulfil target BSDs. Table 3 therefore matches target BSDs against relevant state operations and possible control flow paths to enact the state operations.

Note that, for the third target BSD (*Impairment document NOT posted*), the relevant state operation *must not* be executed to fulfill the target BSD. This issue generally occurs for fully determinate bivalent target BSDs and for de-composed trivalent target BSDs (cf. Section 3.2).

Building relevant control flow paths necessitates traversing the process model. This is trivial for our simple example, but may get more complex in other cases. Respective algorithms are discussed in, for instance, [24]. Loops with an unspecified number of iterations mostly reflect sets of uniform target elements to be managed. As an example, consider lists of documents to be processed. In line with common modeling approaches [25], this issue is best addressed by using a corresponding structure of super- and sub-processes. Efficacy assessment is then executed separately for each level.

Step 2 (Consolidating Control Flow Paths). To determine the outer conditional environment required to fulfill a target BSD, we need to consolidate the BSDs comprised in relevant alternative control flow paths (cf. Table 3) considering the respective state operations. This can be achieved by “merging” subsequent control flow path elements until each relevant control path has been consolidated into one set of conditional BSDs. The required operations are shortly described in this step, but formalized in [26]. Two subsequent control flow elements are merged as follows:

- (a) Apply the state operations of the first element to the BSDs of the second element. This is necessary to consider that state operations of the first element might affect BSDs of the second element.
- (b) Merge the resulting BSDs with the first element's BSDs.
- (c) Merge the first element's with the second element's state operations.

This provides us with new sets of BSDs and state operations, which jointly describe a new *virtual control flow element*.

Example 5 (Control Flow Path Consolidation). The results of recursively following through the consolidation procedure for the first relevant control flow path of the *Age class amended* target BSD are presented in Figure 8. The top line depicts the relevant control flow path alternative extracted from the process model (cf. Figure 7). In the second line, the merge operation has been executed for the first two control flow elements. In this case, the respective sets of BSDs do not address common environmental elements. Accordingly, the branch-conditional BSD of (2) has simply been added to the merged set of BSDs of the new virtual control flow element. The third line shows the results of following through the merge procedure for the entire control flow path alternative, so that only one virtual control flow element remains. This element bundles all BSDs and state operations as though they would be enacted in a single task.

For more complex processes, the consolidation of control flow paths can be structured along sub-processes that occur multiple times. Virtual control flow elements can then be re-used. In our example, this applies to 1-2-4-5-7-8: this sub-path is relevant to both *Impairment document posted* and *Age class amended*. Note that parallel execution paths can be handled by using block-structured process models and recursively consolidating parallel paths into activities. As an additional consistency condition, this requires that the affected elements of neither parallel path are affecting elements of the other one (cf. Table 2).

Step 3 (Building Necessary and Sufficient Sub-conditions). Necessary and sufficient outer sub-conditions for a target BSD as defined in Figure 5 can now be derived according to a simple schema:

- Each set of merged BSDs of a consolidated control flow path constitutes a sufficient outer sub-condition since fulfillment of the BSDs is sufficient to enable the target BSD. Accordingly, there are as many sufficient outer sub-conditions as there are control flow path alternatives for a target BSD (cf. Table 3)
- Any merged BSD that occurs in each control flow alternative constitutes a necessary outer sub-condition. Note that, for the purpose of necessary outer sub-conditions, BSDs where the same set of affecting elements is covered in each control flow alternative are represented by their most “relaxed” form (in our case, this applies to the available clerk time).
- If the state operation inducing the target BSD has affecting elements, an additional necessary outer sub-condition is derived from the image function describing the operation's content (cf. [26, Def. 4]): the target presumption as included in

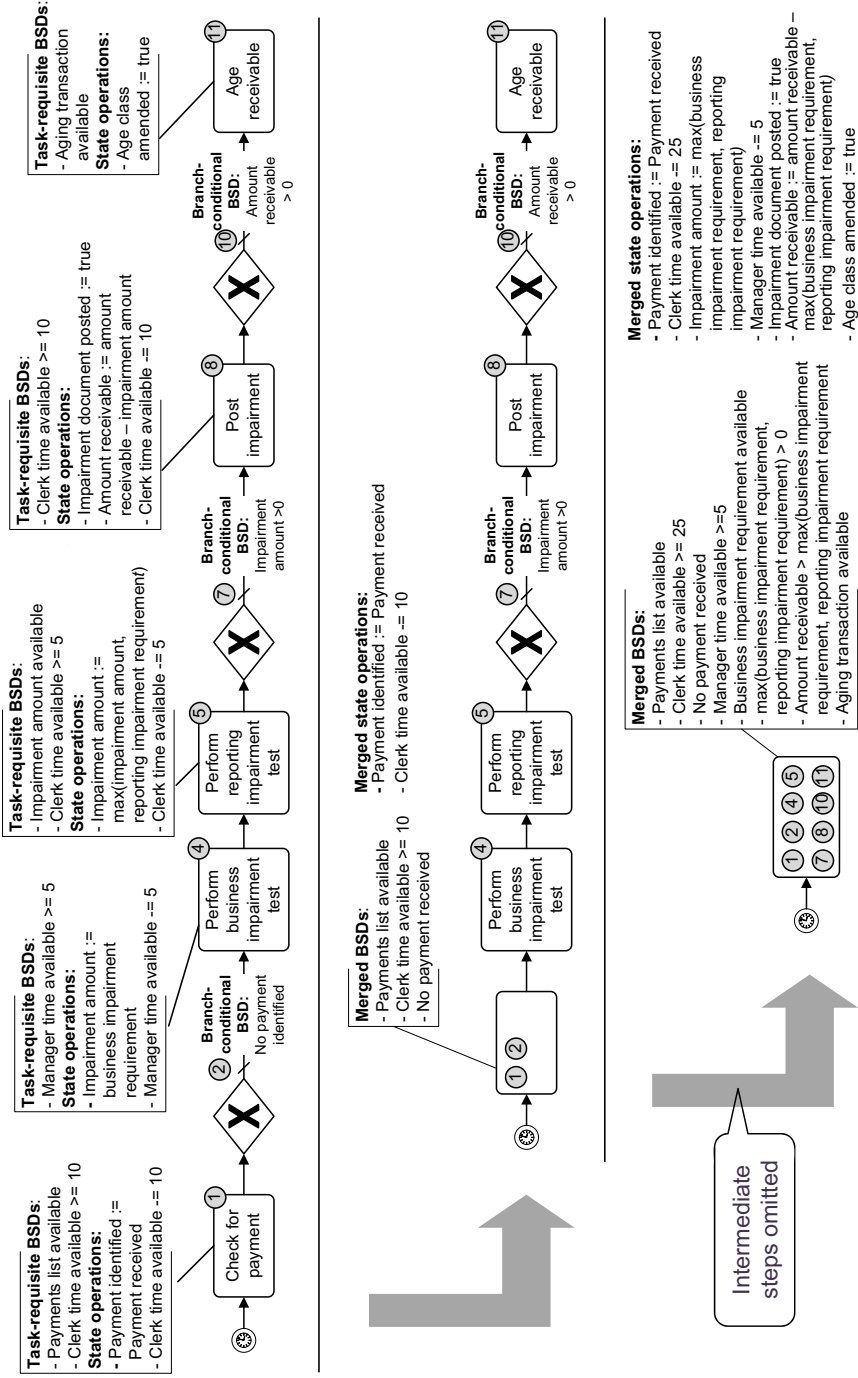


Fig. 8. Control Flow Path Consolidation Example (cf. Fig. 7)

Figure 5. To this end, we substitute the function’s affected element in the target BSD by the image function. The resulting term yields the target presumption. It represents environmental conditions not caused by control flow, but by the final state operation itself.

The resulting sub-conditions are then compared to the respective necessary sub-conditions of the target BSD as per the objective model. Results for *Age class amended* are shown in Table 4.

Table 4. Target BSDs and the Conditional Environment for *Age class amended*

Binary State Determinants	Outer Sub-conditions			Objective Model: Necessary Sub-conditions
	Sufficient: Path 1	Sufficient: Path 2	Necessary	
Payments list available	X	X	X	
Clerk time available ≥ 25		X		
Clerk time available ≥ 15		X	X	
Payment received = false	X	X	X	X
Manager time available ≥ 5	X	X	X	
Business impairment req'ment available	X	X	X	
$\max(\text{impairment requirements}) > 0$	X			
$\max(\text{impairment requirements}) = 0$		X		
Amount receivable $> \max(\text{imp. req's})$	X	X	X	
Aging transaction available	X	X	X	

Step 4 (Assessing Efficacy). To compare the outer conditional environment as defined by a process model to the conditional environment of a business objective, we must mainly consider the inconsistencies. According to Def. 1, Table 4 enables to conclude:

- Target BSDs included in the business objective but not covered by state operations signify that the business process is *not formally efficacious*, because the business process alone is not sufficient to fulfill all target BSDs. This is not the case in our example.
- Necessary sub-conditions of the business objective not covered by the process indicate that the business process is *not formally efficacious*, because it may induce target BSDs without considering relevant constraints. Again, this is not the case in our example.
- Necessary outer sub-conditions of the process model with regard to a Target BSD that do not correspond to necessary sub-conditions of the business objective indicate resources required by the business process to induce a target BSD. It needs to be judged whether these are considered as reasonable – the process may be *not fully efficacious* even if it is *formally efficacious*.

For our sample process, we can conclude that the process is formally efficacious. Whether it is fully efficacious will mainly depend on whether the associated requirements regarding available labor resources are deemed as reasonable by subject matter experts. Our sample validation has thus shown that our business

process meta-model (cf. Figure 6) enables efficacy assessment. As a next step, we might use the objective model and the assessment method to evaluate alternative implementation options for business process optimization.

7 Conclusion

In this paper, we built an approach towards efficacy-aware business process models based on the design science paradigm, insights on related work, and available results on business objectives modeling. We derived required terminology from functional requirements, and integrated the results into a meta-model extending BPMN. We described a method to assess the efficacy of a corresponding process model, and demonstrated the validity of our approach by application to a sample case, thus addressing our functional effectiveness criterion.

Together with the business objectives modeling approach presented in [11], the results presented yield the capabilities required for application scenarios described in Section 1: the ability to model business objectives independently from business processes, and the ability to formally assess efficacy. Seamless integration with established BPM methods is warranted. As an example of how application scenarios might be further pursued, reconsider our first scenario:

Scenario 1 (Automated Business Process Optimization). Formal efficacy assessment as demonstrated in Section 6 permits to determine whether process adaptations compromise business objective achievement. It thus becomes possible to automatically propose adaptations aimed at cost or time optimization and test their efficacy. Propositions might either be derived from empirically assessing “wasteful” execution paths, or based on random selection and subject to subsequent statistical assessment. A prospective approach would require preliminary determination of the efficacious scope of action. In this respect, the procedure of consolidating efficacy-aware process models (cf. Section 6) would have to be inverted.

In addition to adoption of the results presented into practical application scenarios, future work will also address integration of the business objectives modeling approach with requirements engineering frameworks such as KAOS [21]. Corresponding to the integration into the BPM field of knowledge, this will further improve the practical appeal of the approach.

References

1. Davenport, T.J., Short, J.E.: The new industrial engineering: Information technology and business process redesign. *Sloan Mgmt. Rev.* (4) (1990) 11–27
2. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. *Europ. J. of Inf. Syst.* **15**(1) (2006) 91–102
3. The Object Management Group: Business Process Model and Notation: Version 2.0 (2011) <http://www.omg.org/spec/BPMN/2.0>.
4. Scheer, A.W., Thomas, O., Adam, O.: Process Modeling Using Event-Driven Process Chains. In: *Process-aware Information Systems*. Wiley (2005) 119–145

5. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer (2012) to appear.
6. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. *J. Sw. Mnt. Ev. Res. Pract.* **22**(6-7) (2010) 519–546
7. Li, C., Reichert, M., Wombacher, A.: Mining business process variants: Challenges, scenarios, algorithms. *Data & Knowl. Eng.* **70**(5) (2011) 409–434
8. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. *Comput. Ind.* **62**(5) (2011) 467–486
9. Camp, R.C.: Benchmarking: the search for industry best practices that lead to superior performance. Quality Press (1989)
10. IDS Scheer: Process intelligence white paper: What is process intelligence? (2009) <http://www.process-intelligence.com>.
11. Lohrmann, M., Reichert, M.: Modeling business objectives. In: Proc. 4th S-BPM ONE – Scientific Research. LNBIP 104 (2012) 106–126
12. Simon, H.A.: The Sciences of the Artificial. 3rd edn. MIT Press (1996)
13. March, S.T., Smith, G.F.: Design and natural science research on information technology. *Decis. Support Syst.* **15**(4) (1995) 251–266
14. Kueng, P., Kawalek, P.: Goal-based business process models: creation and evaluation. *Bus. Process Manag. J.* **3**(1) (1997) 17–38
15. Neiger, D., Churilov, L.: Goal-oriented business process modeling with EPCs and value-focused thinking. In: Proc. 2nd BPM. LNCS 3080 (2004) 98–115
16. Markovic, I., Kowalkiewicz, M.: Linking business goals to process models in semantic business process modeling. In: Proc. 12th EDOC, IEEE (2008) 332–338
17. Soffer, P., Wand, Y.: On the notion of soft-goals in business process modeling. *Bus. Process Manag. J.* **11**(6) (2005) 663–679
18. Lin, Y., Sølvsberg, A.: Goal annotation of process models for semantic enrichment of process knowledge. In: Proc. 19th CAiSE. LNCS 4495 (2007) 355–369
19. Engelsman, W., Quartel, D., Jonkers, H., van Sinderen, M.: Extending enterprise architecture modelling with business goals and requirements. *Ent. Inf. Sys.* **5**(1) (2011) 9–36
20. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proc. 3rd Int'l Symp. on Requirements Engineering, IEEE (1997) 226–235
21. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. *Sc. of Comp. Programming* **20**(1-2) (1993) 3–50
22. 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: Proc. CAiSE'10 Forum. LNBIP 72 (2010) 76–91
23. Sølvsberg, A.: Data and what they refer to. In: Conceptual Modeling. Springer (1999) 211–226
24. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 3rd edn. MIT Press (2009)
25. Weske, M.: Business Process Management. Springer (2007)
26. Lohrmann, M., Reichert, M.: Formalizing concepts for efficacy-aware business process modeling. Technical Report UIB-2012-05, Databases and Information Systems Institute, Ulm University (2012)

Automated Resource Assignment in BPMN Models Using RACI Matrices*

Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés

Universidad de Sevilla, Spain
{cristinacabanillas,resinas,aruiz}@us.es

Abstract. Organizations need to manage the responsibility of their employees with respect to all the activities that are daily carried out within them. Process-oriented organizations need to do it, in addition, in accordance to the business processes their members participate in. However, powerful mechanisms to manage responsibility in combination with business processes are missing in current modelling notations, usually limited to indicating who is in charge of undertaking the activities. RACI matrices, on the contrary, were specifically conceived to provide responsibility management information. They enable the specification of the level of responsibility each human resource has with regard to each activity carried out in a company, ranging from the performer of the work to the resource that must approve it or receive certain notifications. In this paper, we propose the use of RACI matrices together with business process models to manage human resource responsibilities in processes. Focused on a concrete type of RACI matrices, called RASCI, we introduce a novel approach to automatically generate a BPMN model with RASCI information given a BPMN model that does not handle resources, and a RASCI matrix. The resulting model is BPMN-compliant and, thus, it is ready to be executed in existing business process management systems. With this approach, the assignment of responsibilities and the management of processes can be designed separately, while being executed together.

Keywords: Responsibility management, RACI matrix, RACI-aware BPMN model, RASCI sub-process, RASCI meta model.

1 Introduction

Organizations need to manage the assignment of responsibilities to their members with respect to the activities that must be carried out within them. This means that, in order to have an action plan of the work performed by every member, not only associating functions to each member of the organization is necessary, but providing a way to organize and display these responsibility assignments is required too. This can be done by means of a Responsibility Assignment Matrix (RAM), also known as RACI matrix or Linear Responsibility

* This work has been partially supported by the European Commission (FEDER), Spanish Government under project SETI (TIN2009-07366); and projects THEOS (TIC-5906) and ISABEL (TIC-2533) funded by the Andalusian Local Government.

Chart (LRC) [1]. Such matrices provide a way to plan, organize and coordinate work, and consist of assigning different degrees of responsibility for each activity developed in the company to the members of an organization, such as who is in charge of undertaking the activity and who must be informed once the action is complete [2]. Several variants extending the functions considered in traditional RACI matrices have appeared (e.g., RASCI matrices).

Besides, process-oriented organizations need to organize and control the activities that are carried out in the company. This is typically done with business process (BP) models that represent the control flow of the activities, together with other perspectives of the process including data and resource management. With regard to resources, most BP modelling notations existing at present allow only the specification of who is in charge of performing the activities of the BP, which is short scope with respect to all the issues involved in human resource or responsibility management (i.e. with respect to RACI's expressiveness). For instance, the de-facto standard for BP modelling, Business Process Modelling Notation (BPMN) [3], has this limitation.

Thus, there is an important distance between the responsibility information that should be managed in an organization, and the one that is actually handled with current BP modelling notations. Furthermore, given the increasing interest of organizations to work with RACI matrices in combination with BPs [4], it is evident that it is necessary either the improvement of the responsibility management capabilities of current BP modelling notations, or the development of a mechanism to enrich the resource-related information contained in BP models.

In this paper, we address this problem and work in the latter direction. Specifically, we introduce a novel approach to generate a BP model with complete responsibility information (i.e. a RACI-aware BP model) from a resource-unaware BP model and a RACI matrix. Our approach tackles two main problems. On the one hand, getting BP models with all the information required to be able to execute them implies generating very accurate resource assignments for the activities of the BP. However, this cannot be done directly due to the high level at which RACI matrices and BP models are built. To overcome this issue, it is necessary to provide extra information for the RACI matrix. In particular, some information about the context in which the process is going to be executed, and some restrictions to be considered, have to be indicated. We call this extra information *binding information*. On the other hand, the control flow of the BP model must be changed according to the functions defined in RACI. To this end, we propose a collection of transformations to model the information of a modality of RACI matrix called RASCI, together with the binding information, into BPMN models. The transformations are as generic as possible and can be automated, and the resulting BP model has no intrusive information about RASCI at first sight. Indeed, RASCI information is modelled at sub-process level¹.

Notice that RASCI information could actually be modelled in BPMN with no need of our approach in an ad-hoc manner. The use of swimlanes constitute a possible mechanism to narrow this gap between RASCI and BPs. We could

¹ Please notice that in this paper we may use terms RACI and RASCI interchangeably.

use them to represent organizational roles and place the proper activities in the proper lanes to comply with the matrix. We worked on that line last year, introducing a collection of *RASCI patterns* and an extension for BPMN 2.0 [3] to allow the modelling of such patterns [5]. However, we realized that proceeding that way has several problems. On the one hand, RASCI functions for each activity are disseminated in the BP model, while they are actually part of the work carried out for a single activity. On the other hand, the resulting BPMN model may unnecessarily become very large and, consequently, difficult to read and understand, due to the increase of lanes. Also, binding information cannot be introduced with this approach based on swimlanes. Finally, keeping the information of both elements (a RASCI matrix and its associated BP model) consistent is difficult, since the modifications performed on one element should be performed on the other as well. The replication of the information thus derives in a synchronization problem. Therefore, the approach we present in this paper has the following advantages with respect to the previous work:

- The resulting BPMN model is complete from the viewpoint of responsibility management with respect to RASCI matrices. In addition, its appearance is very similar of the initial BP model.
- The output BPMN model is ready to be executed in current Business Process Management Systems (BPMSs), provided that they support the allocation of resources to tasks.
- Synchronization of the BP model and the matrix can be performed automatically by carrying out the transformations when the RASCI matrix and/or the binding information change.
- There is a decoupling of BP management and resource management at design time, but they can be automatically mixed together to be executed in combination at run time.

This paper is structured as follows. Section 2 introduces RACI matrices and their use with BP models. Section 3 gives details about the type of binding information required to complement RASCI information. Section 4 describes the meta model that represents the whole application scenario. Section 5 details our transformation-based approach together with some examples. A prototype of the proposal is briefly described in Section 6. Then, some related work is summarised in Section 7, and finally, conclusions drawn from this work and some future work are presented in Section 8.

2 RACI Matrices

RACI matrices constitute a mechanism to represent the assignment of responsibility of the members of an organization. In their standard modality, they are utilised to associate activities with (human) resources, typically by using the organizational roles (e.g. Project Coordinator, Sales Manager) the members of an organization play within the company or given a specific context (e.g. in a specific project or area) [1]. Figure 1 illustrates an example of RACI matrix. The

rows represent *activities* undertaken in a company, the columns of the matrix are (*human*) *resources*, and each cell contains zero or more *RACI initials* indicating the type of responsibility of *such a* resource on *such an* activity. As aforementioned, resources normally come in the form of *organizational roles*, as shown in the table. However, depending on the company, resources may be represented at different levels. For instance: (i) small companies could opt for using persons directly in each column; or (ii) at a very high level we could find RACI matrices in which each column would refer to specific organizational units. In this paper, we are using the standard way, i.e., columns represent organizational roles.

The initial in the cells are different functions, called *roles* in RACI²:

- *Responsible (R)*: person who must perform the work, responsible for the activity until the work is finished and approved by an accountable. There is typically only one person responsible for an activity.
- *Accountable - also Approver or Final Approving Authority - (A)*: person who must approve the work performed by the person responsible for an activity, and who becomes responsible for it after approval. There must be one and only one accountable for each activity.
- *Consulted - sometimes Counsel - (C)*: this role involves the people whose opinion is sought while performing the work, and with whom there is two-way communication.
- *Informed (I)*: person who is kept up-to-date about the progress of an activity and/or the results of the work, and with whom there is just one-way communication. There may be more than one informed person for an activity.

There are several variants of the original version of RACI matrices. Some are based on extending the number of RACI roles to be considered for every activity, e.g, RASCI or RACI-VS. Others give different meanings to the RACI initials. In this paper, we build on RASCI matrices because they use a function that

Table 1. RASCI matrix for the process at pool *ISA Group* of Figure 1

	Project's PhD Student	PhD Thesis Supervisor	Project Coordinator	Project's Administrative Assistant	Research Group's Clerk
Submit Paper	R/A				
Fill Travel Authorization	R		A/C		
Sign Travel Authorization	I		R/A		
Send Travel Authorization	I				R/A
Register at Conference	R/A	I	C/I	I	
Make Reservations	R/A	C	C	C/I	S

² We will use the term *RACI role(s)* to differentiate them from organizational roles.

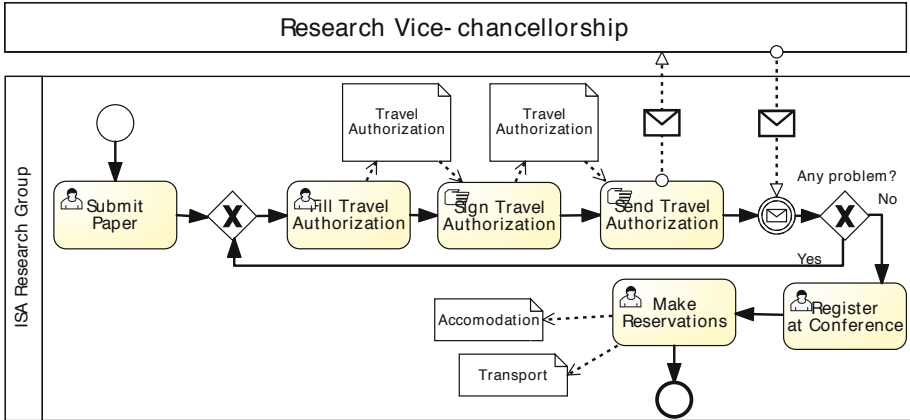


Fig. 1. Conference Travel Management Process

may be interesting specially to IT organizations, where work or tasks needed to complete an activity can usually be delegated to other people. RASCI matrices involve the aforementioned RACI roles together with RASCI role *Support*:

- *Support (S)*: people who may assist in completing an activity, i.e., the person in charge can delegate work to them. Unlike *Consulted*, who may provide input information to the activity (i.e., information helpful to perform some work), *Support* will actively contribute in the completion of the activity.

A process-oriented organization could build one RASCI matrix for each BP used in the company. The matrix would list its activities and the organizational roles that participate in them for each RASCI role. Figure 1 shows a BPMN diagram representing a *collaboration* between two BPs: one BP at pool *Research Vice-chancellorship* and another one at pool *ISA Research Group*³. It illustrates a simplified version of the procedure to manage the trip to a conference, according to the rules of the University of Seville. We are going to focus on the BP carried out at pool *ISA Research Group*. As can be seen in Table 1, the activities of the matrix are exactly the BP activities in the model.

In outline, the process works as follows. It starts with the submission of the Camera Ready version of an accepted paper by the PhD student whose paper has been accepted for publication. After that, *that* student fills in an authorization request to attend and present the paper at the conference. The coordinator of *the project that will finance the trip expenses* must sign the authorization and inform the student when it is done. The clerk of *the research group the PhD student belongs to* is in charge of delivering the form for approval. In absence of problems, *the student* must register at the conference and inform *his/her* PhD thesis's

³ We remind the reader that in BPMN a process takes place within a single pool. Diagrams with two or more pools, in which messages between the pools are exchanged, are called collaborations. All the process-oriented concepts used in this paper are taken from BPMN 2.0 [3].

supervisor, as well as *the project coordinator* and *the administrative assistant of the project*. Finally, *the PhD student* books the tickets needed, assisted by the clerk of *his/her* research group, if required.

However, note that the previous description of the process contains some nuances that are not incorporated in the RASCI matrix. In particular, specific information about the individuals that have to be assigned to the RASCI roles (e.g. the same PhD student during a single execution), or the context within which it has to be done (e.g. the project coordinator of a specific project), is missing in the matrix. The reason is that, as an organization, we aim at modelling BPs that can be applied in different areas of the company (e.g. the same BP may represent how to proceed with the application for a job, regardless of the specific department in which the job is being offered). Similarly, RASCI matrices must be as generic as possible, avoiding the replication of information due to the application of a BP to those different areas or “contexts”. This flexibility in the design of BPs and RASCI matrices is important, but means a problem when trying to automate the combination of both elements, and the generation of the corresponding resource assignments in the resulting BP model. How to solve it? In order to overcome this issue, some extra information must be provided, which we have called *binding information*.

3 Binding Information for Resource Assignment

Binding information complements the resource information provided by RASCI matrices in order to enable automated BP resource assignment. This information can be mainly of two types:

- *Organizational unit context*. Indicating only the organizational role for a RASCI role is usually insufficient, since it does not limit the context in which the BP is going to be run. Let us see it with an example. According to the RASCI matrix in Table 1, role *Project Coordinator* is responsible for activity *Sign Travel Authorization*. However, a project coordinator can sign forms only for the project(s) he/she coordinates, so *not any* project coordinator can perform this task in *any* execution of the process. Therefore, it is necessary to indicate either directly the concrete data required (e.g. name of the project we refer to in the current process instance), or *where* this information can be found, e.g. in our BP the name of the project appears in the *Travel Authorization* form filled in by the student (cf. Figure 1).
- *Additional restrictions*. Other information may be necessary in order to constrain the set of people that can be assigned certain RASCI role. For example, sometimes it is essential that two activities of the same BP be carried out not only by the same organizational role, but by the same person, i.e. Binding of Duties (BoD). In the scenario at hand the same PhD student submits the paper and fills in the travel authorization form. Other times, exactly the opposite may be necessary, i.e. Segregation of Duties (SoD), in order to avoid conflicts of interests between individuals. Restrictions concerning specific skills required to carry out a certain task may also be common.

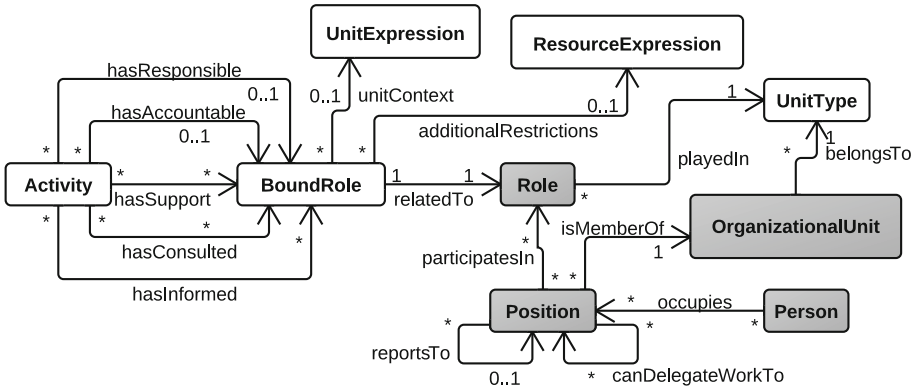


Fig. 2. RASCI meta model with binding information

It is important to let the user define all these additional restrictions that need to be taken into account for resource allocation at run time.

Notice that binding information must be given at RASCI role level, that is, for each RASCI role participating in each BP activity of the matrix.

4 RASCI Meta Model with Binding Information

Taking all the aforementioned aspects into consideration, the meta model of a RASCI matrix with binding information can be defined as shown in Figure 2.

- Class *Activity* represents the activities of the BP the RASCI matrix is associated to, and in which we aim to insert the responsibility-related information necessary to make it work according to the matrix.
- Five relations between *Activity* and *BoundRole* represent the five RASCI roles to be distributed among the members of the organization. The role is bound because it may have binding information associated. The expressions to specify the organizational unit context and any other additional restriction can be defined in classes *UnitExpression* and *ResourceExpression*, respectively.

We have added the following conditions between *Activity* and *BoundRole* in order to define some existence relations between RASCI roles. We use Object Constraint Language (OCL)⁴ to specify the following invariants:

- When there is not a resource responsible for an activity (e.g. an automatic task executed directly by the system), the other RASCI roles cannot exist, except RASCI role I. We exclude the information function (I) because there may be automatic activities in the process consisting of a notification message automatically sent by the system, but whose destination can be a resource indicated in the RASCI matrix.

⁴ <http://www.omg.org/spec/OCL/2.0/>

```

context Activity inv:
if self.hasResponsible->isEmpty()
then
self.hasAccountable->isEmpty() and
self.hasSupport->isEmpty() and
self.hasConsulted->isEmpty()
endif

```

- When RASCI role R is in, then there must be an accountable, since this role is mandatory according to RACI definition⁵ (cf. Section 2).

```

context Activity inv:
if not(self.hasResponsible->isEmpty())
then not(self.hasAccountable->isEmpty())
endif

```

- The classes in gray in the figure represent the part of the organizational meta model described by Russell et al. [6] we have relied on to build the structure of an organization. We have added class *UnitType* for the sake of understanding. In particular, each *BoundRole* is associated to a *Role* of the organizational structure of the company. However, as aforementioned, a person has a role in the context of an organizational unit (e.g. coordinator of a certain project, or research assistant in a specific research group). This relation is modelled by means of class *Position*. A position, thus, represents a collection of roles in one specific organizational unit.

Let us take as example *Activity Sign Travel Authorization* of our use case (cf. BP model in Figure 1 and RASCI matrix in Table 1) to exemplify the RASCI meta model shown in Figure 2. The organizational roles that participate in this activity (i.e. *Project's PhD Student* and *Project Coordinator*) fit in class *Role*. As aforementioned, every role is related to an organizational unit. In this case, it is a *project* (class *UnitType*) called *THEOS* (class *OrganizationalUnit*). There is a positional hierarchy for each organizational unit. For project THEOS it is shown in Figure 3. It has six positions, occupied by seven persons. The relation *participatesIn* of the meta model is outlined in the table attached to the figure. For the rest of organizational units in the company, a similar model is required.

The rest of classes of the RASCI meta model (i.e. *BoundRole*, *UnitExpression* and *ResourceExpression*) are specified at cell level. For RASCI roles R and A, *BoundRole* contains the assignment to role *Project Coordinator*, together with a *UnitExpression* stating that the name of the project can be found in file *Travel Authorization* (handled in the process) during execution. For RASCI role I, *BoundRole* is role *Project's PhD Student* plus a *ResourceExpression* indicating that it has to be the same person who performed activity *Submit Paper*. The language used to specify the binding information, and thus, the whole resource

⁵ The lack of A in the table is interpreted as R and A being assigned to the same role.

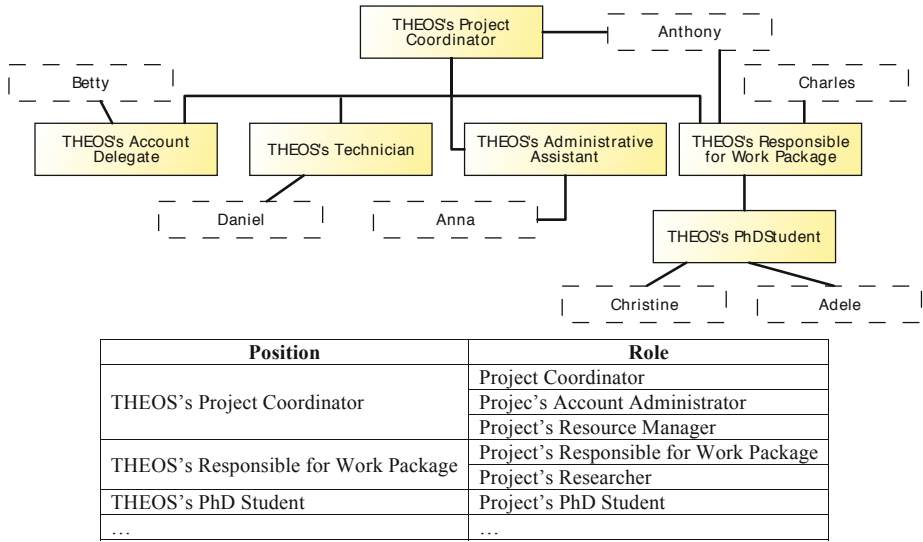


Fig. 3. Excerpt of the organizational model of ISA Group from a project perspective

assignment expression for the *BoundRole*, depends on the language supported by the BP modelling notation used. For instance, BPMN uses XPath⁶ by default to define resource assignments.

5 Using a RASCI Matrix to Specify Resource Assignments in BPMN Models

We already know all the information required to be able to automatically insert RASCI information into a BP model in order to make it compliant with the resource assignments of the matrix. As stated at the beginning of this paper, BP modelling languages existing nowadays do not provide a explicit way to model this RASCI-related information within BP models, being limited to the assignment referring to RASCI role Responsible (R) in most cases. Nevertheless, notations such as BPMN 2.0 offer extension mechanisms that may allow the introduction of any type of information into the models and, thus, we can make use of those features to add RASCI information [5].

We are working with BPMN [3] because it is the de-facto standard for process modelling, and because its extension capabilities are sufficient to enable the addition of the RASCI information we need to insert into the process models.

In the following, we introduce our approach to make a BPMN model RASCI-aware in a generic and automatable way. Furthermore, the output BP after the transformation from RASCI to BPMN is BPMN-compliant and has the required information to be executed in existing BPMSs.

⁶ <http://www.w3.org/TR/xpath/>

5.1 Resource Assignment Expressions in BPMN 2.0

Although swimlanes seem an easy and quick way to assign resources to the activities of a BPMN model, they are not a convenient form to do it. The problem basically relies on the lack of specific semantics for pools and lanes, which makes them remain an element for pure visual organization of the process elements in the model, as stated in the BPMN specification [3]. BPMN actually manages resources at activity level, using by default XPath expressions to specify resource assignments, although it permits the use of other languages. Making use of that feature, in this paper we use Resource Assignment Language (RAL) as an alternative to XPath to build resource assignment expressions.

RAL is a Domain Specific Language (DSL) specifically developed to express resource assignments in BP activities [7]. It is based on the same organizational meta model we used in Figure 2, which was defined by Russell et al. as basis to describe the so-called Workflow Resource Patterns (WRPs)[8]. These are a collection of patterns aimed at capturing the various ways in which resources can be handled in workflows (WFs). RAL expressions range from very simple assignments based on specific individuals of the company, to complex assignments containing access-control constraints (e.g. SoD) between activities, as well as compound expressions. For instance:

RAL 1: IS Anna

RAL 2: NOT (IS PERSON WHO DID ACTIVITY SubmitPaper)

RAL 3: (HAS ROLE ProjectCoordinator) OR (HAS UNIT ResearchGroup)

Therefore, RAL allows expressing role-based assignments to specify the direct assignments extracted from RASCI matrices, as well as binding information:

- *Organizational unit context.* To state that one RASCI role has to be performed by an organizational role within a *concrete* organizational unit, we could use RAL expression HAS UNIT UnitName, or expression HAS UNIT IN DATA FIELD PathToData.DataField in case the information must be retrieved from a file stored in the Information System (IS) used in the company. Notice that the specific way to access data stored in the IS depends on the implementation of the BPMS where the process is used, and so is the way to express the path to the required file.
- *Additional restrictions.* RAL offers expressions to specify many types of constraints, such as expression IS PERSON WHO DID ACTIVITY ActName to indicate BoD of a RASCI role with respect to the performer of another activity, i.e. both RASCI roles have to be allocated to the same person. Similarly, the negation of the previous expression can be used to define SoD. Expressions such as SHARES SOME ROLE WITH PersonName and HAS CAPABILITY CapabilityName allow the specification of other kinds of restrictions.

The reasons why we use RAL instead of XPath are:

- Even though XPath is a standard, it is barely supported by current BPMSs, which usually implement resource assignments in an ad-hoc fashion.

- XPath is not conceived specifically for resources, so it is difficult to use it to define resource assignment expressions.
- Derived from the previous point, XPath does not allow to express some binding information such as SoD constraints or skill-based restrictions. We refer the reader to [7] for further information about how to use RAL with BPMN 2.0.

Besides, RAL’s formal semantics based on Description Logics (DLs) [9] provides it with powerful analysis capabilities, which could be useful in case we subsequently wanted to extract and analyse RASCI information included in the resulting BP model [10]. This analysis consists of answering questions related to how resources are being managed in a BP. For instance, in a RACI scenario, we could analyse the BPMN model resulting from our approach to check: (i) whether there can be any person responsible (or accountable) for all the activities of the process after allocation; or (ii) all the activities in which some person may participate somehow (with any RASCI role); among others. The design-time analysis of RAL expressions and its implementation within BPMN was described in [10]. Run-time analysis has already been developed and is being tested as part of a Business Process Management System (BPMS) called Activiti⁷.

5.2 Generation of RASCI-Aware BPMN Models

In this section we introduce a collection of transformations that can be used to include information coming from a RASCI matrix (plus binding data) in a resource-unaware BPMN model. The BP activities that appear in the RASCI matrix are changed into a sub-process with the name of the activity. All the RASCI information will be contained in the sub-process. We will sometimes refer to such a sub-process as *RASCI sub-process*. Within it, for each RASCI role it is necessary to indicate:

- The control flow elements required, with the *name convention pattern* we will use to make the transformation as automatic and generic as possible.
- The proper resource assignment expression associated to each new task. This expression comes from class *BoundRole* of the RASCI meta model (cf. Figure 2). Using RAL, the expression will be (i) (HAS ROLE Role IN Unit_In_UnitExpression) AND (ResourceExpression), if there is a *Unit-Expression*; (ii) (HAS ROLE Role) AND (ResourceExpression), otherwise.

We assume that there is only one person responsible and one accountable for each activity (cf. meta model in Figure 2). Furthermore, the approval action (RASCI role A) takes place after the completion of the work developed for the activity, and only then the notification action (RASCI role I) can be performed. We could opt for a different order of RASCI roles or, even, for allowing them at different phases of the activity life cycle (for instance, to inform also before the start of the task or during execution). However, in the latter case, changes should be made in the RASCI matrix, and it is out of the scope of this paper.

⁷ <http://activiti.org/>

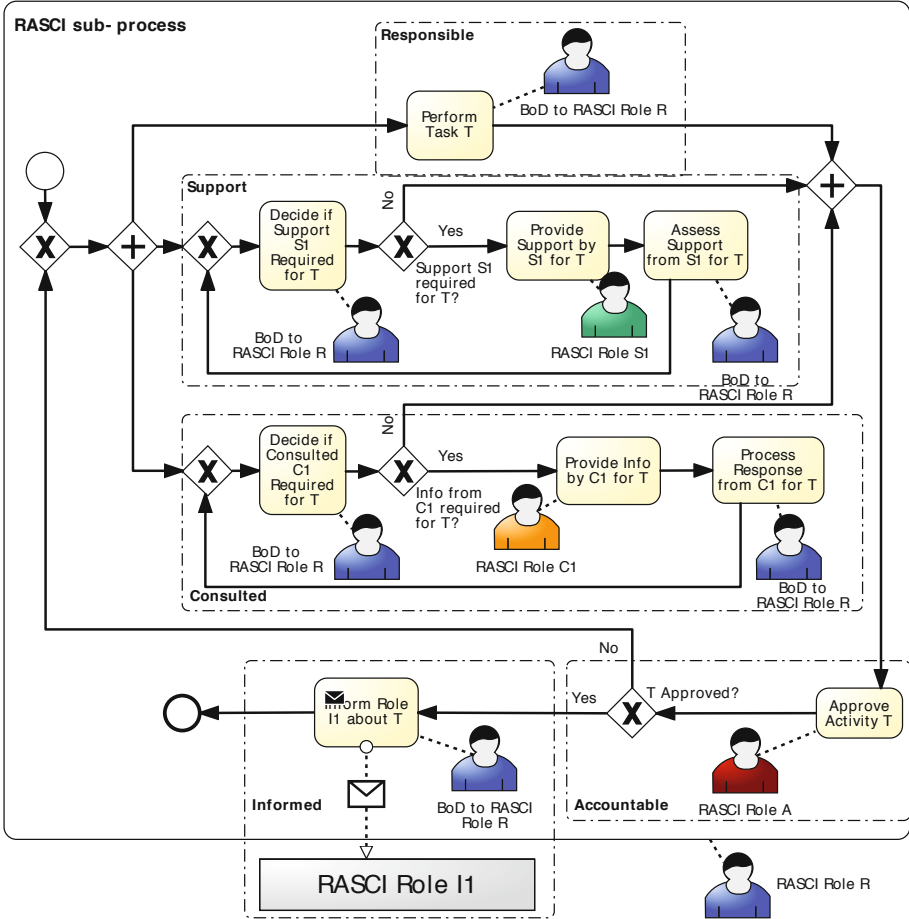


Fig. 4. Overview of a RASCI sub-process

The overview of a RASCI sub-process is depicted in Figure 4. BPMN groups define the process fragments related to RASCI roles. In case a RASCI role does not participate in the activity, the corresponding process fragment will be omitted in the sub-process. If, on the contrary, there are several roles performing a RASCI role, the associated process fragment will be added for everyone of them. We will use activity *Register at Conference* of our RASCI matrix (c.f. Table 1) to explain the transformations. Figure 5 shows the RASCI sub-process for it.

Responsible (R). This is the only RASCI role whose resource assignment expression is associated to the RASCI sub-process itself, i.e. for activity *Register at Conference*, the new sub-process has the following RAL expression: (HAS ROLE ProjectsPhDStudent) AND (IS PERSON WHO DID ACTIVITY SubmitPaper).

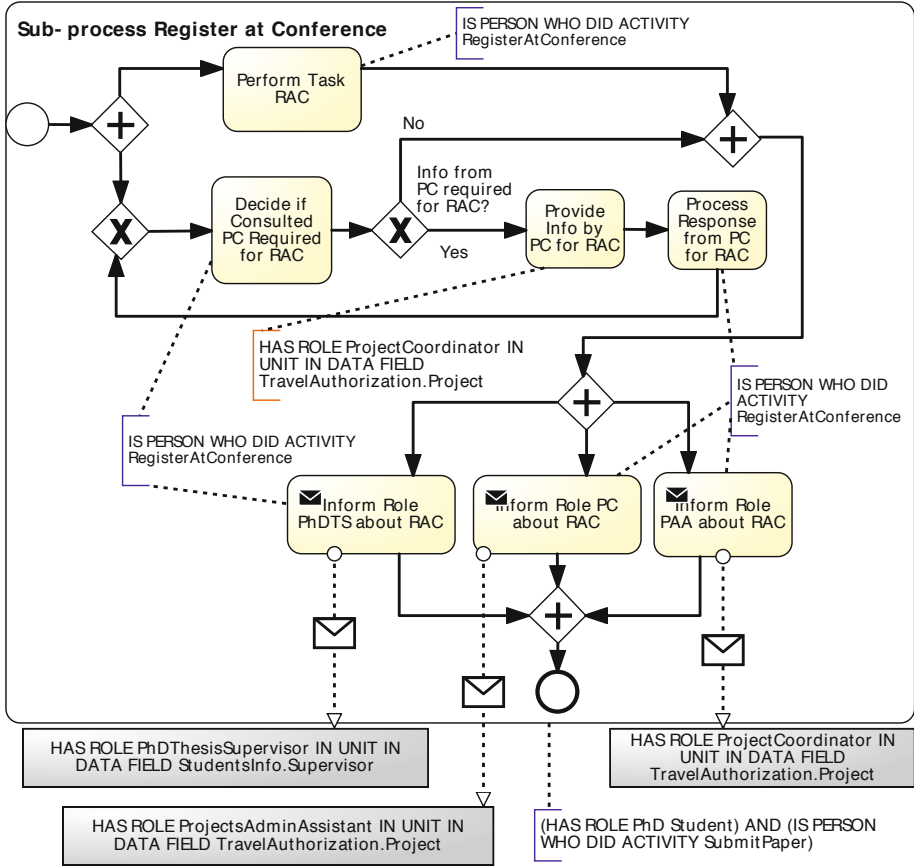


Fig. 5. RASCI sub-process for activity *Register at Conference* of the BP in Figure 1. Acronyms have been given for activities and roles for the sake of visualization.

Nonetheless, task *Perform Task ActivityName* is introduced in the RASCI sub-process to represent the actual work to be completed for the activity. This task is directly assigned to the performer of the sub-process, i.e. the RAL expression for task *Perform Task Register at Conference* is *IS PERSON WHO DID ACTIVITY RegisterAtConference*. This allows every element within the subprocess to make reference to the performer of the activity being sure that there is already an allocated performer. Note that if the activity itself were a sub-process, then *Perform Task Register at Conference* would be that sub-process.

Accountable (A). To model this RASCI role we insert a new task into the RASCI sub-process named *Approve Activity ActivityName*, in charge of approving the work developed in the activity at hand. Moreover, we have to add the control flow required to go back to the beginning of the sub-process in case the activity was not approved, by means of an *XOR join* gateway. The assignment expression of RASCI role A will be assigned to the new task.

This process fragment can be omitted only if R and A are assigned to the same organizational role in the RASCI matrix, and the binding information for A consists of a BoD with respect to R (i.e. BoD between A and R). For size reasons in Figure 5, we assume this is satisfied in our example activity. Note that if the previous condition is met and no other RASCI role participates in the activity, then the whole RASCI sub-process can be omitted and, thus, the result of the transformation is the same initial activity with the resource assignment expression corresponding to R.

Support (S). Inserting this RASCI role is not as straightforward for several reasons: (i) support is not mandatory. It is a decision of the person in charge of the task whether support is required to complete the work; (ii) it is said nowhere that support cannot be requested more than once to the same organizational role associated to RASCI role S in the matrix; (iii) it is inherent to term support that the work performed by the person “external” to the task must be evaluated by the resource in charge of the task (R) in order to decide whether the goal of the support has been achieved and/or whether more support is required; and (iv) it is not evident at which moment in task execution, support can be requested. In this sense, we have to make some decisions. So, we propose the control flow structure depicted in Figure 4 for RASCI role S, basically composed of tasks *Decide if Support Role Required for ActivityName*, *Provide Support by Role for ActivityName* and *Assess Support from Role for ActivityName*, and a couple of XOR gateways. The task targeted at providing the required support is assigned to the organizational role performing RASCI role S in the matrix with the appropriate resource assignment expression. The rest of new tasks belong to the person that is responsible for the activity. Thus, the resource assignment expression for these tasks is a BoD constraint in RAL: `IS PERSON WHO DID ACTIVITY ActivityName`. According to our RASCI matrix, activity *Register at Conference* does not need support.

Consulted (C). The translation of this RASCI role into BPMN language is very similar to RASCI role S. As depicted in the figure, the structure introduced in the sub-process is the same, as well as the considerations to be made. The only difference between the application of the two RASCI roles is in the name of the tasks involved, and in the semantics of tasks *Provide Support by Role for ActivityName* and *Provide Info by Role for ActivityName*. The person in charge of the latter is not so much compromised with the global activity, as his/her involvement is limited to providing certain information. For activity *Register at Conference*, the RAL expression associated to the task performed by the project coordinator, could be `HAS ROLE ProjectCoordinator IN UNIT IN DATA FIELD TravelAuthorization.Project`. For the rest of tasks of this process fragment, the associated RAL expression is `IS PERSON WHO DID ACTIVITY RegisterAtConference` (cf. Figure 5).

Informed (I). This RASCI role has a very special difference with respect to the others. The organizational role indicated in the matrix is the target person of

the notification action, not the performer like in the rest of cases. The problem is that in BPMN we do not have a way to specify the resource “affected” by a task. However, there is a mechanism based on message interchange to communicate information to people working on other processes. The key point here is whether the informed person can be considered an external participant or not. In case that role does not participate in any other activity of the BP, it is undoubtedly somebody external to the process. Thus, we could use messages to send the notification. Otherwise, that person may have his own assigned tasks in the process, and we do not have a way to notify something to that person without interrupting his work flow. We believe that, given the RACI definition for I, it is reasonable to consider it an external participant of the process under any circumstances, due to the absence of collaboration from his part. Independently of his responsibilities with respect to other BP activities, for *that* activity in question he is a target, not an executor. Therefore, we will introduce task *Inform Role about ActivityName* to represent an activity that sends a message to a collapsed pool representing RASCI role I (cf. Figure 4). This task is assigned to the person in charge of the main activity, like in RASCI roles S and C. For activity *Register at Conference*, there are three informed people. The control flow and the RAL expressions for them are directly shown in Figure 5.

Following these rules we can convert the initial BP model into another one with the information necessary to implement RASCI in our organization. Furthermore, the resulting model is very clean from the visualization perspective, in the sense that it is very similar to the initial one. So, the overall understandability of the initial process is maintained. The real complexity related to the RASCI information is found only when the RASCI sub-processes are opened.

6 Implementation

An overview of the prototype we have developed for the approach is shown in Figure 6. RACI2BPMN module receives the BPMN description of a resource-unaware BP model as an XML file, and the representation of the associated RASCI matrix together with the required binding information in a JSON file. Applying the transformations explained in Section 5.2, the tool automatically generates a BPMN model in which the previous tasks are now collapsed sub-processes containing the RASCI-related information, i.e. the required control flow and the proper assignment expressions. As depicted in the figure, the appearance of the resulting model is very similar to the initial process. We have used RAL to write the resource assignment expressions because of its advantages with respect to other options such as XPath (cf. Section 5.1). Nonetheless, a requirement for the design of the prototype has been to allow the easy modification of the language used for the assignment expressions.

The BPMN generated can be manipulated in any BP modelling tool, such as Oryx [11], and it can be executed in BPMSs such as Activiti, in which we have previously included the required functionality to process RAL expressions.

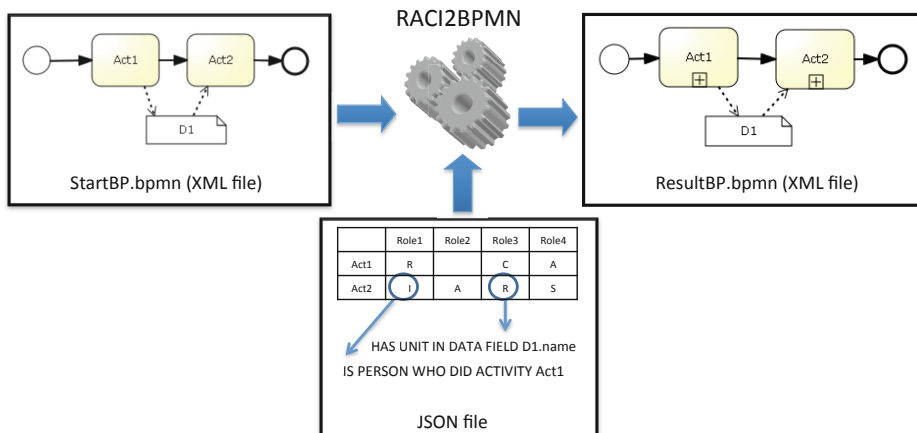


Fig. 6. Overview of the RACI2BPMN prototype

Notice that a pre-processing of the resulting BPMN model could be performed before launching the process in order to make sure that the task labels in the RASCI sub-processes conform to the name convention patterns established.

Further information about the prototype and examples can be found at www.isa.us.es/cristal.

7 Related Work

Human resource management in BPs is an appealing challenge that is recently catching much attention in academy. However, existing approaches are mainly focused on indicating who must perform the BP activities and/or on introducing the so-called access-control constraints (e.g. SoD, BoD) in WFs and BP models [7, 12–15], but they leave aside the rest of responsibility functions involved in RACI matrices.

Last year, we gave a step forward in the merger of RASCI matrices and BPMN, introducing a collection of *RASCI patterns*, and an extension for BPMN 2.0 [3] to allow the modelling of such patterns [5]. Unlike our current approach, that work was purely centered at design time and the assignment of organizational roles to RASCI roles in the BPMN model was done by means of the swimlanes. Now that RAL has a well-defined semantics, we believe it is a good alternative to perform the resource assignments at activity level, as proposed by BPMN 2.0. To the best of our knowledge, there are not other proposals so far pursuing the same goal we are chasing [5]. Indeed, as derived from recent studies, the handling of RACI matrices may become a very tough task [16].

In the market, on the contrary, it is noticeable the increasing interest of companies and software developers in the use of RACI responsibility management with BPs. Academic Signavio⁸ has recently added capabilities to indicate what

⁸ <http://www.signavio.com/en/academic.html>

RACI role each resource assignment associated to a task refers to, and to generate the RACI matrix automatically from the BPMN model. However, the solution is ad-hoc, it is “just” a matter of design (without any semantics), and the generation of the RACI matrix is something trivial having all the information already in the model. The same is allowed in and applies to iGrafx for Enterprise Modeling⁹. Other tools such as YAWL [17], WS-HumanTask [18] and BPEL4People [19] do not provide support for RACI, being their improvements on resource management mainly oriented to the introduction of capabilities for task delegation, re-allocation, team work, and the like (see the definition of the WFPs for further information about resource management in WFs [8]).

8 Conclusions and Future Work

From this work we can conclude that it is possible to join RASCI information with BP models, and so be able to perform *complete* responsibility management in our organization. Furthermore, it is possible to do so by generating a BP model containing all the required information about resources (i.e. resource assignment expressions including the binding information necessary) to be executed by a process engine (i.e. a BPMS) with no need for any changes. In this sense, with our approach we outperform the scope of previous work focused on the use of BP swimlanes, which actually do not contribute in the generation of an executable BP due to their lack of semantics, and the limited flexibility as for expressing resource assignments. Our solution gets to keep the resulting BPMN as “clean” as possible from the viewpoint of the initial process, since all the RASCI information is at sub-process level. Besides, by decoupling responsibility management and BP management, the problem of how to maintain the consistency between the information in RASCI matrices and in BP models is already solved. Any change regarding resources can be automatically applied to the BP model by performing the transformation rules we introduced in this paper. This, thus, avoids synchronization problems.

The approach presented in this paper could be specially useful to those organizations that are not using BPs yet but want to move to a process-oriented style, and use RACI to manage responsibility. Our transformations could help them build RACI-aware BP model directly by re-using the information they have in the RACI matrices.

The analysis of RASCI matrices regarding resource management, as well as the resource assignment expressions resulting from the transformation, is part of our future work in this line, for which we already have some results.

References

1. Smith, M.: Role And Responsibility Charting (RACI). In: Project Management Forum (PMForum), p. 5 (2005)
2. Conchúir, D.O.: Human Resource Management Processes. In: Overview of the PMBOK Guide, pp. 129–145. Springer, Heidelberg (2011)

⁹ <http://www.igrafx.com/products/process4em/>

3. BPMN 2.0, recommendation, OMG (2011)
4. ARIS, RACI. ARIS Community's Website (2012), <http://www.ariscommunity.com/raci>
5. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: Mixing RASCI Matrices and BPMN Together for Responsibility Management. In: VII Jornadas en Ciencia e Ingeniería de Servicios (JCIS 2011), vol. 1, pp. 167–180 (2011)
6. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow Resource Patterns. Tech. rep., BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven (2004)
7. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: RAL: A High-Level User-Oriented Resource Assignment Language for Business Processes. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 50–61. Springer, Heidelberg (2012)
8. 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)
9. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logics Handbook: Theory, Implementations, and Applications. Cambridge University Press (2003)
10. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: Defining and Analysing Resource Assignments in Business Processes with RAL. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 477–486. Springer, Heidelberg (2011)
11. Decker, G., Overdick, H., Weske, M.: Oryx – An Open Modeling Platform for the BPM Community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008)
12. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.* 2, 65–104 (1999)
13. Awad, A., Grosskopf, A., Meyer, A., Weske, M.: Enabling Resource Assignment Constraints in BPMN. Tech. rep., BPT (2009)
14. Wolter, C., Miseldine, P., Meinel, C.: Verification of Business Process Entailment Constraints Using SPIN. In: Massacci, F., Redwine Jr., S.T., Zannone, N. (eds.) ESSoS 2009. LNCS, vol. 5429, pp. 1–15. Springer, Heidelberg (2009)
15. Strembeck, M., Mendling, J.: Modeling process-related RBAC models with extended UML activity models. *Inf. Softw. Technol.* 53, 456–483 (2011)
16. Bronkhorst, J.: RACI matrices - how difficult can it be? HP's Website (June 2010), <http://h30507.www3.hp.com/t5/ITILigent-Service-Management/RACI-matrices-how-difficult-can-it-be/ba-p/41138>
17. Adams, M.: YAWL v2.3-User Manual. Tech. rep., The YAWL Foundation (2012)
18. Web Services-Human Task (WS-HumanTask) v1.1. Tech. rep., OASIS (2010)
19. WS-BPEL Extension for People (BPEL4People). Tech. rep., OASIS (2009)

Semantic Machine Learning for Business Process Content Generation

Avi Wasser¹ and Maya Lincoln²

¹ University of Haifa, Israel

`awasser@haifa.ac.il`

² ProcessGene Ltd.

`maya.lincoln@processgene.com`

Abstract. Business process modeling is considered a manual, labor intensive task. It requires significant domain expertise and may be prone to errors or inconsistencies due to reliance on human factors. Hence, automation through reuse of predefined process models is becoming a common practice for generating new models. In this work we extend a previously proposed generation method by adding semantic learning capabilities that opt to improve the quality of generated business process models. The learning mechanism analyzes, in real-time, the linguistic relationships between process descriptors and adjusts them according to human inputs that are accumulated during the modeling process. To demonstrate the method we present a case-study from the food manufacturing industry. To estimate the applicative value we further experimented the method on a real-life process repository, showing that the learning mechanism increases the effectiveness of the previously suggested method for automating the design of new business process models.

Keywords: Business process model design, Business process repositories, Business process semantic similarity, Machine learning.

1 Introduction

Business process modeling is considered a manual, labor intensive task. It requires significant domain expertise and is prone to errors or inconsistencies due to reliance on human, non-machine assisted activities. Hence, automating the reuse of predefined process models is becoming a common practice for creating new business process models. Research in this field has focused on structured reuse of existing building blocks and pre-defined patterns that provide context and sequences [5]. The work in [11] established a method for designing new business process models from process repositories, based on semantic similarity. This method guides business analysts that are non domain experts, by suggesting process steps that are relevant for the realization of the process goal. The business logic for such suggestions is extracted from process repositories through the analysis of existing business process model activities. Each activity is encoded automatically as a *semantic descriptor* using the Process Descriptor Catalog (“PDC”) notation, suggested first in [12] and elaborated in [11].

This work aims to take the framework presented in [11] several steps forward by: (1) proposing a machine learning mechanism that will take into account the designer preferences at each design phase and adjust (in real-time) the suggestions made by the automated design mechanism at the next design phases; and (2) applying the suggested framework on real-life processes. Our work presents the following innovations: (a) it provides a generic, real-time, machine learning mechanism for the design of new business process models; (b) it equally utilizes objects and actions for machine learning: we make use of all activity linguistic components (object, actions and their qualifiers) concurrently, without special focus on objects (as object centric methods do) or on actions (as activity-centric methods do); and (c) it significantly extends the descriptor space model [11] to enable the ongoing update of its underlying business logic as a result of learning, turning it into a *learning* descriptor space.

The proposed extended method can assist process analysts in designing new business process models while making use of knowledge that is encoded both in the design of existing, related process models, and also from the accumulated knowledge of the human designer. The extended framework is illustrated throughout the paper using an example based on real-life processes from the food manufacturing industry. We also use this process-repository to demonstrate a case study, and use it as a basis for experiments that measure the effectiveness of the proposed machine learning framework.

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 semantic descriptor notion [11] as background to this work. In Section 4 we elaborate the descriptor space concept presented in [11] to support the learning framework. We describe the extended automation method for designing new business process models in Section 5. Section 6 introduces the case study and our experiments and empirical analysis. We conclude in Section 7.

2 Related Work

Research on automated process model generation mainly focuses on supporting the design of alternative process steps within existing process models [16,5,6,1]. The identification and choice of relevant process components are widely based on the analysis of linguistic components - actions and objects that describe business activities. Most existing languages for business process modeling and implementation are activity-centric, representing processes as a set of activities connected by control-flow elements indicating the order of activity execution [19,10]. Other works are action-centric, analyzing the connectivity and relationships between actions [17].

In recent years, an alternative approach has been proposed, which is based on objects (or artifacts/entities/documents) as a central component for business process modeling and implementation. This relatively new approach focuses on the central objects along with their life-cycles. Services (or tasks) are used to specify the automated and/or human steps that help move objects through their

life-cycle, and services are associated with artifacts using procedural, graph-based, and/or declarative formalisms [8]. Such object-centric approaches include artifact-centric modeling [14,2], data-driven modeling [13] and proclets [18].

Although most works in the above domain are either object, action or activity centric, only few works combine the three approaches in order to exploit an extended knowledge scope of the business process. The work in [9] presents an algorithm that generates an information-centric process model from an activity-centric model. The works in [12,11] present the concept of business process descriptor that decomposes process names into objects, actions and qualifiers. In our previous review (see [11]) we identified only a few works that addressed the design of *new* models. The work presented in [13], for example, utilizes the information about a product and its structure for modeling large process structures. [15] presents a method for designing new manufacturing related processes based on product specification and required design criteria. The work in [6] supports modeling recommendations based on the interpretation of process descriptions.

Focusing on our previous work in [11], we realized that although it presented a method for new process model design based on business process descriptor analysis, it didn't apply a machine learning mechanism to provide a real-time improvement of the suggested framework based on human inputs. We did find some works that involve learning as means for achieving other business processes repository utilization targets. For example, some works suggest frameworks for better understanding business process models that apply learning during simulations [3] or runtime [4], and the work in [7] uses reinforcement learning to solve a resource allocation optimization problem.

In this work we elaborate research in this domain by: (a) proposing a learning mechanism that will take into account the designer preferences at each design phase and adjust (in real-time) the suggestions made by the design assistant framework at next design phases; and (b) elaborating the descriptor space concept to support learning frameworks.

3 The Semantic Descriptor Model

In the Process Descriptor Catalog model ("PDC") [12] each activity is composed of one action, one object that the action acts upon, and possibly one or more action and object qualifiers, as illustrated in Fig. 1, using UML relationship symbols. 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,"¹ can be used to automatically decompose process and activity names into *process/activity descriptors*.

For example, the activity "Manually mix wheat flour" generates an activity descriptor containing the action "mix," the action qualifier "manually," the object "flour" and the object qualifier "wheat."

¹ <http://nlp.stanford.edu:8080/parser/index.jsp>

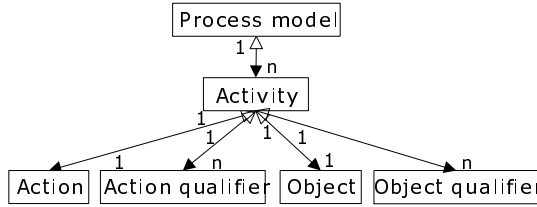


Fig. 1. The activity decomposition model

In general, given an object, o , an object qualifier, q_o , an action, a , and an action qualifier, q_a , a descriptor, d , is denoted as follows: $d = (o, q_o, a, q_a)$. A *complete action* is an action with its qualifier, and similarly, a *complete object* is an object with its qualifier. We denote by $a(d)$ the complete action part of the descriptor, e.g., “manually mix” in the example, and similarly, $o(d)$ denotes the complete object part. In addition, $q_o(d)$ denotes the object qualifier part, $o_{thin}(d)$ denotes the object part of the descriptor (without its qualifiers), and $a_{thin}(d)$ denotes the action part.

3.1 A Descriptor Model for Process Design

The PDC model of [12] was enhanced by [11] to support automated process design. The extended model has two basic elements, namely objects and actions, and four taxonomies are delineated from them, namely an *Action Hierarchy Model (AHM)*, an *Object Hierarchy Model (OHM)*, an *Action Sequence Model (ASM)* and an *Object Lifecycle Model (OLM)*. The business action and object taxonomy models organize a set of activity descriptors according to the relationships among business actions and objects both hierarchically and in terms of execution order, as detailed next.

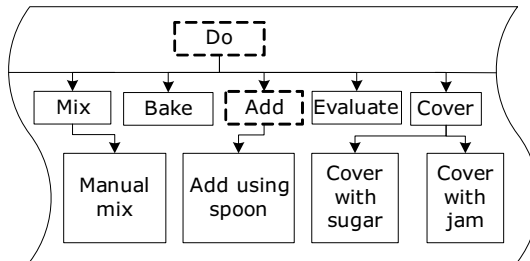


Fig. 2. Segment of an action hierarchy model

The hierarchical dimension of actions and objects is determined by their qualifiers. To illustrate the hierarchical dimension, a segment of the action hierarchy model of a bakery is presented in Fig. 2 and a segment of the object hierarchy

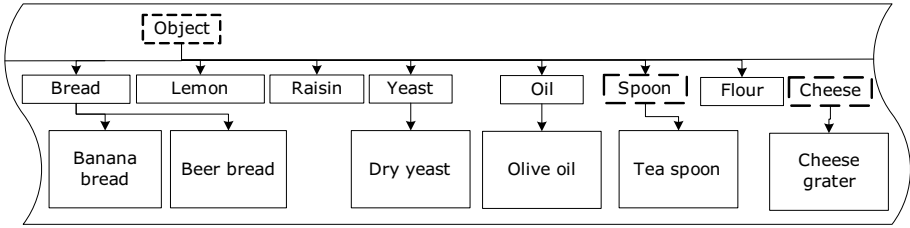


Fig. 3. Segment of an object hierarchy model

model of a bakery is presented in Fig. 3. In the action hierarchy model, for example, the action “Mix” is a subclass (a more specific form) of “Manual mix,” since the qualifier “Manual” limits the action of “Mix” to reduced action range.

It is worth noting that some higher-hierarchy objects and actions are generated automatically by removing qualifiers from lower-hierarchy objects and actions. For example, the action “Add” was not represented without qualifiers in the bakery process repository, and was completed from the more detailed action: “Add using spoon” by removing its action qualifier (“using spoon”) (see Fig. 2). This type of objects and actions, namely: *artificial* objects and actions, are marked with a dashed border. In addition, a root node “Do” is added to any action hierarchy model and a root node “Object” is added to any object hierarchy model.

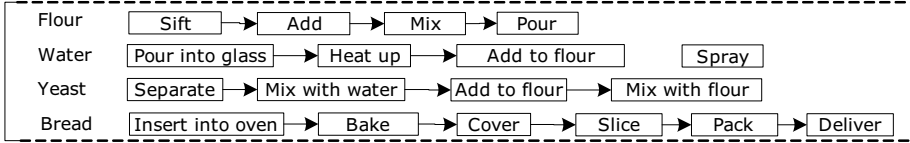


Fig. 4. Segment of an action sequence model of a bakery

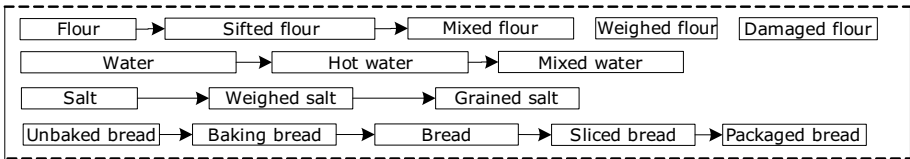


Fig. 5. Segment of an object lifecycle model of a bakery

In the action sequence model, each object holds a graph of ordered actions that are applied to that object (see illustration in Fig. 4). For example, the object “Flour” is related to the following action sequence: “Sift” followed by “Add,” “Mix,” and finally “Pour.”

In the object lifecycle model each object holds a graph of ordered objects that expresses the object’s lifecycle, meaning - the possible ordering of the object’s

states. In other words, an *OLM* is a graph of ordered complete objects that expresses the possible ordering of the object's states (see illustration in Fig. 5). For example, the object "Flour" is part of the following object lifecycle: "Flour" \rightarrow "Sifted flour" \rightarrow "Mixed flour."

Note that *ASM* and *OLM* are defined as sets of sequences and not as a single sequence, since different unconnected processes in the repository may involve the same object, and therefore contribute a different sequence to these models. We denote the procedure for creating an *ASM* and an *OLM* for a complete object, o , as: $create_{ASM}(o)$ and $create_{OLM}(o)$, respectively.

4 The Learning Descriptor Space

In this section we extend the Descriptor Space (DS) concept presented in [11] to express learned information regarding a designer's preferences and knowledge.

Based on [11], it is possible to visualize the operational range of a business process model as a descriptor space comprised of related objects and actions. The descriptor space describes a range of activities that can be carried out within a process execution flow. The coordinates represent the object dimension, the action dimension, and their qualifiers. Therefore, each space coordinate represents an activity as a quadruple $AC = \langle o, q_o, a, q_a \rangle$.

Once constructed, the descriptor space includes all the possible combinations of descriptor components, forming a large and diversified set of possible descriptors. It includes several "virtual" combinations - that did not originally exist in the original process repository. These virtual combinations, together with existing activities, form a significantly extended repository that is used for the automated design of new business processes as well as for the learning mechanism. For every two coordinates in the descriptor space a *distance function* is defined as a linear combination of changes within each of its dimensions. Therefore, four specific distance measures are defined as follows.

Definition 1. Object distance (OD): Let o_i and o_j be two objects, OD_{ij} is the minimal number of steps connecting o_i and o_j in the object lifecycle model.

In a similar way *Action distance, AD*, is defined, calculated based on the action sequence model. For example, the action distance between "Sift" and "Mix" when acted on "Flour" is 2 (see Fig. 5).

Definition 2. Object hierarchy distance (OHD): Let o_i and o_j be two objects, OHD_{ij} is the minimal number of steps connecting o_i with o_j in the object hierarchy model.

In a similar way *Action hierarchy Distance, AHD*, is defined, calculated based on the action hierarchy model.

Definition 3. Object learned proximity (OLP): Let o_i and o_j be two objects in the object lifecycle model, OLP_{ij} is a constant positive number representing the learned proximity between o_i and o_j in the object lifecycle model. *OLP* is calibrated to 0 at the beginning of the first design step, and is updated according to

the learning mechanism presented in Section 5. Higher values of *OLP* represent learned proximities.

In a similar way: (1) *Action learned proximity, ALP*, is defined, calculated based on the action sequence model; (2) *Object hierarchy learned proximity, OHLP*, is defined, calculated based on the object hierarchy model; and (3) *Action hierarchy learned proximity, AHLPL*, is defined, calculated based on the action hierarchy model.

OD, AD, OHD and *AHD* are combined with *OLP, ALP, OHLP* and *AHLPL* to generate a specific distance function between any two activities AC_i and AC_j , as follows (brackets are for readability only):

$$Dist(AC_i, AC_j) = (OD_{ij} - OLP_{ij}) + (AD_{ij} - ALP_{ij}) + (OHD_{ij} - OHLP_{ij}) + (AHD_{ij} - AHLPL_{ij}) \quad (1)$$

It is worth noting that the hierarchy distances (*OHD* and *AHD*) can always be calculated since the hierarchy models that they rely on are bidirectional trees. However, the distances *OD* and *AD* can be undefined in some cases (e.g., when the two objects are not connected in the object hierarchy model, or when the two actions are not acted upon the same object and therefore do not take part in the same action sequence). In these cases the above distance components contribute a *no-connection* distance to the overall distance function. This distance is an application-specific tunable parameter.

In general, it is possible to navigate within the descriptor space (hence, move from one descriptor to another) in a meaningful way. This navigation enables us to move up to more general or drill down to more specific action and object scopes as well as to navigate to: (a) preceding and succeeding actions that act on the descriptor's object and (b) advance to a successor (more advanced) state of the object's current state or recede to a predecessor (less advanced) state.

5 Method for Automated Generation of Business Process Content

In this section we extend the method presented in [11] by adding a learning mechanism.

The design assistance method relies on an underlying process descriptor space and at any phase, based on the user's decision, it either refines an existing process activity or suggests a next process activity. Based on each such user decision, the design assistant learns more about the relationships between the involved actions and objects and adjusts their distances in the descriptor space accordingly.

The design assistant is illustrated in Fig. 6. The design process starts when a process designer defines the name of the new process model. This name is decomposed into a process descriptor format. For example, a new process named: "Bake raisin bread," will be transformed into the following process descriptor: object="bread," action="bake," object qualifier="raisin," action qualifier="null."

Based on the process descriptor input, the design assistant produces options for the first process activity (see Section 5.1). The process designer reviews the

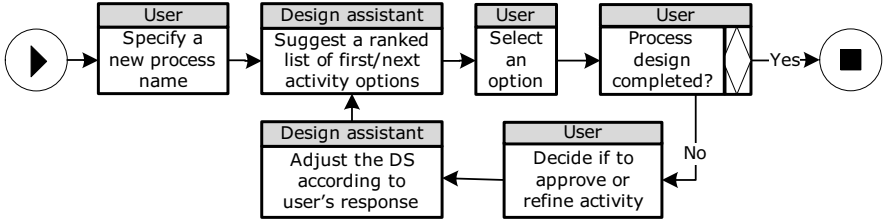


Fig. 6. The design assistant mechanism

output option list, and either selects the most suitable first activity for the newly designed process, or suggests an alternative. At any next phase the designer either requests to refine the current activity (see Section 5.2) or advance to design the next activity (see Section 5.3). Each time the design assistant is requested to suggest activities as part of the design process it outputs a list of options, sorted and flagged according to the option's relevance to the current design phase and based on the current descriptor space (see Section 5.4). Based on the designer's preferences regarding the most suitable activity from the option list and whether to refine or proceed to the next activity, the design assistant deduces new knowledge regarding relationships between actions and objects in the descriptor space and adjusts its distances accordingly (see Section 5.5).

After selecting the most suitable process activity from the suggested list, the designer examines the newly designed process model to determine if it achieves the process goals. If goals are achieved, the design is terminated; else - the design procedure continues until the process goal is achieved.

5.1 Suggesting the First Process Activity

To suggest the first process activity, the design assistant searches the target object and its more specific objects within the object hierarchy model. It then creates first activity suggestions in the format of activity descriptors comprised of the retrieved objects and the first action that acts upon them in the action sequence model. Continuing the example above, the following first activity options will be suggested (see Fig. 3 and Fig. 4): "Insert bread into oven" and "Insert banana bread into oven."

5.2 Refining the Currently Suggested Process Activity

A refinement can be performed by five orthogonal methods. To illustrate each of these methods we will show how the action "Cover bread" can be refined.

Action and Object Refinement. To refine the reference action, the design assistant navigates the descriptor space by drilling *down* the action hierarchy to more specific actions. It then combines the retrieved, more specific, actions with the reference object. The refinement of objects is done in a similar manner.

By applying an action refinement to our example’s reference activity, the refinement option: “Cover bread with sugar” is retrieved (see Fig. 2).

Action and Object Generalization. The generalization method is similar to the action and object refinement method, only this time the design assistant navigates the descriptor space by moving *up* the action and the object hierarchical dimension, respectively.

Advance an Action or an Object State. To advance the object’s state within an activity, the design assistant navigates the descriptor space by moving *forward* in the object lifecycle sub-dimension. In a symmetrical manner, to advance an activity’s action, the design assistant moves forward in the action sequence sub-dimension of the descriptor space. In our example the object “Sliced bread” represents a more advanced state of the object “Bread” (see Fig. 5) and the action “Slice” follows the action “Cover” in the action sequence applied on “Bread” (See Fig. 4). Therefore, the following two refinement suggestions are constructed: “Cover sliced bread” and “Slice bread.”

Recede to a Less Processed State of the Object or to a Former Action. The receding method is similar to the advancing method, only this time the design assistant navigates the descriptor space by moving *backwards* in the object lifecycle and action sequence sub-dimensions. For example, the action “Bake” is acted on “Bread” before this object is covered (before the action “Cover” is applied) (see Fig. 4), hence creating the option: “Bake bread.”

Move to a Sibling Action or Object. In order to move to a sibling action, the design assistant moves horizontally within the action hierarchical sub-dimension. By fixing the reference action’s level, it retrieves sibling actions for this action. Moving to a sibling object is conducted in a similar manner. Continuing our example, a navigation to sibling actions to “Cover” retrieves a list of activities that includes: “Mix bread” and “Evaluate bread” (see Fig. 2).

5.3 Suggesting the Next Process Activity

This step can be achieved in two alternative ways: either by advancing to a later action that acts on the currently accepted (reference) object, or advancing to a sibling object combined with the reference activity’s action. To demonstrate this step, consider the activity following “Add yeast to flour.” The design assistant finds in the action sequence model the option: “Mix yeast with flour” (see Fig. 4). In addition, sibling objects to “Yeast” are also retrieved from the object hierarchy model, creating additional options such as “Add raisin” and “Add lemon” (see Fig. 3).

5.4 Preparing a Set of Output Options

The design assistant assesses the output options in each navigation phase and combines an ordered option list to assist the user in selecting the most suitable

option. The design assistant sorts the options according to their relevance to the current design phase based on two considerations. First, on proximity to the design phase reference coordinate - which represents the last selected activity when suggesting a refined or next activity, or to the targeted process descriptor when suggesting the first process activity. Second, the design assistant considers to what extent was it changed comparing to actual activities that were part of the underlying process repository. Therefore, the construction of the ordered option list is conducted according to the following three stages: (a) sort by proximity to the reference activity; (b) internally sort by similarity to processes in the repository; and (c) flag each option, as further detailed below.

Sort by Proximity to the Reference Activity. The design assistant calculates the distance between the reference coordinate and each of the list options (see definition 1), and sorts the list in an ascending order - from the closest to the most distant option.

Internally Sort by Similarity to Processes in the Repository. The design assistant also takes into account the extent to which a proposed activity was changed in comparison to actual activities in the underlying process repository. For this purpose the design assistant distinguishes between three change levels: (a) *No change*- the suggested activity is represented “as is” within the underlying business process repository. These options are not marked by any flag; (b) *Slight modification* - there is an actual activity in the underlying business process repository containing the same object and action with different qualifiers. These options are marked with “~”; (c) *Major change* - the object and action within the suggested activity were not coupled in any of the activities within the underlying business process repository. These options are marked with “M”.

According to the example presented in Section 5.3, several options were generated as candidates for next activities to be conducted after the activity “Add yeast to flour.” Most of these options were produced by combining the action “Add to flour” with siblings of the object “Yeast,” hence having the same distance from the reference activity. Nevertheless, these options can further be differentiated. For example, “Add lemon” is an actual activity in the bakery process repository, and therefore is flagged as such. Nevertheless, “Add oil” has no representation in this repository, but since “Add olive oil” does, this option is flagged by “~.” Since there is no descriptor that combines the action “Add” and the object “Spoon” in this repository, the option “Add spoon” is flagged by “M.”

Flag Each Option. After assessing each option’s relevance to the current navigation phase and sorting the option list accordingly, the design assistant tags each option with both the numerical distance value and the change level. For example, the option “Add oil” from the example above will be flagged “[2,~].”

5.5 Applying a Learning Mechanism

At each design step the process designer is provided with a list of optional next activities, and is required to make the following two decisions: (1) select the most suitable descriptor, d_s , from the option list; and (2) decide whether to refine the selected descriptor or to accept it and proceed to the design of the next activity. The learning mechanism receives the two above designer decisions and deduces new conclusions regarding the underlying business rules and business know-how encoded in the descriptor space. As a result, the learning mechanism adjusts the descriptor space, which is then used as a basis for producing the next/refined optional activity list. The learning mechanism analyzes the following designer decisions as detailed next.

Selecting Artificial Activities. As presented in Section 3, some actions and objects in the descriptor space are artificial (automatically generated in the action and object hierarchy models), and hence are not represented in any of the process repository activities. In case the designer selects a descriptor that contains an artificial action or object, the learning mechanism deduces that these are “real-life” semantic elements and amends the descriptor space as follows.

1. Representing the artificial action or object as a real one in its hierarchy model. For example, in case the designer selects the activity “Use grater,” the object “Grater” becomes a real object in the object hierarchy model, and its dashed line is replaced by a regular one (see Fig. 3).
2. Generating a new action sequence for the selected activity’s object. Following the above example, a new sequence in the action sequence model is automatically generated for the object “Grater,” including one action, “Use.” Formally, this suggestion can be given by:

$$create_{ASM}(o(d_s)) \quad (2)$$

3. Automatically updating the object lifecycle model by adding a representation of the selected activity’s object. Following the above example, the object “Grater” is added to the *OLM* as a single-object lifecycle. Formally, this suggestion can be given by:

$$create_{OLM}(o(d_s)) \quad (3)$$

Selecting Virtual Activities. Some options in the suggested descriptor list are marked as “Slight modification” or “Major change,” hence representing a virtual activity in the descriptor space (e.g. “Add oil,” “Add spoon”). In case the process designer selects such options, the learning mechanism deduces that the complete action can be applied to the complete object in “real-life” processes. the descriptor space is therefore amended as follows.

1. Adding $a(d_s)$ to the action sequence of $o(d_s)$. Following the above example, the action “Add” is added to the action sequence of “Oil” in the action sequence model. In order to update the *ASM*, we regenerate $ASM(o(d_s))$. Formally, this suggestion can be given by Eq. 2.

2. Updating the object lifecycle of $o(d_s)$. Formally, this suggestion can be given by Eq. 3.

Selecting Distant List Options. As mentioned above, each optional activity flag indicates the activity’s numerical distance from the reference descriptor, d_r . Options with the minimal distance, $Dist_{min}$, are presented at the top of that list, followed by other options in an ascending distance order. For example, given the reference activity “Sift flour,” the optional activity list starts with the options “[1] Add flour” followed by “[2] Mix flour” (see Fig. 4). In case the process designer selects an activity flagged by distance $Dist(d_s) > Dist_{min}$, it is possible to learn that the *actual* distance between the selected and reference activities is shorter than the one currently represented in the *DS*. Higher values of $Dist(d_s)$ represent a greater difference between the actual and current *DS*. In our example, the designer may select the activity “[2] Mix flour” although it is not the first option in the list.

In response to such designer selections, the learning mechanism reacts as follows. First, it calculates the difference between the distance components - *OD*, *AD*, *OHD* and *AHD* of $Dist_{min}$ and those of $Dist(d_s)$. We denote this difference as the actual gap, *AG*. In our example the actual gap between *OHD* of the first list option, $d_{first} = (flour, null, add, null)$, and the selected option, $d_s = (flour, null, mix, null)$, is: $AG_{OHD}(d_{first}, d_s) = 0$ (See Fig. 3), and the actual gap between their action distance is: $AG_{AD}(d_{first}, d_s) = AD(d_s) - AD(d_{first}) = 2 - 1 = 1$ (see Fig. 4).

Second, the learning mechanism corrects the learned proximities related to each of the four distance components (*OLP*, *ALP*, *OHLP* and *AHLP* (see Section 4)) by adding them a numerical value proportional to their actual gap. Additions to learned proximities in case of “next activity” decisions are more significant than additions in case of “activity refinement” decisions, since the first indicates an *exact* match while the second indicates *proximity* only. Formally: the object learned proximity between $o(d_r)$ and $o(d_s)$, $OLP_{new}(o(d_r), o(d_s))$, in case of a “next activity” decision, is corrected as follows: $OLP_{new}(o(d_r), o(d_s)) = OLP(o(d_r), o(d_s)) + AG_{OD}(d_{first}, d_s) * h_{next}$, where h_{next} is a tunable parameter that can be optimized in a future work. The calculation in case of an “activity refinement” is similar, using the tunable parameter $h_{refine} < h_{next}$. *ALP*, *OHLP* and *AHLP* are updated similarly. In continuous to our example, assuming the designer chooses to move to the next activity, “Mix flour,” and assuming $ALP(o(d_r), o(d_s)) = 0$ and $h_{next} = 0.1$, we calculate the new action learned proximity as follows: $OLP_{new}(o(d_r), o(d_s)) = 0 + 1 * 0.1 = 0.1$.

Selecting Refined List Options. In some cases the next activity is selected after several refinement steps. After such design phases, that involve n refinements, the learning mechanism shortens the distance between the previous (first reference) selected descriptor, d_r , and the new (currently selected one), d_s , as follows: $Dist_{new}(d_r, d_s) = Dist(d_r, d_s) - n * h_{next}$.

6 Case Study and Experiments

6.1 Case Study: An Example for Designing a New Process Model

To illustrate the proposed framework we present two short examples from the field of bakeries. The bakery process repository covers bakery activities starting from the raw material procurement, through the manufacturing of baked goods and terminating as the baked goods are sold to the customer. The newly designed processes are related to the bakery field, but are not covered by the process repository. The first new process, “Bake a chocolate cake,” extends the process repository by baking a new product. The second new process, “Sell baked goods via the Internet” extends the process repository by offering an additional service to customers, that eliminates the need for their arrival to the store. Using these examples we will show how the learning mechanism can be utilized to guide and improve the design of new processes. In both examples h_{next} was set to 0.6 and h_{refine} was set to 0.5.

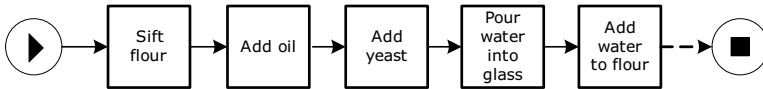


Fig. 7. The new designed process diagram for “Bake a chocolate cake”

The first example supports the design of a new business process for: “Bake a chocolate cake.” The generated output (new process model) of this example is illustrated in Fig. 7 as a YAWL (Yet Another Workflow Language) diagram. The design process starts when the (human) process designer inserts the following process descriptor: (action=“bake”, action qualifier=null, object=“cake”, object qualifier=“chocolate”) to the process assistant and determines that the first activity is: “Sift flour.” Respectively, the process assistant searches the descriptor space, looking for next activity possibilities. The result set includes the following activities (see Sections 5.3 and 5.4): “[1] Add flour,” “[2] Sift oil” and “[2,M] Sift yeast.” The designer selects the option “Add flour” and decides to refine it. After one refinement the activities “Add oil” and “Add yeast” are selected, and in the next design phase after four refinements the activity “Pour water into glass” is selected. As a result, the process designer suggests an option list for the next activity that starts with the option: “[1] Heat up water.” This first option is selected for refinement and as a result an option list is suggested, containing the option: “[1] Add water to flour.” This option is selected as a valid option in the process model. Therefore, its distance from the original reference activity, “Pour water into glass” was shortened from 2 into $2 - 2 * 0.6 = 0.8$. In contrast to the original descriptor space (Fig. 4), the activity “Add water to flour” is now more closer to “Pour water into glass” than the activity “Heat up water.” This automatically acquired knowledge seems reasonable, since cake preparation processes do not usually require warm water as in the case of bread preparation processes. After

12 additional design phases, the process design reaches the phase of preparing a jam cover for the cake. In this case the activity “Pour water into glass” was selected again and this time the activity “Add water to flour” was suggested *before* (and instead of) the activity “Heat up water,” saving one refinement step.

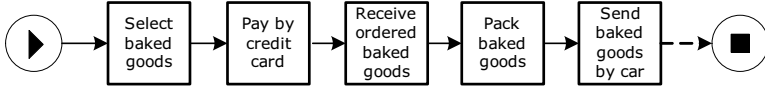


Fig. 8. The new designed process diagram for “Sell baked goods via the Internet”

The designer is now interested to design the new business process: “Sell baked goods via the Internet.” The design process is conducted in a similar manner to the one presented above and results in the process diagram presented in Fig. 8. An interesting observation in this design process is the learning usage of the activity “Send baked goods by car.” While the original business process repository contained the action “Send by car” applied only to the object “damaged flour,” the terminating activity combines this action with the object: “baked goods.” This was achieved by the following design phases: after accepting the activity: “Pack baked goods,” the designer asks for next activity suggestions and receives an option list. Knowing that in order to fulfill the process goal, the last activity should involve a sending by car action, she selects the option: “Send damaged goods by car” and asks to refine it since it does not provide the exact required activity. Since the objects “Damaged goods” and “Baked goods” are siblings in the object hierarchy model, one of the refinement options is: “[2,M] Send baked goods by car,” although it represents a major change to the underlying process repository. The designer approves the new process design as a complete design that fulfills the process goal, and the learning mechanism deduces that the action “Send by car” can be applied on “baked goods” in real-life scenarios and updates the descriptor space accordingly. This new learned knowledge assists in improving the design process of other new process models such as: “Sell baked goods via phone” and “Donate baked goods to poor families.”

6.2 Experiments

We now present an empirical evaluation of the proposed method effectiveness. In order to evaluate the learning mechanism’s contribution, we implement the evaluation method used in [11]. We first present our experimental setup and describe the data that was used. Based on this setup we present the implemented methodology. Finally, we present the experiment results and provide an empirical analysis of these results.

Experiment Setup. The “no-connection” distance (defined in Section 4) was set to 500; h_{next} was set to 0.2 and h_{refine} was set to 0.1.

Data. We chose a set of 12 real-life processes from the bakery process repository, comprising: (a) five processes from the core “Baking” category, with 45 activities altogether; (b) three processes from the “Sales” category, with 22 activities altogether; and (c) four processes from the “Maintenance” category, with 39 activities altogether. The “Baking” data set contains core and industry-specific activities, while the “Maintenance” data set represents a combination of industry-specific as well as more industry agnostic activities. The most generic activity collection is represented in the “Sales” domain, which shares many of its activities with the food manufacturing industry. Using the selected 12 processes we created a “process repository database.”

Evaluation Methodology. To evaluate the suggested method we conducted 12 experiments, each repeated twice: first without applying the learning mechanism (a reference experiment) and second by applying the learning mechanism (a full method experiment). Each experiment was conducted according to the following steps: (a) preparation: remove one of the processes from the database so that the database will not contain any of its descriptor components; (b) run the design assistant mechanism in a stepwise manner. At each phase we try to identify an activity (“goal activity”) that is compatible with the removed process, according to the following steps: (1) if the goal activity’s linguistic components are represented in the Process Repository Database, run the “find next activity” algorithm (see Section 5.3). If the output list contains the goal activity - continue to reconstruct the next goal activity. Else, run the “activity refinement” algorithm (see Section 5.2). If the option list produced by the refinement step does not include the goal activity, choose the activity that shares the largest amount of common descriptor components with the goal activity as a basis for an additional refinement. If, after 10 successive refinements, the required activity is still not represented by one of the output options, it is inserted manually as the next process activity and the design process is continued by locating the next activity; (2) else (the goal activity’s linguistic components are not represented in the Process Repository Database), the next goal activity is inputted manually by the experimenter. In full method experiments, at the end of each such phase the learning mechanism is applied, correcting the current descriptor space as an input to the next design phase (see Section 5.5).

Results and Analysis. Table 1 presents a summary of the experiment results. Each experiment of creating a new process model was based on a database with the set of all activity descriptors in all process models, excluding the set of activity descriptors of one goal process. This means that we aim at recreating the goal activities from a partial set of activity descriptors. On average, for 83.3% of the goal activities, all descriptor components were contained both in the goal process and in another process (see column #3). This was the case despite the relatively small experiment size (11 processes, whereas the entire bakery process repository includes around 70 processes), highlighting the amount of similarity one would expect when designing new processes based on an existing repository.

For the remaining 16.7%, at least one descriptor component was missing. In such a case, the activity was inserted manually during the design process. It is worth noting that for the 83.3% of activities that had the potential of reconstruction from the database, 100% were reconstructed successfully using our method (see Table 2).

Table 1. Experiment results

Column #	1	2	3	4	5	6	7
Column name	# of total processes in DB	# of total activities in DB	% of goal activities represented in the DB	% improvement in avg. # of steps per design phase	% improvement in avg. location of correct option in 'next activity'	% improvement in avg. location of correct option in 'refine activity'	% improvement in avg. location of the correct option per design phase
Avg.-all	12	106	83.3%	12.7%	32.9%	26.4%	28.5%
Avg.-Baking	5	45	85.2%	17.8%	38.6%	30.1%	31.0%
Avg.-Sales	3	22	80.8%	11.6%	34.2%	25.4%	27.7%
Avg.-Maintenance	4	39	82.7%	7.0%	24.8%	22.5%	25.9%

In addition, Table 1 shows that by applying the learning mechanism the following measures were improved. First, on average, the number of iterations required for reconstructing a goal activity (see column #4) was improved by 12.7%. The design of Sales processes required less steps than the design of Baking and Maintenance processes, and therefore was less improved (7% vs. 17.8 and 11.6% on average, respectively). It should be noted that the location of the goal activity was improved significantly in the ranked list of suggested activities (average improvement: 28.5%, see column #7). This location was even better improved at phases that did not involve refinement (average improvement: 32.9%, see column #5); and was a little lower in steps in which a refinement was required (26.4% on average, see column #6). This may be due to the fact that refinement steps include a much larger amount of alternatives, and the tunable parameter h_{refine} is lower than h_{next} (hence, learning is slower in refinement steps). Again it should be noted that results within the Baking category were better than results within the Sales and Maintenance categories - probably due to the larger amount of activities representing each of the Baking processes, which enables more learning opportunities. Another reason may be the similarity between Baking processes which enables applying the learning results from one process to others as well.

Table 2 analyzes the difference in the number of refinements that are needed to design the correct goal activity due to the usage of the learning mechanism.

Table 2. Distribution of successful predictions vs. the number of required refinements

# of refinements	0	1	2	3	4	5	6	7
% of difference in the # of successful predictions	11%	17%	10%	1%	-10%	-9%	-7%	-9%

For each number of refinements (0-7), we record the percentage of cases where this number of refinements was needed: (a) when the learning mechanism is not applied; and (b) when the learning mechanism is applied. Then, we calculate the difference between the results in both cases. We observe, for example, that by applying the learning mechanism, the ability of the system to reconstruct the goal activity after one refinement was improved by 17%. In total, it can be observed that the learning mechanism reduced the number of refinements required to reach the correct goal activity. These results clearly demonstrate that the learning mechanism improves the speed and efficiency of the design method. As hypothesized earlier - a larger database would probably yield even better results.

To summarize, we have shown the usefulness of using the learning mechanism in identifying activities for a new business process. We also showed the mechanism to be effective in the given experimental setup, both in terms of improvement in the number of design steps and in the number of refinements that are needed.

7 Conclusions

We proposed a learning mechanism to improve a machine-assisted design method. Such a mechanism saves design time and supports designers in creating new business process models. The proposed method provides a starting point that can already be applied in real-life scenarios, yet several research issues remain open, including: (1) an extended empirical study to further examine the quality of newly generated processes; and (2) an extended activity decomposition model to include an elaborated set of business data and logic (*e.g.*, roles and resources).

As a future work we intend to investigate further language semantics by using more advanced natural language processing techniques, as well as semantic distances between words. Finally, we intend to apply the learning technique we have developed to create new methods for workflow validation.

References

1. Becker, J., Delfmann, P., Herwig, S., Lis, L., Stein, A.: Towards Increased Comparability of Conceptual Models-Enforcing Naming Conventions through Domain Thesauri and Linguistic Grammars. In: ECIS (June 2009)

2. 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)
3. Cronan, T.P., Douglas, D.E., Alnuaimi, O., Schmidt, P.J.: Decision making in an integrated business process context: Learning using an erp simulation game. *Decision Sciences Journal of Innovative Education* 9(2), 227–234 (2011)
4. Ghattas, J., Soffer, P., Peleg, M.: A Formal Model for Process Context Learning. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 140–157. Springer, Heidelberg (2010)
5. Gschwind, T., Koehler, J., Wong, J.: Applying Patterns during Business Process Modeling. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 4–19. Springer, Heidelberg (2008)
6. Hornung, T., Koschmider, A., Lausen, G.: Recommendation Based Process Modeling Support: Method and User Experience. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 265–278. Springer, Heidelberg (2008)
7. Huang, Z., van der Aalst, W.M.P., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering* 70(1), 127–145 (2011)
8. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
9. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: Bellahsene, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
10. Leopold, H., Smirnov, S., Mendling, J.: On the refactoring of activity labels in business process models. *Information Systems* (2012)
11. 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)
12. 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)
13. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
14. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Systems Journal* 42(3), 428–445 (2003)
15. Reijers, H.A., Limam, S., Van Der Aalst, W.M.P.: Product-based workflow design. *Journal of Management Information Systems* 20(1), 229–262 (2003)
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. Smirnov, S., Weidlich, M., Mendling, J., Weske, M.: Action patterns in business process model repositories. *Computers in Industry* (2012)
18. Van der Aalst, W.M.P., Barthelmeß, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems* 10(4), 443–482 (2001)
19. Wahler, K., Küster, J.M.: Predicting Coupling of Object-Centric Business Process Implementations. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 148–163. Springer, Heidelberg (2008)

Hierarchical Process Verification in a Semi-trusted Environment

Ganna Monakova

SAP Research Karlsruhe
Vincenz-Priessnitz-Str. 1
76131 Karlsruhe, Germany
ganna.monakova@sap.com

Abstract. In a business collaboration different parties work together to create a compositional process that incorporates internal processes of the participants to achieve a common goal. It must be assured that certain requirements, such as legal regulations, are fulfilled by the overall composition. Verification of such requirements requires knowledge about the internal functionalities of the involved parties, who in turn do not want to reveal their processes. This work presents a technique for hierarchical verification of the requirements over the process composition based on the guarantees, called *property assertions*, provided by the collaboration participants.

1 Introduction

Business collaborations require participants to work together to create a composite process that incorporates internal processes of the involved parties. It has to be assured that the resulting process composition satisfies certain requirements, such as applicable legal regulations or service agreements between the parties. If participants fully trust each other, then they can reveal their internal processes. In this case verification of a requirement over composition (called compositional requirement) is no different from verification of a requirement over a local business process, which has been widely discussed in the literature, see Section 5. If, however, the operating environment is not fully-trusted, then participants will not reveal their internal processes as they might contain sensitive information, such as business decisions captured in the process structure. In this case the problem of *How to verify a requirement over process composition without knowledge of the processes* arises.

In this paper we propose the following solution to this problem: instead of revealing the processes, participants provide *property assertions* that reflect certain characteristics of their processes¹. A property assertion specifies *what* is guaranteed by a process, as opposite to the process structure that reveals *how* a

¹ A semi-trusted environment is required, as some of the (abstracted) information must be revealed to enable verification of the process composition. A non-trusted environment would prohibit revelation of any information.

guarantee is achieved. Provided property assertions only need to contain information required by the compositional requirement. Therefore property assertions allow to hide implementation details but still provide sufficient information to enable verification of compositional properties.

The derivation of an appropriate process abstraction that only reveals process properties necessary for a requirement verification is the subject of a different work. In this paper we show how the compositional requirement can be proven based on the provided property assertions. The presented technique can be used for the bottom-up verification, as well as for the top-down process design:

- In a bottom-up approach the existing processes abstract their implementation with the property assertions. This on one hand allows for verification of the requirements in a semi-trusted environment, on the other hand reduces the complexity of the verification problem due to the abstraction of the unnecessary implementation details.
- In a top-down process design the top-level requirement is broken down into sub-requirements that are assigned to the different parties as requirements on the future process implementation. In this case the requirements become the property assertions and can be used as a contract between the participants.

In both cases it is necessary to be able to prove compositional requirements based on the given property assertions and on the structure of the composition, which is the scope of this paper.

1.1 Running Example

As a running example consider a supply chain process shown in Fig. 1. The example collaboration includes three participants: *Retailer*, *Producer* and *Reseller*. The internal processes of the participants are not visible. Instead, participants specify a number of property assertions that are fulfilled by the internal process implementations. We use a semi-formal notation for the property assertions at this point, later in Section 3 we define the formal property assertion language. The flow of the process is the following: *Retailer* computes product demand and, depending on the number of required items, sends an order request either to *Reseller* or to *Producer*. *Reseller* does not reveal their process structure, but guarantees that if the number of ordered items is less than 5000, then the products can be delivered via an express delivery. *Reseller* also specifies that the express delivery is *preceded by* quality check. *Producer* specifies that an order will be either rejected or the goods will be produced and delivered. *Producer* also guarantees that any order above 10.000 will be accepted. They furthermore guarantee that the production of goods is *followed by* a quality check.

As an example requirement over the collaboration we consider the *Product quality must be checked before it can be sold* requirement. In this paper we show how this requirement can be formally proven based on the provided property assertions.

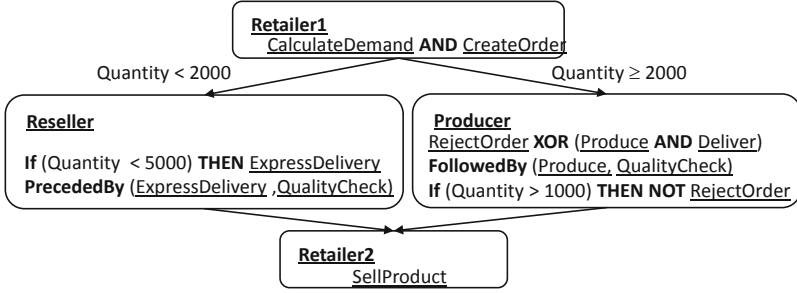


Fig. 1. Example Service Composition

1.2 Approach Overview and Paper Structure

To reflect hierarchical structure of a collaboration, where internal processes are composed to the overall collaboration, we use nested graphs as the formal language for collaborative models. A node in a nested graph can be a graph itself, in which case it is called a hypernode. A nested graph naturally captures semantics of scopes and reflects local and global process visibility through the notion of hypernodes. Hiding the structure of a hypernode on a higher level corresponds to abstraction of an internal process on a higher level.

To be able to prove requirements over the nested process graphs, we introduce a set of generic axioms in Section 2 that reflect execution semantics of nested process graphs. The presented axioms are used to create a set of assertions that model execution semantic of a particular process model. We call such assertions a *verification basis*. A hypernode in a nested graph provides some guarantees about the internal processes in form of process assertions. Section 3 introduces property assertion language. Section 4 shows how the provided property assertions are used to generate models of internal processes. Generated models are added to the verification basis to complete the process specification. Finally, we show how the compositional requirement is mapped to an assertion over the process activities. The mapping is based on the activity types that are used in the requirement specification. The satisfiability of the modelled assertions is checked using the Satisfiability Modulo Theories (SMT) solver Z3 [13]. An SMT solver solves satisfiability problems for Boolean formulas containing predicates of underlying theories. Such theories can be, for example, theories of arrays, lists and strings [2]. In addition, an SMT solver can be extended with new theories as shown in [16]. In the proposed approach we use the theory of the linear arithmetic.

Summarizing, the contribution of this paper are three-fold: first, we present a formal theory of hierarchical process models based on the nested graphs and show how to model process execution semantics with logical assertions. Second, we present an assertion language for specifying process properties without revealing the structure and business logic of internal processes. Third, we present a hierarchical verification technique for verifying global properties of distributed business processes based on the hierarchical process theory.

2 Nested Process Graph Theory

A process graph is a directed acyclic nested graph $G = (A, L)$, where A is the set of (hyper)nodes representing process activities and $L = \{L_k = (A_i, A_j) | L_k.\text{source} = A_i \wedge L_k.\text{target} = A_j\}$ is the set of directed edges representing synchronisation links between activities, where $L_k.\text{source}$ denotes the source and $L_k.\text{target}$ the target activity of link L_k . Activities can be simple or structured activities, in the second case the activity is represented through a directed acyclic subgraph that is viewed as a hypernode from the parent graph. A hypernode is used to represent a subprocess or a process scope.

Each link L_i in a process graph has a transition condition $L_i.\text{condition}$, with default condition being set to *true*. After completion of an activity, states of all outgoing links are evaluated as follows: if activity has been executed and transition condition of an outgoing link L_i evaluates to *true*, then the link status $Status(L_i)$ evaluates to *true*; if activity has been skipped or if the transition condition evaluates to *false*, then link status $Status(L_i)$ evaluates to *false*.

Let $L^{in}(A_i) = \{L_i | L_i.\text{target} = A_i\}$ denote the set of all incoming links for activity A_i . Each activity in the process graph has a join condition $JC(A) = F(Status(L_1^{in}), \dots, Status(L_k^{in}))$, which is a logical expression over the states of the incoming links. The join condition is evaluated after the states of all incoming links have been evaluated. If the activity join condition evaluates to *true*, then activity starts its execution, otherwise it is skipped. The default join condition of each activity is the *OR* function over the incoming link states.

2.1 Activity Modelling and State Axioms

Each activity in a process has an execution lifecycle represented through the following activity states: *ready*, *started*, *skipped*, *executed*, *faulted*, *terminated*, and *compensated*. Figure 2 shows possible transitions between these states. Activity reaches state *ready* when control flow reaches this activity, which happens when all predecessor activities complete their execution and the statuses of all incoming links are evaluated. When an activity is reached, its join condition is evaluated. If it evaluates to *true*, then activity is *started*, otherwise it is *skipped*. If an activity has been *started*, then it either successfully finishes its execution and reaches state *executed*, faults and goes into the state *faulted*, or it terminates and goes into the state *terminated*. When an activity reaches states *skipped* or *executed* it is automatically reaches state *completed*. This triggers evaluation of the outgoing link statuses, which means that successor activities no longer have to wait for this activity to complete. The state diagram contains two additional transitions which are depicted with dashed lines: transition from *executed* to *compensated* state happens if a compensation actions are taken at some point after execution of the activity, transition from *faulted* to *completed* happens if a fault handler repairs activity fault and allows for further execution of the process. These two state transitions are only possible if the corresponding constructs (fault handler and compensation handler) have been defined in the process model.

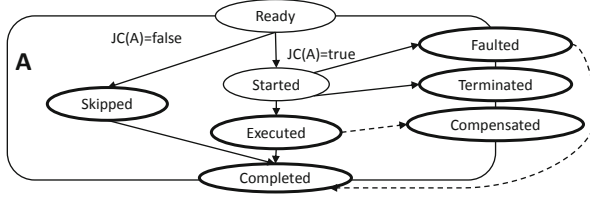


Fig. 2. Activity state transitions

In addition to the state variables, each activity is characterised by the activity type it belongs to. We use A^T notation to refer to an activity type, in contrast to A that is used to denote an activity. Activity types can be arranged into a type-subtype hierarchy. For example, a more fine-granular activity type *AirTransport* is a subtype of a more coarse-granular activity type *Transport*. Each activity type in our approach is represented through an integer interval $ActivityType := Record(start : int, end : int)$. The subtype relations are then represented through the interval inclusion:

$$A_i^T \subset A_j^T \Leftrightarrow (A_i^T.start \geq A_j^T.start) \wedge (A_i^T.end \leq A_j^T.end)$$

For example, an activity type *Transport* is the supertype of activity types *AirTransport* and *GroundTransport*. A possible interval assignment to these types would be $[0, 10]$ for *Transport* and $[0, 3]$ for *AirTransport* and $[4, 6]$ for *GroundTransport*.

To reflect all activity properties, we define activity type as a record consisting of the following fields:

$$Activity := Record(ready : int, \dots, completed : int, atype : int)$$

The value of *atype* field decides to which type this activity belongs: $A \in A^T \Leftrightarrow A.atype \geq A^T.start \wedge A.atype \leq A^T.end$. For example, if a process specifies that it executes an activity of type *Transport*, then the *atype* field of variable A_T that represents this activity will be restricted to $(A_T.atype \geq 0) \wedge (A_T.atype \leq 10)$. Similar, if we assign 2 to the *atype* field, then we specify that this activity is of type *AirTransport* and *Transport*. We will use activity types to define properties of the subprocesses in Section 3.

To represent activity state model and capture semantic of the state transitions, we use eight integer fields: $A.ready$, $A.started$, $A.skipped$, $A.executed$, $A.completed$, $A.faulted$, $A.terminated$ and $A.compensated$. The value of the state field represents the point of time, starting from 0 denoting the process start, when activity reaches this state. If activity does not reach a certain state, the value of the corresponding state field is -1 . Axioms from Table 1 capture state transition semantics of each activity A by defining dependencies between the field values:

- Axiom (N_1) defines that each activity will eventually be started or skipped if it has been reached. We assume that skipping happens in the next step $(+1)$, this assumption can be changed if required. Starting an activity can

Table 1. Activity state transitions axioms

$A.\text{ready} > 0 \rightarrow (A.\text{started} \geq A.\text{ready} \wedge A.\text{skipped} = -1) \vee$ $(A.\text{skipped} = A.\text{ready} + 1 \wedge A.\text{started} = -1)$	(N_1)
$A.\text{started} > 0 \rightarrow ((A.\text{faulted} > A.\text{started}) \vee$ $(A.\text{terminated} > A.\text{started}) \vee$ $(A.\text{executed} > A.\text{started}))$	(N_2)
$A.\text{started} > 0 \rightarrow ((A.\text{faulted} = -1 \wedge A.\text{terminated} = -1) \vee$ $(A.\text{faulted} = -1 \wedge A.\text{executed} = -1) \vee$ $(A.\text{executed} = -1 \wedge A.\text{terminated} = -1))$	(N_3)
$A.\text{started} > 0 \leftarrow (A.\text{faulted} > 0 \vee A.\text{terminated} > 0 \vee A.\text{executed} > 0)$	(N_4)
$A.\text{compensated} > 0 \rightarrow A.\text{executed} > 0 \wedge A.\text{compensated} > A.\text{executed}$	(N_5)
$A.\text{skipped} > 0 \rightarrow A.\text{completed} = A.\text{skipped}$	(N_6)
$A.\text{executed} > 0 \rightarrow A.\text{completed} = A.\text{executed}$	(N_7)
$A.\text{completed} > 0 \rightarrow A.\text{executed} > 0 \vee A.\text{skipped} > 0$	(N_8)

on the other hand take longer if we consider resource requirements, therefore we use the \geq operator.

- Axioms $(N_2) - (N_4)$ state that if an activity has been started, then it will either complete successfully, fail, or be terminated, whereby only one of these states can be reached.
- Axiom (N_5) specifies that an activity can only be compensated if it has been executed.
- Axioms $(N_6) - (N_8)$ state that an activity reaches state completed only if and as soon as it has been skipped or executed.

Table 2. Fault, termination, and compensation handler axioms

$A_F.\text{ready} = A.\text{faulted}$	(F_1)
$A_T.\text{ready} = A.\text{terminated}$	(F_2)
$A_C.\text{started} > 0 \rightarrow A.\text{executed} > 0$	(F_3)
$A_C.\text{executed} > 0 \rightarrow A.\text{compensated} = A_C.\text{executed}$	(F_4)

If there is a fault handler A_F , compensation handler A_C or termination handler A_T defined for activity A , then the additional state transition axioms depicted in Table 2 need to be added to the general state transition axioms.

2.2 Parent-Child Axioms

In addition to the relations between the states of a single activity, relations between parent and child activities represented through the hypernodes, which represent process scopes, need to be specified. If A_i is a node inside the hypernode A , then we call A the parent activity of child activity A_i .

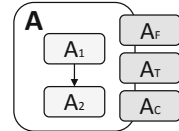

Fig. 3. Parent-Child

Figure 3 shows an example hypernode with two children activities, Figure 4 shows relations between the states of the parent and child activities.

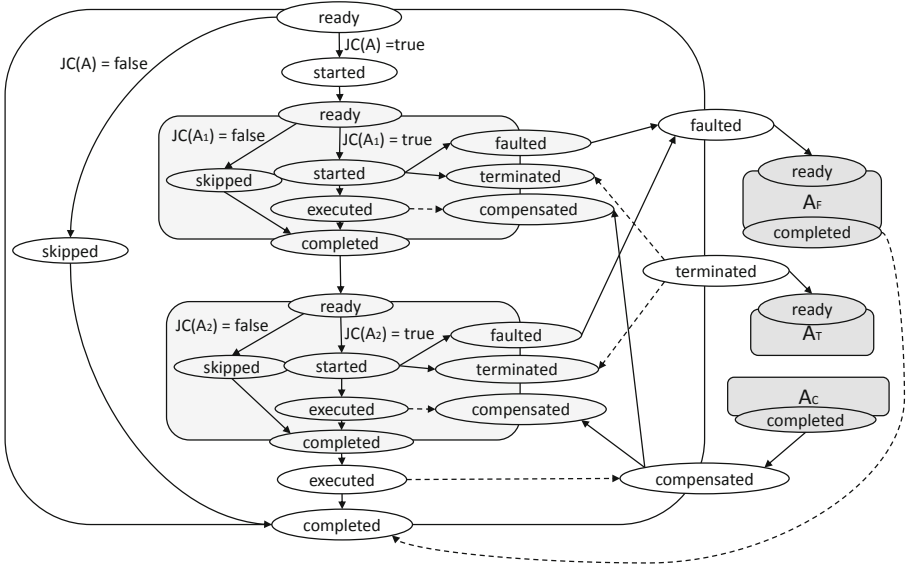


Fig. 4. Parent-Child State Transitions

Table 3. Parent-child state dependencies axioms

$A_i.\text{ready} > 0 \rightarrow A.\text{started} > 0 \wedge A_i.\text{ready} \geq A.\text{started}$	(H ₁)
$A.\text{skipped} > 0 \rightarrow A_i.\text{skipped} > 0 \wedge A_i.\text{skipped} = A.\text{skipped}$	(H ₂)
$A.\text{completed} > 0 \rightarrow$ $\bigwedge_{i \in [1..n]} (A_i.\text{completed} > 0 \wedge A.\text{completed} \geq A_i.\text{completed})$	(H ₃)
$A.\text{terminated} > 0 \rightarrow$ $(A_i.\text{completed} > 0 \wedge A_i.\text{completed} < A.\text{terminated}) \vee$ $(A_i.\text{started} > 0 \wedge A.\text{terminated} = A_i.\text{terminated}) \vee$ $(A_i.\text{ready} < 0)$	(H ₄)
$A_i.\text{faulted} > 0 \rightarrow A.\text{faulted} > 0 \wedge A.\text{faulted} = A_i.\text{faulted}$	(H ₅)
$\forall i \in [1..n] : A_i.\text{faulted} > 0 \rightarrow (\forall j \in [1..n] : j \neq i \rightarrow$ $((A_j.\text{ready} > 0 \wedge A_j.\text{terminated} = A_i.\text{faulted}) \vee$ $(A_j.\text{completed} > 0 \wedge A_j.\text{completed} < A_i.\text{faulted}) \vee$ $A_j.\text{ready} < 0)$	(H ₆)
$A_C.\text{ready} > 0 \rightarrow A_{i_C}.\text{ready} > A_C.\text{ready}$	(H ₇)
$A_C.\text{completed} \geq A_{i_C}.\text{completed}$	(H ₈)
$\forall (A_i, A_j) \in L : (A_i \in \text{Children}(A) \wedge A_j \in \text{Children}(A))$ $\rightarrow (A_{i_C}.\text{ready} > A_{j_C}.\text{completed})$	(H ₉)

Table 3 contains a set of axioms that capture parent-child state relations:

- Axiom (H_1) specifies that a child activity can only be reached if the parent activity has started.
- Axiom (H_2) specifies that if parent activity is skipped, then all children activities are skipped too.
- Axiom (H_3) specifies that a parent activity completes when all child activities complete
- Axiom (H_4) specifies the termination semantic: If the parent activity is terminated, all of its children activities, which are still running, are terminated. Therefore all children activities will either be completed at the point of parent activity termination, will not be reached yet, or will be terminated together with the parent activity (immediate termination).
- Axiom (H_5) specifies the fault propagation semantic: Fault propagation is applied if there is no explicit fault handler defined for activity A_i . In this case any fault of any child activity is propagated to the parent activity. If a child activity has a fault handler, then the default fault propagation behaviour is overwritten by the fault-handler execution semantic. In this case activities defined in the fault handler are modelled using the same set of axioms, and dependency between fault handler and corresponding activity is modelled according to the fault axiom F_1 .
- In addition to the fault propagation, axiom (H_6) specifies that if an activity has faulted, then all still running activities in the same scope are terminated.
- Axioms (H_7) – (H_9) specify the default compensation handler semantic, which invokes compensation handlers of the child activities in a reverse order. Here A_C denotes the compensation handler of the parent activity A , and A_{iC} denotes compensation handler of a child activity A_i .

2.3 Process Structure Axioms

Table 4 lists axioms that reflect dependencies between activities based on the process structure.

- (S_1) models synchronisation dependencies of a process activity A_i : An activity in a process graph is executed if it has been reached and its join condition evaluates to *true*. An activity is reached if all predecessor activities have completed their execution, which means $A.completed > 0$ is *true*. This can happen through activity execution, activity skipping, or reparation of a faulted activity through a fault handler as described in previous section. Each link in a flow graph represents a synchronisation dependency between the source and the target nodes. We model each link in a process graph as a variable of type $Link := Record(source : Activity, target : Activity, condition : bool)$, where *condition* represents the link transition condition. $D(A_i)$ denotes the set of nodes that activity A_i is synchronised on. This set is computed as $D(A_i) = \{A_j | \exists L_k \in L^{in}(A_i), L_k.source = A_j\}$.
- Axiom (S_2) specifies that if all predecessor activities of activity A_i have completed, then A_i will reach state ready.

- S_3 and S_4 specify the link status evaluation rules for any link L .
- Axioms (S_5) and (S_6) reflect the control flow after an activity has been reached: if activity join condition evaluates to *true*, then activity will start as defined by (S_5) , otherwise it will be skipped as defined by (S_6) .

Table 4. Process structure axioms

$\forall A_j \in D(A_i) :$	
$(A_i.\text{ready} > 0 \rightarrow A_i.\text{ready} \geq A_j.\text{completed} \wedge A_j.\text{completed} > 0)$	(S_1)
$\bigwedge_{A_j \in D(A_i)} A_j.\text{completed} > 0 \rightarrow A_i.\text{ready} > 0$	(S_2)
$L.\text{source.executed} > 0 \rightarrow \text{Status}(L) = L.\text{condition}$	(S_3)
$L.\text{source.skipped} > 0 \rightarrow \text{Status}(L) = \text{false}$	(S_4)
$A.\text{ready} > 0 \wedge \text{JC}(A) \leftrightarrow A.\text{started} > 0$	(S_5)
$A.\text{ready} > 0 \wedge \neg \text{JC}(A) \leftrightarrow A.\text{skipped} > 0$	(S_6)

3 Property Assertions

A property assertion guarantees certain behaviour of a process without revealing how it has been implemented. Therefore property assertions allow participants in a process collaboration to hide the process implementation details and offer a natural form of abstraction. Due to their descriptive, yet formal nature, property assertions can be used as a declarative description of a subprocess in the bottom-up process composition, as well as a requirement specification for a subprocess in the top-down process design, called process refinement.

Property assertions make statements over activity types rather than actual activities. This supports specification of properties with the different abstraction levels, depending on the granularity of the chosen activity type. If we define a property assertion PA over activity type *Transport* which is defined by interval $[0, 10]$, we would restrict the set of activities these assertions apply to as follows: $\forall A : (A.\text{atype} \geq 0 \wedge A.\text{atype} \leq 10) \rightarrow PA(A)$. Using more fine-granular activity types allows for specification and verification of more specific constraints, but it might reveal unnecessary details.

In this work we use an SMT solver to prove collaboration properties. Usage of SMT solvers allows us to specify requirements in propositional logic, as well as use additional theories, such as linear arithmetic theory. First Order Logic (FOL) over finite domains can be mapped to the propositional logic over the elements of the corresponding domain. The domain of an activity type is defined by the number of activities of this type present in the process model. As there is a limited number of activities in a process model, we will use FOL over activity types to express properties of the corresponding activities.

In the following sections we will show some examples of the process properties that can be expressed using combination of linear arithmetic with the logical operators. The examples are chosen to demonstrate expressiveness of the language. To ease property specification for a business user, property templates for the common properties can be defined similar to the Count and *FollowedBy* templates that are defined in the following sections.

3.1 Activity Occurrence Specifications

A restriction on the number of activity executions of a specific type is specified through $N_{min} \leq \text{Count}(A^T) \leq N_{max}$ constraint, which allows execution of minimum N_{min} and maximum N_{max} activities of type A in any process run. We define Count function of an activity A_i and of activity type A^T as follows:

$$\text{Count}(A^T) = \sum_{A_i \in A^T} \text{Count}(A_i) = \sum_{A_i \in A^T} \begin{cases} 1 & \text{if } A_i.\text{executed} > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3.1)$$

Although activity state `.executed` is the one that will be most commonly used to express activity occurrence, we can extend count function to allow count of any other activity states, that would allow to count the number of, e.g., failed or compensated activities. The extended count function $\text{Count}(A^T, S)$ counts the number of activity of type A_T that reach state A_S :

$$\text{Count}(A_i, S) = \begin{cases} 1 & \text{if } A_i.S > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3.2)$$

In some cases we will want to restrict time period for an activity state transfer. For this purposes we can refine Count template with the time parameters:

$$\text{Count}(A^T, S_1, S_2)_{[t_1, t_2]} = \sum_{A_i \in A^T} \begin{cases} 1 & A_i.S_1 \geq t_1 \wedge A_i.S_2 \leq t_2, \\ 0 & \text{otherwise.} \end{cases} \quad (3.3)$$

Where t_1 and t_2 can be relative or absolute timestamps.

Using refined count template we can specify that minimum *MAX* and maximum *MIN* activities executions of type A^T must happen in time period $[t_1, t_2]$ as follows:

$$MAX \leq \text{Count}(A^T, \text{started}, \text{executed})_{[t_1, t_2]} \leq MIN \quad (3.4)$$

3.2 Temporal Properties Specifications

To capture temporal relations between activities we specify relations between activity state fields using linear arithmetic operators $>$, $<$, \geq , \leq and $=$. In addition, we use operators $+$ and $-$ over activity state variables and integers representing time intervals.

Having different states for each activity allows us to restrict activity execution duration as follows:

$$\begin{aligned} A.\text{executed} > 0 \rightarrow A.\text{completed} - A.\text{started} &\geq \text{MinDuration}(A) \\ \wedge A.\text{executed} - A.\text{started} &\leq \text{MaxDuration}(A) \end{aligned} \quad (3.5)$$

Similarly, temporal dependencies between different activities and their states can be specified using arithmetic operators. To demonstrate the expressiveness of the

requirement specification language we consider *FollowedBy* example constraint that has been often used in the process verification approaches and show how different semantic variations of this constraint can be specified using FOL over activities, predicates and operators from linear arithmetic over activity states, and Count function defined above.

The *FollowedBy* constraint used in [19] is the A^T *must be followed by* B^T . To express such a requirement in our model we use activity state variables and linear arithmetic to specify the *FollowedBy* template as follows:

$$\begin{aligned} \text{FollowedBy}(A^T, S_1, B^T, S_2) := \forall A \in A^T : A.S_1 > 0 \rightarrow \\ \exists B \in B^T : B.S_2 > A.S_1 \end{aligned} \quad (3.6)$$

Similar to *FollowedBy* template, *PrecededBy* template can be defined as follows:

$$\begin{aligned} \text{PrecededBy}(A^T, S_1, B^T, S_2) := \forall A \in A^T : A.S_1 > 0 \rightarrow \\ (\exists B \in B^T : B.S_2 > 0 \wedge B.S_2 < A.S_1) \end{aligned} \quad (3.7)$$

We assume that *FollowedBy* template has default parameters $S_1 = \textit{executed}$ and $S_2 = \textit{started}$, as well as that *PrecededBy* has default parameters $S_1 = \textit{started}$ and $S_2 = \textit{executed}$. This means that semantic of $\text{FollowedBy}(A^T, B^T)$ is equivalent to $\text{FollowedBy}(A^T, \textit{executed}, B^T, \textit{started})$. By varying activity states in these templates we can slightly change their semantic, for example $\text{FollowedBy}(A^T, \textit{started}, B^T, \textit{started})$ would allow parallel executions of $A \in A^T$ and $B \in B^T$ as long as B starts after A has started.

To extend the *FollowedBy* requirement with the time restrictions as used in [8] to express B^T *must follow* A^T *within* N *time units* constraint, we can define the *FollowedByWithin* template as follows:

$$\begin{aligned} \text{FollowedByWithin}(A^T, S_1, B^T, S_2, T) := \forall A \in A^T : \\ A.S_1 > 0 \rightarrow \exists B \in B^T : (B.S_2 > A.S_1) \\ \wedge (B.S_2 < A.S_1 + T) \end{aligned} \quad (3.8)$$

Constraint $\text{FollowedByWithin}(A^T, \textit{executed}, B^T, \textit{started}, 2)$ specifies that B must start after A if it has been executed and has to finish its execution within 2 time units. It can be varied to specify that B must be *completed* within 2 hours after execution of A or that it has to reach a certain state in a time frame relative to start of A . In general, any states defined in Section 2.1 can be used to parameterise these templates.

3.3 Data Dependent Properties

A property assertion can be refined by adding data conditions over the process input data D_i that influence execution of activity. To specify that A_i will be executed if and only if $C(D_i)$ evaluates to true, we use following property assertion:

$$(\text{Count}(A_i, \textit{executed}) \leq 1) \wedge (C(D_i) \leftrightarrow \text{Count}(A_i, \textit{executed}) = 1)$$

If activity execution condition contains expressions over local variables or if the exact activity execution condition should not be exposed, it can be abstracted to a necessary or a sufficient condition. A sufficient condition $C(D_i)$ leads to execution of the activity: $C(D_i) \rightarrow \text{Count}(A_i, \text{executed}) > 0$, while necessary condition is fulfilled when activity is executed: $\text{Count}(A_i) = 1 \rightarrow C(D_i)$. Similar to the activity occurrence constraint, any other property specification can be refined with linear data conditions over the input data.

4 Hierarchical Process Verification Using Process Theory

In this section we will show how a requirement R over the parent process P can be proven or disproven based on the property assertions $PA(P_i)$ of the subprocesses P_1, \dots, P_n , and on the structure of P .

4.1 Representing Subprocess Abstractions

Property assertions of a subprocess make statements about the internal process behaviour. Such assertions need to be adapted to be included into the global context of the overall process. For example, consider a subprocess P_k that makes a statement $\text{Count}(A_i^T, \text{executed}) = 1$ with respect to its behaviour. Furthermore, consider a subprocess P_j that makes a statement over its behaviour about the same activity type $2 \leq \text{Count}(A_i^T, \text{executed}) \leq 3$. Without adjustment of these assertions to the global context, the combined assertion set would contain both statements, which obviously would lead to a contradiction.

To adjust a local assertion to the global context, we create a set of unique variables for each subprocess that represent internal activities of the subprocess. In the above example we would create a variable $A_{i_1}^{P_k}$ that would represent activity of type A_i in process P_k , and three variables $A_{i_1}^{P_j}$, $A_{i_2}^{P_j}$ and $A_{i_3}^{P_j}$ to represent activities of type A_i in process P_j . Count assertion is then mapped to the assertions over the internal activities. For P_k it will be equivalent to

$$A_{i_1}^{P_k}.\text{executed} > 0$$

And for P_j it will be equivalent to

$$A_{i_1}^{P_j}.\text{executed} > 0 \wedge A_{i_2}^{P_j}.\text{executed} > 0$$

Without loss of generality we specified that the first two activities are always executed. There is no statement about the third activity, leaving a freedom for SMT solver to decide whether it will be executed or not.

Let us assume a subprocess P_i provides a set of assertions $PA(P_i)$ over activity types $T(PA(P_i)) = \{A_{i_1}^T, \dots, A_{i_k}^T\}$. Let us assume that the overall process requirement R is defined over activity types $T(R) = \{A_1^T, \dots, A_m^T\}$. In the first step we need to create a generic model of P_i with respect to the overall requirement R . A model consists of the subprocess activities represented through the

corresponding variables, as well as the property assertions of the subprocesses mapped to these activities. As implementation of P_i is unknown, we do not know how many activities of each type can occur in P_i . To generate a suitable number of activity instances, we use the following approach.

1. For each activity type $A_i^T \in T(PA(P_i))$ we generate as many instances as specified by the maximum count constraint for this activity type in $PA(P_i)$. If no maximum count constraint is defined for this activity type, we generate D number of activity instances, where D is the depth of the verification.
2. For each $A_j^T \in T(R)$ if $A_j^T \subseteq \bigcup_{A_{i_k}^T \in T(PA(P_i))} A_{i_k}^T$, meaning that activity type A_j^T is covered by types mentioned in $PA(P_i)$, then activities for this type have already be generated in the first step. If A_j^T type refers to a type that is not part of or is broader than types from $T(PA(P_i))$, then two strategies can be taken:
 - (a) We assume a closed world model (if subprocess does not mention this type, then it does not have it) and generate no additional activities of this type.
 - (b) Introduce an additional negotiation step, where each subprocess will be asked to provide information with respect to this type.
 - (c) Assume open world model: as property assertions only represent part of the process functionality, it can happen that the subprocess contains activities relevant for the overall requirement that are not reflected in the property assertions. In this case we would generate D activities of type $A_j^T \setminus \bigcup_{A_{i_k}^T \in T(PA(P_i))} A_{i_k}^T$.

4.2 Verifying Process Requirement

Let process P contain sub-processes P_1, \dots, P_n . Let $PA(P_i) = PA_1(P_i), \dots, PA_k(P_i)$ denote property assertion of sub-process P_i . Let $SA(P)$ denote structural assertions derived from the structure of process P , which include temporal as well as execution condition dependencies of P_1, \dots, P_n . Let $Act(P_i)$ denote specifications of internal activities of P_i as described above, including activity state axioms and child-parent relations between P_i and specified internal activities. Then a requirement R on process P is fulfilled iff the following is fulfilled:

$$\left(SA(P) \wedge \bigwedge_{i \in 1..n} Act(P_i) \wedge (P_i.executed > 0 \rightarrow \bigwedge_{PA_j \in PA(P_i)} PA_j) \right) \rightarrow R \quad (4.1)$$

Including different activity states allows us to verify certain properties of fault behaviour, but requires additional assertions to be added if we want to restrict property verification to the non-faulty behaviour. For this purpose we need to specify that all of the process activities must reach state *executed* if they reach state *started*. This would force states *faulted* and *terminated* to be set to -1 , which reflects the non-faulty execution of this activity. To force verification of the non-faulty behaviour for any activity A_i , we add the following assertion:

$$A_i.started > 0 \rightarrow A_i.executed > 0 \quad (4.2)$$

If on the other hand faulty behaviour of certain activities should be taken in consideration during verification, assertion 4.2 should be skipped for such activities.

4.3 Application to the Case Study

In the motivating example we have four subprocesses modelled in the process graph, represented through the hypernodes, which represent abstractions of the corresponding processes. These are *Retailer*₁, *Reseller*, *Producer* and *Retailer*₂. To model these subprocesses we declare four variables *Retailer*₁, *Reseller*, *Producer* and *Retailer*₂ to be of type *Activity* and add the corresponding state relation axioms to our verification basis. The type of these activities has not been specified, therefore we do not restrict *atype* field of these variables. Our example model also contains one data variable *Quantity*, which is added to the verification basis as an *Integer* variable. Next we will show how the abstract processes are modelled using *Reseller* activity and corresponding property assertions as an example.

The *Reseller* does not specify activity occurrence restrictions, therefore to generate internal activities for this abstracted process we need to choose the depth parameter. For this example we choose verification depth $D = 2$ and use closed world assumption, which means that we will create two activities of each type declared by the process abstraction and create no additional activities. In our case we create two activities ED_1, ED_2 of type *ExpressDelivery* and two activities QC_1, QC_2 of type *QualityCheck*.

After activity declaration and addition of the activity axioms, we add parent-child axioms according to Table 3. An example axiom (H_1) applied to activities of type *ExpressDelivery* looks as follows:

$$\begin{aligned}
 ED_i.ready > 0 &\rightarrow Reseller.started > 0 \wedge ED_i.ready \geq Reseller.started \\
 ED_2.ready > 0 &\rightarrow Reseller.started > 0 \wedge ED_2.ready \geq Reseller.started
 \end{aligned}$$

Next we will specify property assertions of *Reseller* mapped to the generated internal activities. The first property assertion PA_1 of *Reseller* is *If (Quantity < 5000) THEN ExpressDelivery*, which can formally be represented as $(Quantity < 5000) \rightarrow Count(ExpressDelivery, executed) > 0$. This assertion is mapped to the internal activities as follows:

$$PA_1 = (Quantity < 5000 \rightarrow (ED_1.executed > 0 \vee ED_2.executed > 0))$$

The second property assertion PA_2 of *Reseller* is *PrecededBy(ExpressDelivery, QualityCheck)* is equivalent to $\forall A \in ExpressDelivery : A.started > 0 \rightarrow (\exists B \in QualityCheck : B.executed > 0 \wedge B.executed < A.started)$ To map the \forall quantor to the internal activities, we need to add the corresponding assertions for each generated activity of type *ExpressDelivery*. In our case we map $PA_2^{ED_i}$, where $i \in [1, 2]$ denotes the index of the generated *ExpressDelivery* activity, to the generated process activities as follows:

$$PA_2^{ED_i} = \begin{aligned} & ED_i.\text{executed} > 0 \rightarrow \\ & ((QC_1.\text{executed} > 0 \wedge QC_1.\text{executed} < ED_i.\text{executed}) \\ & \vee (QC_2.\text{executed} > 0 \wedge QC_2.\text{executed} < ED_i.\text{executed})) \end{aligned}$$

In the next step we add process structure axioms to the verification basis according to Section 2.3. For this we first define link variables of type *Link* for each process link:

$$Link := Record(\text{source} : Activity, \text{target} : Activity, \text{condition} : bool)$$

An example link between *Retailer*₁ and *Reseller* is defined as

$$L_1 : Link(\text{source} = Retailer_1, \text{target} = Reseller, \text{condition} = Quantity < 2000)$$

Next we apply axioms from Table 4 to each of the process activities and each of the process links. Here is an example of S_1 , S_2 and S_5 axioms applied to the *Reseller*:

$$\begin{aligned} Reseller.\text{ready} > 0 &\rightarrow Reseller.\text{ready} \geq Retailer_1.\text{completed} \\ &\quad \wedge Retailer_1.\text{completed} > 0 \\ Retailer_1.\text{completed} > 0 &\rightarrow Reseller.\text{ready} > 0 \\ Reseller.\text{ready} > 0 \wedge Status(L_1) &\leftrightarrow Reseller.\text{started} > 0 \end{aligned}$$

In addition, we specify that we only want to verify non-faulty runs through elimination of faulty behaviour using rule 4.2. Applied to *Producer* this rule looks as follows ²:

$$Producer.\text{started} > 0 \rightarrow Producer.\text{executed} > 0$$

After modelling the process through the assertions according, we can prove the collaboration requirement R : *Product quality must be checked before the product is sold*, which is mapped to the generated activities as follows:

$$\begin{aligned} SellProduct_1.\text{started} > 0 &\rightarrow \\ & ((QC_1.\text{executed} > 0 \wedge SellProduct_1.\text{started} > QC_1.\text{executed}) \\ & \vee (QC_2.\text{executed} > 0 \wedge SellProduct_1.\text{started} > QC_2.\text{executed}) \\ & \vee (QC_3.\text{executed} > 0 \wedge SellProduct_1.\text{started} > QC_3.\text{executed})) \end{aligned}$$

Using SMT solver Z3 [13], we add the process model assertions and negation of the requirement to the verification context and check its satisfiability. The solver returns UNSAT, which means that the requirement R is fulfilled. If we slightly modify property assertions provided by the participants, e.g. change *Reseller* guarantee for express delivery for quantities under 1000, then the SMT solver returns SAT with a model that represents violation of the requirement. The model returned by SMT solver assigns a value between 1000 and 2000 to the quantity variable, and sets $QC_1.\text{executed}$, $QC_2.\text{executed}$, $ED_1.\text{executed}$ and $ED_2.\text{executed}$ to *false*.

² We can skip corresponding assertions for the internal activities as their fault or termination would lead to fault or termination of the parent activities, which is forbidden through the above assertions.

4.4 Performance Discussions

SMT solvers have been developed in academia and industry with increasing scope and performance [3]. However, as the problem is NP-hard, the verification of large models can still take quite a lot of time. The presented approach can naturally cope with the large models by abstracting parts of the process through the property assertions. Process abstraction through property assertions not only allows participants to hide their implementation details, but also reduces the size of the model by removing unnecessary details. To additionally improve performance we can further reduce the model size by decreasing the *Depth* parameter for generation of the internal activities (depth of 1 is sufficient in a lot of cases), or through simplification of the activity state model. The number of integer variables required to represent N activities is $N \times K + 1$, where K is the number of activity states (1 is for the activity type variable). We can further control the model size by deciding which activity states are required for each verification case. In this work we use 8 state variables for each activity. Based on the verification requirements this number can be reduced to 3 basic states: *ready*, *executed* and *skipped*, which would suffice to analyse *normal* behaviour of a process; or it can be extended with *repaired*, *cancelled* and any other relevant states if required.

5 Related Work

Constraints have been widely used in the business process area for process specification, annotation, verification and validation. Our process modelling approach is closely related to the *flow* construct of the Business Process Execution Language (BPEL, [17]). An extensive overview of existing BPEL formalisations and verification approaches is provided in [4]. In [14] the constraints are used to model the semantic of a BPEL process. This allows verification of other constraints against the set of constraints representing a BPEL process by reducing the constraint verification problem to the constraint satisfiability problem. The approach analyses data flow together with the control flow, which allows verification of the data dependent properties. The data mapping approach presented in this paper was applied to the current work. In [15] an approach for modelling and validation of different constraint types based on the geometrical shape of a business process was presented. Property assertion language presented in this paper heavily influenced the property assertion language in the current work. A lot of work [9,1,10,11] exist in the area of business process verification using petri nets. These works concentrate on verification of a workflow where all the implementation details are known. In [19,18] authors present an LTL based constraint specification language, that is used to specify declarative workflows. In every step of a workflow execution a set of the reachable is computed, so that a user cannot execute an activity which does not lead to a successful process termination. While this work is related to the property specification language, our property assertion language allows to specify data-dependent properties, as well as activity deadlines. An approach based on abstract state machines (ASM)

is presented in [6]. While the mapping covers scopes, it does not consider the relations between data conditions and activity executions. Approaches based on the π -calculus are presented in [5,12]. As with the ASM approaches, these do not consider the data dependencies of the process. In [7] the authors presented a formal approach for modeling and verification of web service composition using finite state process (FSP) notation. Similarly to the Petri Nets approach this approach does not cover analysis of the activity dependency on the process data and data manipulations.

In contrast to other works, we use property assertions to declaratively describe parts of a process. The presented verification approach uses property assertions instead of the process models, which differentiates this work from the related work.

6 Summary and Future Work

In this paper we presented a hierarchical process verification approach that allows process participants to avoid disclosure of their subprocesses and still allow for verification of certain properties of the parent process. We showed how a hierarchical process model can be mapped to assertions, starting from a single activity and its state axioms, through abstracted subprocesses with their guarantees and child-parent axioms, to the complete process structure axioms. Using the generated assertions we showed how a requirement over the process composition can be verified using an SMT solver.

Acknowledgments. The research leading to these results has received funding from the German “Federal Ministry of Education and Research” in the context of the project “RescueIT”.

References

1. Van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
2. Beckert, B., et al.: Intelligent Systems and Formal Methods in Software Engineering. *IEEE Intelligent Systems* 21(6), 71–81 (2006)
3. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. *Frontiers in Artificial Intelligence and Applications*, vol. 185. IOS Press, Amsterdam (2009)
4. van Breugel, F., Koshkina, M.: Models and Verification of BPEL (2006), <http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf>
5. Fadlisyah, M.: Using the π -calculus for modeling and verifying processes on web services. Master’s thesis, Institute for Theoretical Computer Science, Dresden University of Technology (2004)
6. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative control flow. In: 12th International Workshop on Abstract State Machines, pp. 131–151 (March 2005)

7. Foster, H., Uchitel, S., Magee, J., Kramer, J.: A Model-Based Approach to Engineering Web Service Compositions and Choreography in Test and Analysis of Web Services. In: Baresi, L., Di Nitto, E. (eds.), ch. 71–91, pp. 72–91. Springer-Verlag Berlin and Heidelberg GmbH & Co. (2007)
8. Giblin, C., Liu, A.Y., Müller, S., Pfizmann, B., Zhou, X.: Regulations expressed as logical models (realm). In: JURIX, pp. 37–48 (2005)
9. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
10. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
11. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer, Heidelberg (2007)
12. Lucchia, R., Mazzara, M.: A pi-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming* 70(1), 96–118 (2007)
13. Microsoft Research. Z3 an efficient theorem prover,
<http://research.microsoft.com/en-us/um/redmond/projects/z3/>
14. Monakova, G., et al.: Verifying Business Rules Using an SMT Solver for BPEL Processes. In: BPSC (2009)
15. Monakova, G., Leymann, F.: Workflow art: A framework for multidimensional workflow analysis. In: *Enterprise Information Systems* (2012)
16. Nelson, G., Oppen, D.: Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems* 1(2), 245–257 (1979)
17. OASIS. Web Services Business Process Execution Language Version 2.0 (2007)
18. 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)
19. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)

Intentional Fragments: Bridging the Gap between Organizational and Intentional Levels in Business Processes

Mario Cortes-Cornax¹, Alexandru Matei², Emmanuel Letier²,
Sophie Dupuy-Chessa¹, and Dominique Rieu¹

¹ University of Grenoble, CNRS, LIG

{Mario.Cortes-Cornax,Sophie.Dupuy,Dominique.Rieu}@imag.fr
<http://sigma.imag.fr/>

² University College London, Gower Street, London WC1E 6BT, United Kingdom
alexandru.matei.09@ucl.ac.uk, e.letier@cs.ucl.ac.uk

Abstract. Business process models provide a natural way to describe real-world processes to be supported by software-intensive systems. These models can be used to analyze processes in the system-as-is and describe potential improvements for the system-to-be. There is however little support to analyze how well a given business process models satisfies its business goals. Our objective is to address these problems by relating business process models to goal models so that goal-oriented requirements engineering techniques can be used to analyze how well the business processes for the system-as-is satisfy the business goals. The paper establishes relationships between BPMN 2.0 and the KAOS goal-oriented requirements modelling framework. We present the notion of intentional fragment to bridge the gap between process models and goal models. We conducted an evaluation to analyze use of this concept in the context of a university process.

Keywords: Process Modelling, Business Process Management, Goal-oriented Requirements Modelling, KAOS, BPMN 2.0.

1 Introduction

Business process models provide a natural way to describe real-world processes to be supported by software-intensive systems. These models are widely used in the industry as an important source of information about the current or future processes in a company. A widely recognized problem among the business analysts is the lack of a clear correspondence between business process models and business objectives, rules and constraints [1]. This fact decreases the value of such models, since it keeps the rationale behind each process implicit [2,3]. These problems have also been discussed in the context of the Business Process Model and Notation (BPMN) [4]. Indeed, most practitioners point out the lack of business rules behind BPMN models [5].

Different approaches have been proposed in the academia for relating business process models with business objectives or constraints using frameworks such as Non-Functional Requirements (NFR) [6], i* [7], Tropos [8] or KAOS [9]. Their scope ranges from establishing semantic correspondences between process models and goal models [8] to addressing non-functional requirements satisfaction [6] or process variability and re-engineering methods [7]. These approaches either assume a pre-existing goal model or define a goal model too tied to the process model. They are generally not focused on how to create a goal model in the first place. However, generating a useful goal model represents a challenge for most business analysts.

Our aim is to relate business process models to goal models maintaining a clear separation of concerns between the two models. Traceability links between these two models will allow the business analyst to explicitly state the rationale of each process activity. A goal-based analysis based on this relation can therefore be applied to identify problems in the process model, such as missing or superfluous activities.

The paper establishes the relationship between BPMN 2.0 and the KAOS goal-oriented modelling framework through the concept of *Intentional Fragment*. An intentional fragment is a set of flow elements of the process with a common purpose. By means of intentional fragments goals are therefore related to the BPMN 2.0 process elements. The paper presents several heuristics to extract potential intentional fragments from the business process that will help constructing goal models from the business process model and also guide the goal-based analysis. We conducted an evaluation using the mission process in the Informatics Laboratory of Grenoble (LIG) to analyze the use of intentional fragment as an efficient and simple way to fill the gap between the organizational level represented by the business process model and the intentional level represented by the goal model.

The paper is organized as follows. Section 2 briefly presents BPMN 2.0 and KAOS framework thought a model used in our case study. Section 3 presents the relation between these two models and a precise definition of the notion of intentional fragment. The case study is discussed in Section 4. In Section 5 analysis questions derived from the notion of intentional fragment are raised. Section 6 presents the related works and finally, future work and conclusions are discussed in Section 7.

2 BPMN 2.0 Process Models and KAOS Goal Models

This section presents both BPMN 2.0 and KAOS languages through a running example that relies on the mission process (e.g., conference travel or speech invitation) in the Informatics Laboratory of Grenoble (LIG). We choose BPMN 2.0 since it is the de-facto standard to model business processes. KAOS on the other hand is a well known framework for goal modelling which comes with a powerful set of goal oriented analysis techniques.

2.1 Running Example Modelled in BPMN 2.0

The main scope of BPMN is to describe business processes in an accessible way at different levels of granularity (i.e., from abstract design models to detailed executable models). Figure 1 shows a design model of the mission process in the Informatics Laboratory of Grenoble (LIG) modelled in BPMN 2.0. A *Process* in BPMN 2.0 is defined as “a sequence or flow of activities in a specific organization with the objective of carrying out work” [4]. A *Process* in BPMN 2.0 might be enclosed in a *Pool* which identifies the process responsible (*Participant*)(e.g. *Travel Agency* and *LIG*). To organize and categorize *Activities* within a *Pool*, *Lanes* are used which usually represent different organizational units in a process (e.g. *Team Leader* and *Employee*). The flow of *Activities*, which represent the work (e.g. “E1: set the travel schedules”), is controlled by *Gateways*, which are the decision making (e.g. “Ok?”). A process starts with a *Start Event* and can finish in different ways captured by the *End Events*.

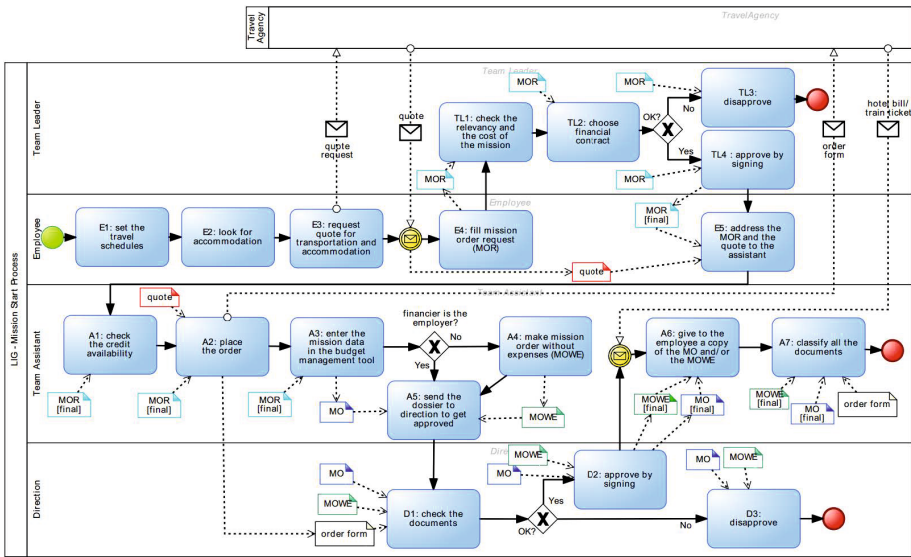


Fig. 1. Mission Process - Before Leaving

Figure 1 describes the steps that permit an *Employee* go in a journey as for example a conference, and then be refunded. An *Employee* must look for convenient travel times and hotel for her destination firstly. Then, she asks for a quote (*quote request*) to the *Travel Agency*. This request is a *Message* linked to a *Sequence Flow*. She calculates the mission costs filling the *Mission Order Request (MOR)* where she add the estimation for the staying expenses in addition to transport and hotel expenses. The *Team Leader* checks the appropriateness and cost of the mission and then approves or disapproves the mission. The *Team Leader* also chooses the contract from which the mission will be financed. Both

the *MOR* and the *quote* are addressed to the *Team Assistant* who is in charge of doing all the administration documents so that the *Employee* can leave with warranties to be covered by an assurance and with the *Direction* approval. We present just the first part of the process model, before the *Employee* leaves. Further details in the BPMN 2.0 constructs may be found in the standard [4].

2.2 Goal-Oriented Requirements Modelling in KAOS

Goal-oriented requirements engineering (GORE) supports “the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements” [10].

The KAOS method for GORE offers a precise (formal) way to reason about Requirements Engineering (RE). It defines several complementary system models, including the goal model. *Goals* captured in the goal model are prescriptive statements about the system, capturing desired states or conditions. Goals are organized hierarchically, starting from high level goals (usually corresponding to business objectives). These goals are iteratively refined into *Sub-goals*. Requirements are under the responsibility of *Agents*. Agents are the active components inside the system.

Figure 2 shows an example of goal model (not fully developed) concerning the mission process of Fig. 1. On top, we find the high level goal “Go in a mission comfortably and covered by an assurance” refined into further sub-goals leading to requirements. The responsibility of the requirements are then assigned to the agents. For example, the *Employee* is responsible of the goal (requirement) “Program an informal discussion with the Team Leader to present the mission”.

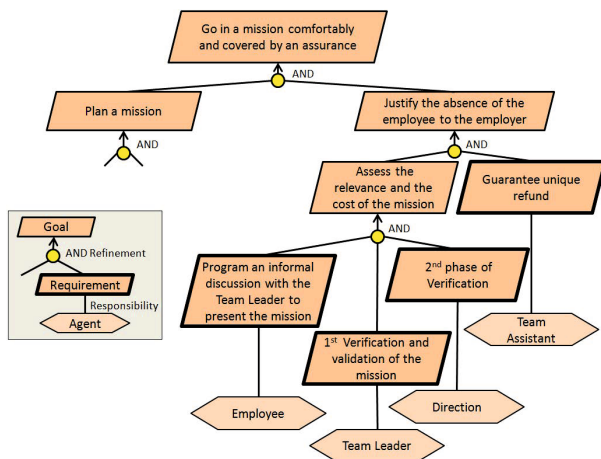


Fig. 2. Example of Goal Model related to the Mission Process

We choose the KAOS method as it has a well developed mechanism to reason about the goal models, including partial goal satisfaction [11] or obstacle analysis [12]. *Obstacles* are conditions that prevent the satisfaction of a goal. Relating

BPMN with KAOS allows the use of these techniques on the goal model inferred from the process model. Other goal-oriented frameworks such as i* [7] are not yet considered because they do not share neither the same terminology nor the theoretical background.

3 Relating Business Process Models and Goal-Oriented Models

This section presents a meta-model that integrates both BPMN 2.0 and KAOS meta-models without changing their individual meaning. It also presents a precise definition of the *Intentional Fragment*.

3.1 The Meta-model Relating BPMN 2.0 and KAOS

This section presents the meta-model in Fig. 3 which introduces the notion of *Intentional Fragment* as a mean to relate the BPMN 2.0 meta-model to the KAOS meta-model. On top of the figure, the KAOS constructs are represented. In the bottom, we represent the BPMN 2.0 constructs that we are interested on. A *Goal* in KAOS could be refined into sub-goals as well as *Agents* could also be refined. An *Agent* MAY be responsible of several *Goals* (0..*). Only the leaf-goals in the goal model are related (not necessarily) to an *Agent*.

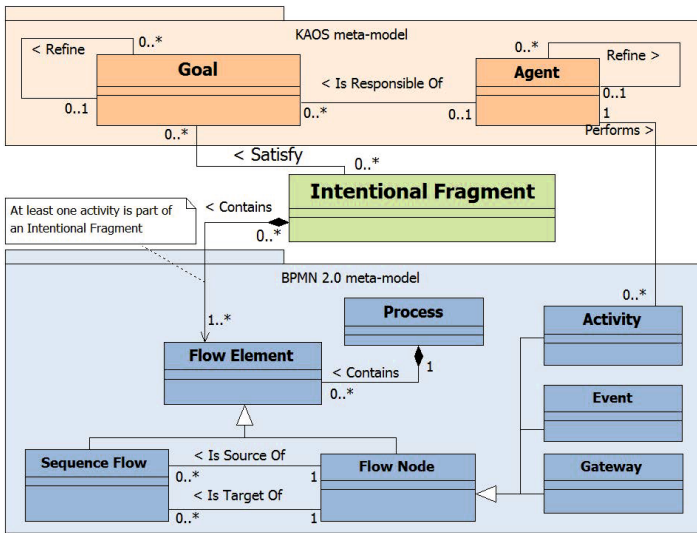


Fig. 3. Meta-model relating BPMN 2.0 and KAOS through Intentional Fragment

In BPMN 2.0 a *Process* contains a set of *Flow Elements* that could be *Flow Nodes* (i.e *Activity*, *Gateway* and *Event*) or *Sequence Flow* (an arrow which defines the sequence of the *Flow Nodes*). An *Intentional Fragment* contains one

ore more *Flow Elements* (at least one activity). Several nodes in the process could be related to an *Intentional Fragment*(0..*). An *Intentional Fragment* therefore could be seen as a set nodes in a process (not necessarily connected) that MAY satisfy a *Goal*. A *Goal* MAY be satisfied by several *Intentional Fragments*(0..*). Each of these *Intentional Fragments* represent an alternative way to satisfy the *Goal*. An *Agent* is considered responsible of a *Goal* if it performs all the *Activities* related to a *Goal* through an *Intentional Fragment*.

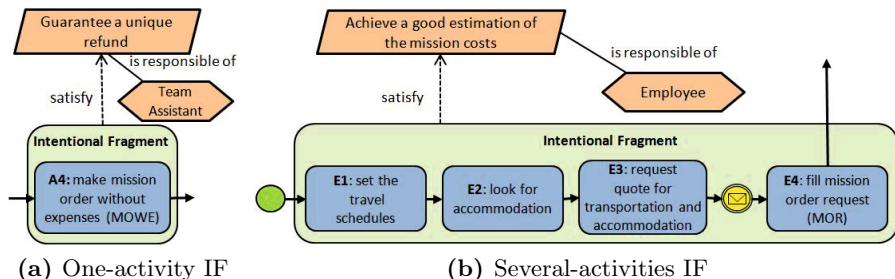


Fig. 4. Examples of Intentional Fragments (IF) that Satisfy a Goal

Based on the mission process, Fig. 4 illustrates two examples of intentional fragments linked with their corresponding goals. In the first example, the intentional fragment contains one activity (“A4: make mission order without expenses (MOWE)”). This activity is performed by the *Team Assistant* when the financier sponsor of the mission is not the employer. The MOWE is the document that is needed to control that the *Employee* will not receive an extra refund because the expenses are in charge of an external financier. Therefore, the intentional fragment in Fig. 4a satisfies the goal “Guarantee a unique refund”. The second intentional fragment satisfies the goal “Achieve a good estimation of the mission costs”. All the activities within an intentional fragment are performed by the *Employee*. This implies that the latter is responsible to fulfill the goal. A more detailed definition of the *Intentional Fragment* is presented in the following section.

We rely on the work of Anaya et al. [13] to use the *KAOS Agent* term to refer to the participant BPMN 2.0 concept. In this work authors present the Unified Enterprise Modelling Language approach (UEML) as a mean of mapping different languages to a common ontology to interrelate construct descriptions at the semantic level (BPMN and KAOS among others). The work shows that an *Agent* in KAOS and a *Participant* in BPMN are both constructs used to show an active entity that do not refer neither to transformations nor states within the overall system. In addition, they could both refer to an abstract entity or a concrete instance. In our work, an *Agent* will also refer to *Lanes* if the latter represent roles in the organization like in our example. One of the advantages of using BPMN and KAOS is that both frameworks allow different levels of granularity: high level goals and sub-processes (compound activities) can be refined into more specific goals and activities, respectively.

Our approach may be developed using resource assignment models [14] to relate agents with performed activities.

3.2 The Intentional Fragment

The *Intentional Fragment* makes explicit a relation between its constituent activities and its corresponding goal that otherwise would not be visible. The identification of intentional fragments is guided by the underlying purpose of the activities. The scope of our discussion is a single process model although the concept of intentional fragment can be expanded in future work to include activities from several process models.

We define a state for an intentional fragment depending on whether or not it satisfies a goal. If the intentional fragment is related to at least one goal, the state of this class will be *Justified* (IFJ). On the other hand if an intentional fragment is not related to any goal, the state is defined as *Potential* (IFP). Following, the definition of this concept is given starting from the definition of a BPMN process:

A BPMN Process $\mathbf{P} = (N, Start, End, \delta)$ is defined by:

- A set N of flow nodes partitioned into *Activities*, *Events* and *Gateways*
- A set of start nodes $Start \subset Events$
- A set of end nodes $End \subset Events$
- A sequence relation $\delta \subset N \times N$ defined as a set of tuples of nodes from the process \mathbf{P} satisfying a set of well-formedness constraints defined in the BPMN specification
 - $e_{start} \in Start$ has no predecessor in δ
 - $e_{end} \in End$ has no successor in δ
 - $act \in Activity$ has exactly one successor and one predecessor in δ
 - $gtw \in Gateway$ has either one predecessor and several successors or several predecessors and one successor

An *Intentional Fragment [Potential]* **IFP** of a BPMN process $\mathbf{P} = (N, Start, End, \delta)$ is a tuple $(N', Start', End', \delta')$ such that:

- $N' \subseteq N$
- $Start' \subseteq N'$ is the set of nodes that have no predecessors in **IFP**
- $End' \subseteq N'$ is the set of nodes that have no successors in **IFP**
- A sequence relation $\delta' \subset N' \times N'$ which is the smallest relation satisfying the following criteria:
 - if n_2 is reachable from n_1 in \mathbf{P} , then n_2 should be reachable from n_1 in **IFP**. We verify whether n_2 is reachable from n_1 using the transitive closure of the sequence flow relation [15] $(\forall n_1, n_2 \in N') n_2 \in n_1 \cdot * \delta \Rightarrow n_2 \in n_1 \cdot * \delta'$.
 - if n_2 is not reachable from n_1 in \mathbf{P} , then n_2 should not be reachable from n_1 in **IFP**: $(\forall n_1, n_2 \in N') n_2 \notin n_1 \cdot * \delta \Rightarrow n_2 \notin n_1 \cdot * \delta'$.

IFJ is an *Intentional Fragment [Justified]* in process $\mathbf{P} = (N, e_{start}, End, \delta)$ for goal \mathbf{G} iff:

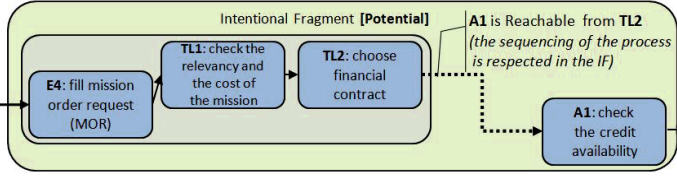
- **Inclusion** criteria: **IFJ** is an Intentional Fragment [Potential] of **P**, as defined above
- **Completeness** criteria: the execution semantic of **IFJ** should be enough to entail goal satisfaction: $[[\mathbf{IFJ}]] \models \mathbf{G}$
- **Minimality** criteria: there is no other fragment **IFJ'** such that $[[\mathbf{IFJ}']] \models \mathbf{G}$ and $\mathbf{IFJ}' \subset \mathbf{IFJ}$

The definition of intentional fragment that is introduced imposes only the minimum set of constraints necessary to preserve the semantics of the process model. As long as these constraints are respected, analysts have the liberty to decide what constitutes an intentional fragment. However, in future work we will introduce additional well-formedness constraints to support formal verification of the completeness and minimality criteria.

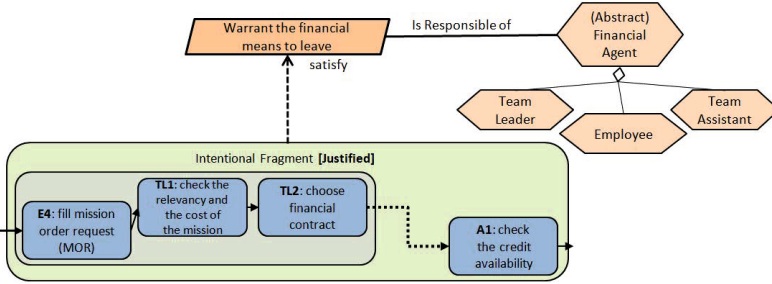
As a precondition to formally verify completeness and minimality criteria, we need to consider how execution semantics for the process are expressed. In the BPMN specification, the execution semantics is presented in a textual form, using the concept of token. A mapping to the Business Process Execution Language for Web Services (WS-BPEL) [16] is also presented, which gives an execution semantics. BPMN semantics have also been expressed using Petri Nets [17] or Calculus of Orchestration of Web Services (COWS) [18]. Although these approaches use event based semantics, state based semantics have also been explored [19].

In future work, we intend to develop automated support for verifying completeness and minimality criteria for IFJ using model checking. This requires expressing the semantics of an intentional fragment in terms of Labelled Transition Systems [20] that can be modelled checked against KAOS goal models [21].

Figure 5a shows an example of intentional fragment in a *Potential* state and we also indicate that the sequencing of disconnected elements follows the process directives (transitive closure property). Figure 5b shows how the *Potential* Intentional Fragment changes its state to *Justified* when it is related to a goal. Every agent that performs at least one of the activities that are part of the fragment is partially responsible for the goal. To represent this joint responsibility, we can introduce an abstract agent that represents the aggregation of all the agents involved. In the example on Fig. 5a the team Leader, the employee and the team Assistant perform activities that are part of the intentional fragment that satisfies the goal. Therefore, the three participants refined an abstract Agent that named (*Abstract*) *Financial Agent*.



(a) Potential Intentional Fragment (IFP). Not Linked to a Goal



(b) Justified Intentional Fragment (IFJ). Linked to a Goal

Fig. 5. Examples of the Different States of an Intentional Fragment

4 Case Study

This section presents the evaluation of the *Intentional Fragment* concept. This experiment was driven by Nadine Mandrin from the PIMLIG¹ team. Firstly, the evaluation protocol is described. Then, some interesting results are discussed.

4.1 The Evaluation Protocol

Table 1 describes the evaluation protocol that we set up for our case study. We wanted to validate the hypothesis that “*An intentional fragment is an intuitive and useful concept to relate business process models and goal models*”. The available material was the case study that describes the mission process presented in Section 2. Figure 1 shows one of the two parts of the process model that were used in the evaluation. An evaluation keeps the trace of how subjects were using the notion of intentional fragment.

To carry out the evaluation we choose a method recommended by sociology and also by computer designers : the semi-structured interviews [22]. This method belongs to the family of qualitative methods such as participant observation [23] or the focus groups [24]. We choose this interview method as it helps developing a lot of ideas, opinions, or habits, even if they are not frequent within the studied subjects. The goal is not to quantify these behaviors or needs but to make a list as large as possible. Sociology recommends a minimum of 20 interviews in order to reach saturation in this list. Experience shows that

¹ <http://www.liglab.fr/pimlig>

Table 1. Evaluation Protocol for the Case Study

Subjects
People involved in the mission process of LIG (employees, team assistant, team leader and direction).
Organization
21 personal semi-structured interviews. Around 45 minutes each one.
Evaluation method
The subjects understand, use and evaluate the concept of intentional fragment in a familiar process.
Protocol
<ul style="list-style-type: none"> • The evaluator explains the mission process (<i>10min</i>). • The evaluator proposes 3 intentional fragments and the subject identifies the associated goal for each one (<i>5 - 10min</i>). • The evaluator proposes 3 goals and the subject identifies the associated intentional fragment for each one (<i>5 - 10min</i>). • The evaluator proposes a free time to identify new goals or intentional fragments. The evaluator captures the trace of the subject's work. The evaluator asks for errors or gaps in the process (<i>15min</i>). • The subject comments the usefulness of the concept of intentional fragment (<i>5min</i>).

from 20 interviews there is a behavior redundancy and new and original ideas are rare. These interviews are organized face to face with an interview schedule grid. We conducted *21 qualitative interviews* with people that takes part of the process and whose demographic profiles were different in terms of gender, age and modelling experience.

When the process model was inferred, some errors and gaps were detected. We did not correct them so we could also evaluate if our approach could help stakeholders to analyze and detect problems in both the process and the model. We did not provide the complete definition of intentional fragment to subjects to avoid extra complexity. We just define it as “*a set of elements in the process related to a goal*”. After the interviews, we analyzed all the responses to the different exercises.

4.2 First Exercise: Identifying Goals from Intentional Fragments

Three main strategies were observed when during the first exercise. Subjects tended to rely on **abstraction** to identify goals. They grouped sequential activities in a more generic one, which they considered the goal. For example, for the first fragment (Fig. 4b), a common goal proposed was “prepare the mission”. Another common strategy is that subjects were guided by the **data objects** that go out from the last activity of the proposed fragment (e.g., “get a mission request order” for the first fragment). Finally, what we consider being the best approach,

11 subjects **synthesized** the elements within the intentional fragment and go beyond abstraction. A good example of goal for the first intentional fragment would be: “achieve a good estimation of the mission costs”.

4.3 Second Exercise: Identifying Intentional Fragments from Goals

The second exercise consisted on identifying the intentional fragment corresponding to a given goal. Two specific goals (the first and the third one) and a more general goal (the second one) were proposed: 1) “the benefit against the cost of a mission has to be evaluated”, 2) “a financial settlement is established in line with the mission costs and regulated expenses” and 3) “the final financial settlement is approved by the employee”. We analyzed the variability of the suggestions for the corresponding intentional fragment.

The identification of an intentional fragment that relates to the second goal (the more general) was difficult and depended on the subject’s interpretation. We also observe much more dispersion in the activities involved. More than 50% of the subjects suggested disconnected activities to fulfill the goal.

4.4 Third Exercise: Identifying New Intentional Fragments and New Goals

The last exercise was the free part of the evaluation. Subjects had to reproduce what they had already done in the two first exercises with their own manner no matter what order. We also asked them during this exercise to identify errors or gaps in the process model.

A total of *43 different goals were inferred* in this exercise. The most common goals corresponded to the activities of the individual agents and also most of them related to passing documents. Subjects commonly distinguish the employee from the rest of the laboratory agents. The more repeated goal was: “The employee wants to do the mission with all the documents in order and then be refund”.

Goals that had no corresponding intentional fragment were source of an error or gap. A total of *49 different errors and gaps were detected*. The most common ones were: “what happens if disapprovals?” or “there is a lack of notifications”. This proves that that an exhaustive analysis could be done by means of intentional fragments.

4.5 Conclusions about the Evaluation

Some limits in our evaluation have to be considered. Firstly, the fact of using a qualitative approach do not permit to generalize the results. However, relevant feedback is given that permits refining our proposal. Secondly, the fact that all the participants were familiar with the process. In this work, we make the assumption that at least a part of the process model is already known. This helped them to infer goals that were not explicitly described in the process model.

On the other hand, we chose this approach to minimize the time dedicated to the explanation of the process and focus on the goal inference through intentional fragments. Another drawback that could be argued is that the ordering of the exercises may affect the outcome. We maintain this strategy because it helped to understand the approach and the notion of intentional fragment by giving them as first exercise some examples. We prioritize the analysis of subjects' behaviour facing the same structure in the evaluation.

The purpose of this evaluation was to evaluate the usage of the notion of *Intentional Fragment* to bridge the gap between goals and process models. We observed that even if the definition of intentional fragment was not completely given to the subjects, it appeared in a natural way. We also observed that this approach helped inferring goals.

5 Applications of the Intentional Fragment Concept

The relation between the process model and the goal model helps analysts to firstly justify the business process and then analyze the process as-is. We develop this points in the following section.

5.1 Inferring a Goal Model that Justifies the Process Model

We previously put forward the difficulty for an analyst to generate a correct goal model. Particularly challenging is the inference of goals. During the evaluation, a set of goals corresponding to the mission process were identified using the notion of intentional fragment. Figure 6 illustrates a goal model that we generate based on the goals elicited during the interviews. In structuring the goal model, we employ some standard goal refinement patterns from KAOS [10]. For example, the milestone driven refinement pattern is used to refine the goal "Plan a Mission". The pattern is applicable to goals where an intermediate condition has to hold true before reaching the prescribed condition. In our example, to be able to warrant the financial means for the mission, an estimation of the costs has to exist – this can be seen as a milestone in planning the trip.

Moreover, the different strategies that subjects used to group process nodes gave us some clues about what are the potential intentional fragments that could be automatically proposed and might be related to a business goal. We propose a set of heuristics (H) to identify potential intentional fragments of different sizes and scopes, based on visible patterns in the process model. The next step will be to establish a correspondence between these potential intentional fragments and goals. This can be done through GORE elicitation techniques [10] (e.g. asking *why?* questions). A potential intentional fragment might be composed by:

- H1. *nodes between a start event and the first sequence flow to another lane.* An example of application for H1 can be found in Fig 4b.
- H2. *nodes between a sequence flow to another lane and an end event.*

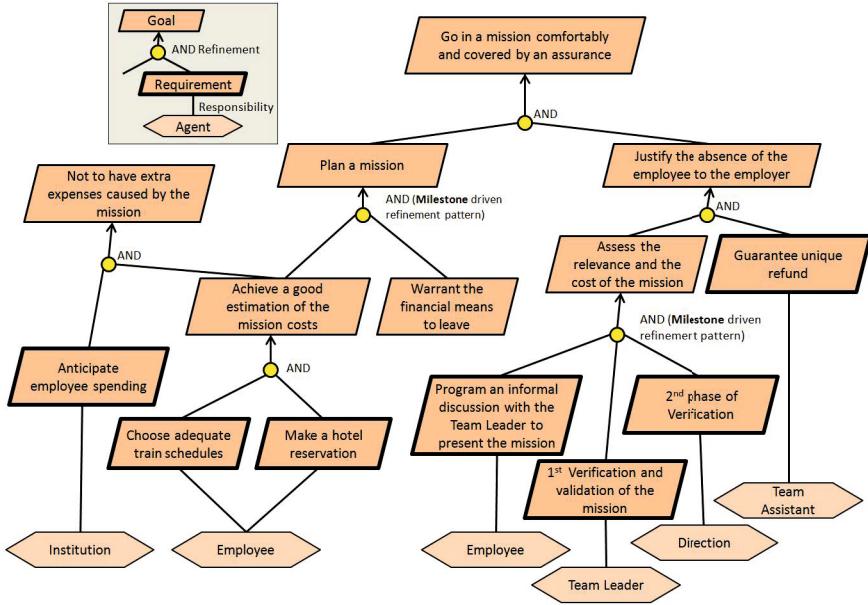


Fig. 6. A Resulting Goal Model Inferred from Goals Proposed by Subjects

- H3. *nodes within the end events (other than the happy path end event) and the immediately preceding XOR gateway.* “Happy path” stands for the path where everything goes right. Figure 7 shows the application of this heuristic. In this case, the intentional fragment is not linked to any goal (subjects did not find a goal associated).
- H4. *nodes immediately after and before a XOR gateway.* They could imply control goals.
- H5. *nodes within the path of two consecutive sequence flows that transit from lane to lane.* Figure 7 shows the application of this heuristic where the intentional fragment satisfies the goal “1st verification and validation of the mission”.
- H6. *nodes between two consecutive messages to/from the same agent.*
- H7. *nodes having as input the same documents.*
- H8. *nodes labeled with similar verbs.* For example verify, inform, sign, etc ...
- H9. *nodes within a lane.*
- H10. *nodes within the “happy path”.* They relate to a high level goal.

Although difficult to infer, the goal model is not an end in itself. Rather, it is useful because it supports further reasoning about the goals, for example through conflict or obstacle analysis. To validate the usefulness of the inferred goal model, we present one result that emerges from the obstacle analysis. Considering the goal “Achieve a good estimation of the mission costs”, we can negate it and infer an obstacle. So, starting from the statement “Do not achieve a good estimation of the mission costs”, a business analyst can ask what happens if some costs are

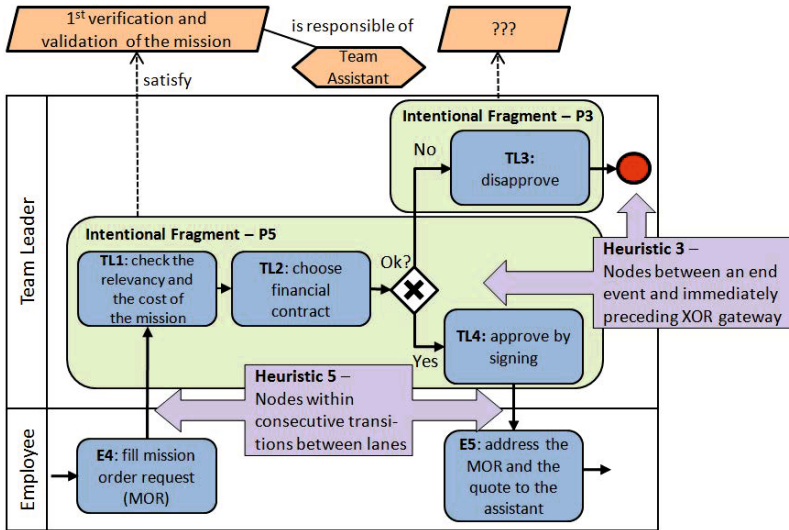


Fig. 7. Example of Application of Heuristics 3 and 5

not considered, or if estimations cannot be obtained, or estimations are much less than in reality. All of these are concerns that need to be addressed. As such, the utility of the intentional fragment is validated by the fact that, by analyzing the goal model inferred, we can identify problematic situations.

5.2 Alignment between the Process Model and the Goal Model

If both models are available and relations between intentional fragments and goals have been established, an alignment analysis can be performed. We present a set of heuristics that can be used to identify miss-alignments between the two models. Figure 8 gives an overview of the alignment between a part of the process model and a part of the goal model. It also illustrates the how we could apply the two last heuristics.

- *Check that for each goal identified in the goal model, there exists at least one corresponding intentional fragment* - this identifies objectives not fulfilled by the considered process or shows that additional process models should be considered. When there is any activity contributing to the satisfaction of goal, a critical problem is detected. An example in our use case regarding Fig. 2 is the goal “Program an informal discussion with the Team Leader to present the mission”. There is no activity that clearly contributes to the satisfaction of this goal in the process model.
- *Check that each activity in the process model is part of at least one intentional fragment* - this step identifies superfluous activities. If an activity is not part of any intentional fragment, an organizational problem is detected. In our case study, the activity “TL3: Disapprove” (see Fig. 1) cannot be related

to any goal . Although there is probably a reason to disapprove the mission order request, the goal was not identified.

- *Check the Agents' responsibility* - for each goal, we may verify if we could assign all the activities that are part of the intentional fragment to an agent. Figure 8 shows that to warrant the financial means of the journey, three participants are involved that are the the team leader, the employee and the team assistant. Looking at the process we observe that there is a double verification of the warrant of financial means by the team leader and the team assistant. We could think about the possibility to delegate this verification to only one agent. Consequently, the performance of the process may be improved.
- *Detecting interlocking intentional fragments* - An interesting question that arises is: If two intentional fragments are intertwined, how do they influence each other? If the goals that relates the intentional fragments do not depend on each other, the possibility of performing the activities in parallel may be considered. In Fig. 8, the travel and hotel planning (i.e. activities E1, E2) may be done in parallel because each activity is related to non-dependent goals.

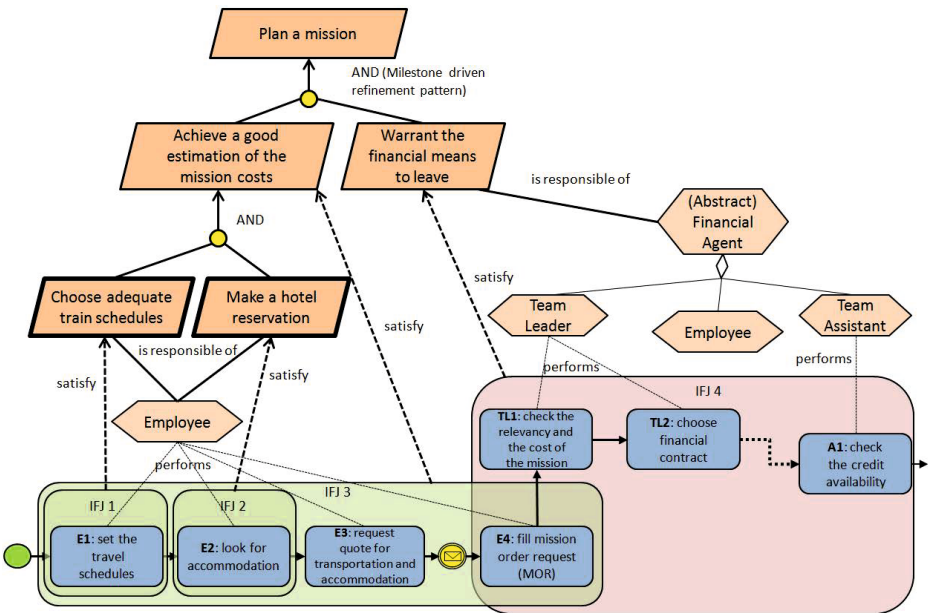


Fig. 8. Alignment between Goal Model and the Process Model

6 Related Work

Other researches have already presented some relation between goal models and business process models. Extending BPMN models to include additional artifacts that allow traceability to goals have been proposed [6,9,25]. However, these

approaches add more complexity to the language. Alternatively, goal models are firstly mapped to process models and then enriched to support variability on the process models [7]. The problem here is that the goal model ends representing almost the same view that the process model rather than being a higher level intentional layer. This approach does not focus on reasoning about the rationale of activities and their purpose as part of the overall process. Hence, its scope is limited. In our approach one of our major aims is to maintain a clear separation of concerns between process models (the organizational layer) and the goal model (intentional layer): the *the Intentional Fragment* is a pivot to bridge the gap between both layers.

The notion of fragment has already been introduced in addition to standard BPMN 2.0 constructs. For example, using reusable fragments that are then mapped to BPEL blocks [26]. Fragments are identified based on certain structural patterns visible in the process model. Fragments have also been used to represent localized knowledge regarding the business process [27]. This approach starts from the assumption that each participant is aware of only some parts of the process, and this constitutes a fragment. These fragments need to be integrated to obtain a complete process model. Fragments are also used as a mean to propose change patterns in process models [28]. These approaches use the fragment concept as a connected set of nodes. In our approach, we also support disconnected parts of the process and in addition, we explicitly define a relation between fragments and goals.

In [29] authors present an aspect-oriented approach to modularize the cross-cutting concerns in a business process modeling and they applied it using BPMN. However, they stay in an organizational level when they identify these concerns. In our approach we consider an intentional level represented by the goal model.

Automatic goal decomposition is performed to support cooperative team formation (process parts assignments) within the context of Instant Virtual Enterprise (IVE) [30]. This work describes how to create the IVE process dynamically matching goal requirements and agents capabilities. Our approach is more focused on facilitating the extraction of a goal model from a process-as-is and all the further analysis that may be applied to identify problems or improve the process model.

7 Conclusion and Future Work

In this paper we introduce the notion of intentional fragment as a mean to relate process models and goal models. As we have shown in the case study, the approach is a simple and very pragmatic solution for a well-known problem. Business analysts as well as stakeholders which are part of the process could be involved in the generation of the goal model from a process model. Goal based analysis could latter be performed and will help gaining a more structured understanding of the process. The critical analysis is supported by delimiting the organizational and intentional level. The BPMN 2.0 model and the KAOS goal model represent different perspectives over the system. These

models complement each other and are used to identify new goals or possible organizational problems in the existing process. Intentional fragments permit establishing the relations between goals and process nodes at different levels of abstraction. These nodes could not be necessarily connected. Both high level goals as well as requirements (i.e. leaf-goals) could be related with one or more intentional fragments. The heuristics to extract potential intentional fragments facilitates goal elicitation and goal-based analysis of the process model.

The work presented here raises several interesting questions for the future. Most importantly, the identified heuristics could be the start-point to automate intentional fragment identification. These heuristics will be integrated into a tool that can help business analysts infer goal models from the business process model. This would allow generating better goal models with less effort and therefore help to justify them. Secondly, the semantics of intentional fragments will be formalized. We also acknowledge the potential to suggest ameliorations in the process, based on the results of the traceability analysis.

References

1. Hepp, M., Roman, D.: An ontology framework for semantic business process management. In: *Proceedings of Wirtschaftsinformatik 2007* (2007)
2. Indulska, M., Recker, J., Rosemann, M., Green, P.: Business Process Modeling: Current Issues and Future Challenges. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) *CAiSE 2009*. LNCS, vol. 5565, pp. 501–514. Springer, Heidelberg (2009)
3. de la Vara, J.L., Sánchez, J., Pastor, Ó.: Business Process Modelling and Purpose Analysis for Requirements Analysis of Information Systems. In: Bellahsene, Z., Léonard, M. (eds.) *CAiSE 2008*. LNCS, vol. 5074, pp. 213–227. Springer, Heidelberg (2008)
4. OMG: Business process model and notation (bpmn 2.0) (2011), <http://www.omg.org/spec/BPMN/2.0/>
5. Recker, J.: Opportunities and constraints: the current struggle with bpmn. *Business Process Management Journal* 16(1), 181–201 (2010)
6. Pavlovski, C., Zou, J.: Non-functional requirements in business process modeling. In: *Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling*, vol. 79, pp. 103–112. Australian Computer Society, Inc. (2008)
7. Lapouchnian, A., Yu, Y., Mylopoulos, J.: Requirements-Driven Design and Configuration Management of Business Processes. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 246–261. Springer, Heidelberg (2007)
8. Cardoso, E., Guizzardi, R., Almeida, J.: Aligning goal analysis and business process modelling: a case study in healthcare. *International Journal of Business Process Integration and Management* 5(2), 144–158 (2011)
9. Koliadis, G., Ghose, A.K.: Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. In: Hoffmann, A., Kang, B.-H., Richards, D., Tsumoto, S. (eds.) *PKAW 2006*. LNCS (LNAI), vol. 4303, pp. 25–39. Springer, Heidelberg (2006)
10. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pp. 249–262. IEEE (2001)

11. Letier, E., Van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. *ACM SIGSOFT Software Engineering Notes* 29, 53–62 (2004)
12. Van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering* 26(10), 978–1005 (2000)
13. Anaya, V., Berio, G., Harzallah, M., Heymans, P., Matulevicius, R., Opdahl, A., Panetto, H., Verdecho, M.: The unified enterprise modelling language—overview and further work. *Computers in Industry* 61(2), 99–111 (2010)
14. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: Defining and Analysing Resource Assignments in Business Processes with RAL. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) *ICSOC 2011*. LNCS, vol. 7084, pp. 477–486. Springer, Heidelberg (2011)
15. Lidl, R., Pilz, G.: *Applied abstract algebra*. Springer (1998)
16. OASIS: Web services business process execution language v2.0 (2007), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
17. Dijkman, R., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in bpmn. *Information and Software Technology* 50(12), 1281–1294 (2008)
18. Prandi, D., Quaglia, P., Zannone, N.: Formal Analysis of BPMN Via a Translation into COWS. In: Lea, D., Zavattaro, G. (eds.) *COORDINATION 2008*. LNCS, vol. 5052, pp. 249–263. Springer, Heidelberg (2008)
19. Soffer, P., Wand, Y.: Goal-Driven Analysis of Process Model Validity. In: Persson, A., Stirna, J. (eds.) *CAiSE 2004*. LNCS, vol. 3084, pp. 521–535. Springer, Heidelberg (2004)
20. Magee, J., Kramer, J.: *State models and java programs*. Wiley (1999)
21. Letier, E., Kramer, J., Magee, J., Uchitel, S.: Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering* 15(2), 175–206 (2008)
22. Hindus, D., Mainwaring, S., Leduc, N., Hagström, A., Bayley, O.: Casablanca: designing social communication devices for the home. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 325–332. ACM (2001)
23. Simiand, F.: *Méthode historique et science sociale*. *Annales. Histoire, Sciences Sociales* 15, 83–119 (1960)
24. Bruseberg, A., McDonagh-Philp, D.: Focus groups to support the industrial/product designer: a review based on current literature and designers’ feedback. *Applied Ergonomics* 33, 27–38 (2002)
25. Morrison, E., Ghose, A., Dam, H., Hinge, K., Hoesch-Klohe, K.: *Strategic alignment of business processes* (2011)
26. Ouyang, C., Dumas, M., Ter Hofstede, A., Van Der Aalst, W.: Pattern-based translation of bpmn process models to bpel web services. *International Journal of Web Services Research (JWSR)* 5(1), 42–62 (2007)
27. Eberle, H., Leymann, F., Schleicher, D., Schumm, D., Unger, T.: Process fragment composition operations. In: *2010 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pp. 157–163. IEEE (2010)
28. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features—enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering* 66(3), 438–466 (2008)
29. Cappelli, C., Leite, J., Batista, T., Silva, L.: An aspect-oriented approach to business process modeling. In: *Proceedings of the 15th Workshop on Early Aspects*, pp. 7–12. ACM (2009)
30. Mehandjiev, N., Grefen, P.: *Dynamic business process formation for instant virtual enterprises*. Springer-Verlag New York Inc. (2010)

Indexing Process Model Flow Dependencies for Similarity Search*

Ahmed Gater¹, Daniela Grigori², and Mokrane Bouzeghoub¹

¹ Université de Versailles Saint-Quentin en Yvelines

45 avenue des Etats-Unis, 78035 Versailles Cedex, France

² Université Paris-Dauphine, Pl. M^{al} de Lattre de Tassigny 75775 Paris, France

Abstract. The importance gained by process models in modern information systems led to the proliferation of process model repositories. Retrieving process models within such repositories is a critical functionality. Recent works propose metrics that rank process models of a repository according to their similarity to a given query. However, these methods sequentially browse all the processes of the repository and compare each one against the query, which is computationally expensive. This paper presents a technique for quickly retrieving process models similar to a given query that relies on an index built on behavioral characteristics of process models.

Keywords: semantic process models, process similarity search, process indexing.

1 Introduction

The importance gained by process models in modern information systems and in service oriented architecture led to the proliferation of process model repositories. These repositories may store collections of hundreds of process models used by large enterprises, best practices processes (like SAP best practice processes¹) or reference models provided by process management systems vendors.

Consequently, there is a critical need for tools and techniques to manage process model repositories, including techniques that allow retrieving process models fulfilling user needs. If the user need is formulated or available as a process model, the most similar processes must be retrieved in the repository. Solving this problem, called process similarity search, requires to (i) define a suitable similarity measure and (ii) propose methods that evaluate the similarity between a process query and a set of target processes in the repository. While the first problem received recently considerable attention ([10,3]), very few approaches addressed the second one.

Given an algorithm calculating a similarity measure for two processes, a naive approach to solve the retrieval problem is to traverse all the processes of the

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

¹ <http://www.sap.com/solutions/businessmaps/composer/index.aspx>

repository and compare each one against the query, and rank these processes according to their similarity to the query. However, majority of existing process matching algorithms ([11,5]) are NP-complete and therefore they do not scale for large process model repositories.

We propose in this paper an effective and fast similarity search technique that allows retrieving the most similar processes to a user query within a process repository. To this end, we use an abstraction function that represents a process as a finite set of *flow dependencies* between its activities. Thus, the similarity of two processes is defined at the basis of the similarity of their *flow dependencies*. To speed up the comparison of the flow dependencies of the query and those of the repository processes, we define an index structure built on the *flow dependencies* and the activities of the processes. Furthermore, we address the case where a process query cannot be fulfilled by a single target process, but by the composition of several processes. To the best of our knowledge there is not other work allowing to propose the composition of a set of processes as an answer of a query.

The remainder of the paper is organized as follows. The next section presents basic definitions and notations. Section 3 presents the abstraction function we used to represent processes. Section 4 explains the index structures and section 5 shows how they are used for query answering. In section 6 we present an experimental study of our technique. Section 7 discusses related works. Finally section 8 draws conclusions and presents ongoing work.

2 Background and Definitions

A business process model consists of a set of related activities that are combined using control flow operators. In this paper, a process model is formalized as a directed attributed graph (A, C, E) , called process graph (p-graph for short), where A is a set of activity nodes, C is a set of connector nodes and E is a set of edges. An activity node represents an atomic task, while connectors represent control flow constraints between activities. An activity $Act = (N, In, Out)$ is described by its name (N), a set of inputs (In), and a set of outputs (Out). The inputs and outputs of activities are annotated with concepts taken from a domain ontology. Connector nodes represent *Split* and *Join* operators of types *XOR* or *AND*. *Split* connectors have multiple outgoing edges, while *Join* connectors have multiple incoming edges.

The processes we handle are block-structured, i.e. sequences, alternative and parallel branchings, and loops are specified with well defined *entry* and *exit* nodes. A block in a process can be an atomic activity, a well-delimited sub-process, or even the process itself. There are five types of blocks: atomic block (a single atomic activity), sequence of blocks $SEQ < B_1, \dots, B_n >$, parallel execution of blocks $AND < B_1, \dots, B_n >$, alternative execution of blocks $XOR < B_1, \dots, B_n >$, and loop through a block $LOOP < B >$. The blocks may be nested, but never overlap, i.e. two blocks are either nested or disjoint (if a node belongs to two blocks then they are nested).

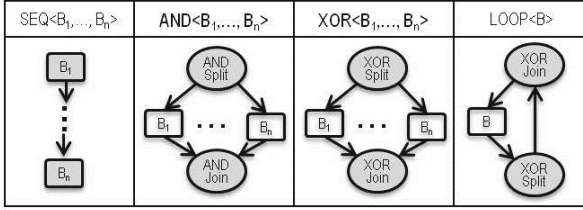


Fig. 1. Block structures

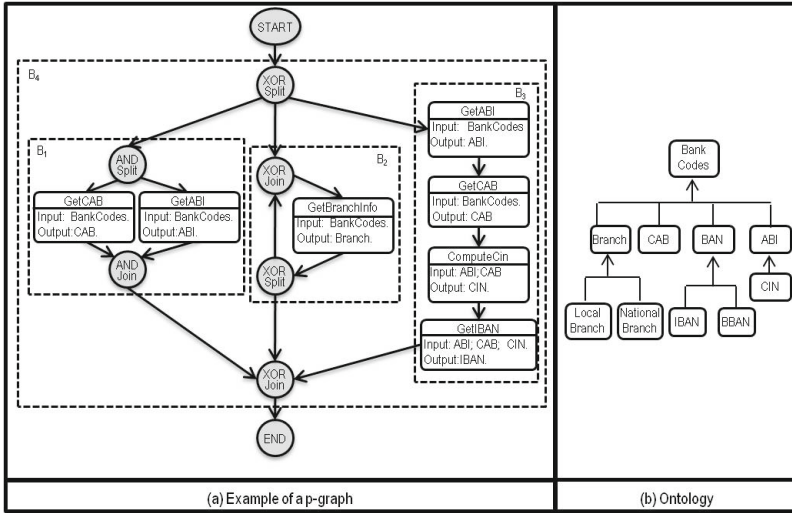


Fig. 2. Running Example

The structures of these blocks are shown by Figure 1, where the highlighted nodes are the *entry* and *exit* of each block. Square boxes represent activity nodes, and the oval ones represent connectors. An example of such block-structured processes is depicted in the left part of Figure 2, where its blocks are shown in the form of dashed boxes (B_1 , B_2 , B_3 and B_4). The inputs/outputs of its activities are annotated by the ontology depicted in the right part of the same figure. Notice that the assumption that processes are structured into blocks is not strong, since recent studies have shown that most of the unstructured processes can be transformed to equivalent structured processes [14].

In the remainder, we use the notions of smallest block containing a set of activities and loop-free path. Their descriptions are given in Definitions 1 and 2.

Definition 1. The smallest block containing a set of activities

Let (A, C, E) be a p -graph and $acts = \{a_1, \dots, a_k\} \subseteq A$ a set of activities. The smallest block containing the activities of $acts$, denoted $\delta(acts)$, is the block B such that:

- B contains the activities of acts, and
- Every block $B_i \neq B$ which contains the activities of acts contains also B .

Definition 2. Path and loop-free path. Let (A, C, E) be a p -graph and a and b be two nodes. A path $a \rightarrow b$ refers to the existence of a sequence of edges $(n_1, n_2), (n_2, n_3), \dots, (n_{k-1}, n_k) \in E$, with $k > 1$, $n_1 = a$ and $n_k = b$. A path that does not contain an edge (n_{i-1}, n_i) , with n_{i-1} is a connector node of type XOR-Split and n_i is a connector node of type XOR-Join, denoted $a \xrightarrow{lf} b$, is called loop-free path.

3 Process Model Representation for Fast Retrieval

While graph indexing algorithms exist in the literature [15], these algorithms can not be directly applied to process graphs. Firstly, p -graphs have two kind of nodes (activities and control nodes) and specific attributes capturing the semantics of the process. Most importantly, p -graphs capture the behavioral semantics of the processes, which is not taken into account by the existing graph indexing techniques that are mainly structural.

The idea is then to transform the p -graphs into another representation which is the most faithfully representative of the p -graphs and, at the same time, makes the evaluation of their similarity faster. This representation must be enough representative to ensure that the similarity of two p -graphs is strongly correlated by the similarity of their new representation.

To this end, we use an abstraction function that captures the essential behavioral characteristics specified by a p -graph. This abstraction function inspired by [6] and called *process type*, represents a p -graph as a finite set of *flow dependencies* that occur between each pair of its activities. A *flow dependency* type specifies how two activities relate to each other, and it is defined as the type of the smallest block containing the two activities as formalized by Definition 3. This definition states that there are four types of *flow dependencies* that may occur between a pair of activities: “SEQ” when one of them is always executed after the end of the execution of the other one, “PATH” if there exists a loop-free execution path between them, “AND” when they are executed in parallel, “XOR” when the activities are never executed in the same run.

Definition 3. Flow dependency type. Let (A, C, E) be a p -graph and $a_i, a_j \in A$ two activities. The type $T_{i,j}$ of the flow dependency that may occur between a_i and a_j is one of the followings:

- **Sequence flow dependency:** $T_{i,j} = SEQ$ iff $\delta(\{a_i, a_j\})$ is of type SEQ and $(a_i, a_j) \in E$. The SEQ flow dependency is not commutative, thus, if $T_{i,j} = SEQ$, then $T_{j,i}$ is not defined.
- **Path flow dependency:** $T_{i,j} = PATH$ iff $\delta(\{a_i, a_j\})$ is of type SEQ and $a_i \xrightarrow{lf} a_j$ occurs. The PATH flow dependency is not commutative, thus, if $T_{i,j} = PATH$, then $T_{j,i}$ is not defined.

- **XOR flow dependency:** $T_{i,j} = XOR$ iff $\delta(\{a_i, a_j\})$ is of type XOR. The XOR flow dependency is commutative, thus, if $T_{i,j} = XOR$, then $T_{j,i} = XOR$.
- **AND flow dependency:** $T_{i,j} = AND$ iff $\delta(\{a_i, a_j\})$ is of type AND. The AND flow dependency is commutative, thus, if $T_{i,j} = AND$, then $T_{j,i} = AND$.

The type of the *flow dependency* of two activities is unique since it is defined on the basis of the type of the smallest block containing the two activities.

In order to take into account loops specified in a p-graph, we define the notion of *flow dependency multiplicity*, emphasizing the fact that some flow dependencies involve activities situated in a loop (and thus possible executed several times in a run).

Definition 4. Flow dependency multiplicity. Let (A, C, E) be a p-graph, $a_i, a_j \in A$ be two activities, $T_{i,j}$ be the type of their flow dependency. The multiplicity of a flow dependency between a_i, a_j , denoted $M_{i,j}$ is:

- $M_{i,j} = *$ iff one of the blocks containing the smallest block containing a_i and a_j ($\delta(\{a_i, a_j\})$) is of type LOOP.
- $M_{i,j} = 1$ otherwise.

The *process type* of a p-graph is defined below as the set of the pairs of activities, with their *flow dependencies types* and multiplicities. Notice that for each pair of activities only one *flow dependency* is added to the *process type*. When the *flow dependency type* between two activities is commutative (XOR and AND), the pair of activities to be added is, by convention, (a_i, a_j) such that the name of a_i is lexicographically prior to the name of a_j .

Definition 5. Flow dependency and Process type. Let $P = (A, C, E)$ be a p-graph and $a_i, a_j \in A$ be two activities.

- The flow dependency between a_i and a_j is defined as a tuple $fd_{i,j} = (a_i, a_j, T_{i,j}, M_{i,j})$, where $T_{i,j}$ and $M_{i,j}$ are respectively its type and multiplicity.
- The process type PT_P of P is the set of all flow dependencies that occur between the pairs of activities of P , such that for any pair of flow dependencies $(a_i, a_j, T_{i,j}, M_{i,j})$ and $(a_k, a_l, T_{k,l}, M_{k,l}) \in PT_P$, $a_i \neq a_k \vee a_j \neq a_l$, $a_i \neq a_l \vee a_j \neq a_k$.

In the following we use $fd(a_i, a_j)$, $fd(a_i, a_j).Type$, and $fd(a_i, a_j).Multiplicity$ to denote, respectively, the *flow dependency* occurring between activities a_i and a_j , its type, and its multiplicity.

4 Indexing Process Models

Two p-graphs are considered as likely similar when they have similar *process types*, i.e. the more two p-graphs share *flow dependencies*, the more they are

similar. The goal is then to speed up the comparison of the *process type* of the query with those of the p-graphs registered in the repository.

To identify p-graphs having similar process types to a query, we first need to identify among registered activities those that are similar to the query activities. Thus, we need an efficient way to find all the activities registered in the repository that are similar to the activities of the query.

Accordingly, we define two index structures, constructed on an off-line pre-processing step of the p-graphs registered in the repository that speed up the evaluation of the queries. The first structure indexes the activities stored in the repository, and is built on the concepts annotating their inputs and outputs. The second one consists on hash tables that store the *process types* of p-graphs as signatures of their *flow dependencies*. We assume hereafter that each p-graph of the repository has a unique identifier, and each activity of each p-graph has also a unique identifier. We assume also that all the p-graphs registered in the repository are annotated using the same ontology. This assumption is not strong since there are many works on ontology alignment and merging (see [7] for a survey in ontology alignment). In the following, we show how these index structures are built.

4.1 Indexing the Activities of the Repository

An important issue in our p-graph retrieval technique is its ability to retrieve among the activities of the repository those that are similar to the activities of the query. The idea is then to define efficient mechanisms that allow finding quickly potential matches of a query activity. Based on the observation stating that activities having similar inputs/outputs are considered more likely to be similar [9,13], we built an index that allows quickly retrieving, among the activities of the repository, those having inputs/outputs similar to those of the query activity.

The index consists of two sets, attached to each concept of the ontology, that record the activities where this concept appears as an input or an output. Precisely, each concept c of the ontology is attached to two sets of annotations In_c and Out_c that record the identifiers of the activities in which this concept appears, respectively, as an input or an output. For instance, let us consider the p-graph example and the piece of the ontology annotating its activities of Figure 2. The attached input and output sets of the concept “ABI” are respectively $In_{ABI} = \{ComputeCIN, GetIBAN\}$ and $Out_{ABI} = \{GetABI - 1, GetABI - 2\}$.

In this work, we are interested in retrieving inexact matches when exact ones do not exist. In other words, an activity in a target process is considered as a potential match for a activity of the query process when it satisfies the constraints (inputs and outputs) of the query activity at a given level, i.e. some mismatches between the inputs/outputs of a query activity and those of a target activity can be tolerated.

For instance, there is no activity of the p-graph depicted in Figure 2 that can strictly fulfill a query activity requiring the concept “NationalBranch” as an output. Accordingly, we have to look for activities having as output the concepts the most similar to “NationalBranch” on the basis of the relationships

between the concepts of the ontology. That way, by considering, for example, the parent concept of “*NationalBranch*” which is the concept “*Branch*”, the activity “*GetBranchInfo*” can be considered as a potential match of this query activity.

Given a concept c annotating an input (resp. output) of a target activity, the idea is to find a reduced set of concepts (to avoid overloaded answers), called *relaxers*, such that, when a query activity requires as input (resp. output) one of the *relaxers* of c , this latter can be considered as a match at a given degree of this activity input (resp. output). To get the set of relaxers of a given concept c , we use three relaxation rules that are formalized in Definition 6. These rules give the possible ways to relax a concept annotating an input or an output. The relaxation rules that have to be applied and the distance (parameter ξ in definition 6, called relaxation degree) between a concept and its relaxers are application dependent reflecting the mismatches that a user accepts for finding activity matches. Notice that the higher is the value of ξ , less accurate are the matches. Setting $\xi = 0$ means that no relaxation is allowed, and thus only exact matches are retrieved. In the remaining, we use $c \xrightarrow{\eta} c'$, where $\eta \in \{desc, asc, cous\}$ to denote that the concept c' is a relaxer of type η of the concept c . Notice that each concept is the *relaxer* of itself, and we note that by $c \xrightarrow{origin} c$.

Definition 6. Concept relaxation rules. *Let c_1 and c_2 be two concepts of the same ontology O , c_{sc} be their least common superconcept, and a natural number $\xi \geq 1$. c_1 can relax c_2 as follows:*

- **Descendant relaxer:** c_1 is a descendant relaxer of generation ξ (denoted $Desc_\xi$) of c_2 iff c_1 is a sub-concept of c_2 , and the length of the path between c_1 and c_2 (in number of intermediate edges) is less than or equal to ξ .
- **Ascendant relaxer:** c_1 is an ascendant relaxer of generation ξ (denoted Asc_ξ) of c_2 iff c_1 is a super-concept of c_2 , and the length of the path between c_1 and c_2 (in number of intermediate edges) is less than or equal to ξ .
- **Cousin relaxer:** c_1 is a cousin relaxer of generation ξ (denoted $Cous_\xi$) of c_2 iff the length of the paths between c_{sc} and c_1 and c_2 are less than or equal to ξ .

Consequently, the set of annotations attached to the input set In_c (resp. output Out_c) of a concept c records also the identifiers of the activities that have as input (resp. output) one of the concepts that it can relax.

For instance, let us consider the concept *ABI* of the ontology depicted by Figure 2 and the activities of the p-graph of the same figure. By considering the aforementioned relaxation rules and relaxers of generation 1 ($\xi = 1$), the output set attached to concept “*ABI*” contains activity identifiers: “*GetABI-1*” and “*GetABI-2*” (because “*ABI*” is an output of these activities), “*ComputeCIN*” (because this activity has concept “*CIN*” as an output which is a descendant relaxer of “*ABI*” of generation 1), “*GetCAB-1*” and “*GetCAB-2*” (because these activities has concept “*CAB*” as an output which is a cousin relaxer of “*ABI*” of generation 1). The formal description of the attached input and output sets of a concept c are given in Definition 7.

Definition 7. Input and Output annotation sets. Let $AIDs$ and $GIDs$ be, respectively, the sets of the identifiers of all the activities and the p-graphs registered in the repository. Let $\eta \in \{desc, asc, cous, origin\}$ be a relaxation rule. Let c be a concept belonging to an ontology O .

- $In_c = \{(Id_A, Id_g, \eta, ss) | Id_A \in AIDs, Id_g \in GIDs, \exists c' \in In(Id_A), c' \xrightarrow{\eta} c, ss = InputSim(c', c)\}$
- $Out_c = \{(Id_A, Id_g, \eta, ss) | Id_A \in AIDs, Id_g \in GIDs, \exists c' \in In(Id_A), c' \xrightarrow{\eta} c, ss = OutputSim(c', c)\}$

The calculation of the similarity between a target concept and its relaxers differs depending on whether it annotates an input or an output. For the case of inputs, the similarity between a concept and its descendant relaxer is 1 since all the attributes required by a target activity input c can be provided by a query activity input which is a descendant of c . Following the same reasoning, the similarity between a target activity output and its ascendants is 1 since the target activity output c can provide all the attributes of a query activity output which is its ascendant. In other cases, any similarity measure [16] can be used to evaluate the similarity between a concept and its relaxer.

The construction of the sets of input and output annotations attached to the concepts of the ontology is done incrementally. Thus, when adding a new activity, only the input and output annotation sets of concepts appearing in this activity

are updated by adding the new annotations. In another hand, when an activity is removed from the repository, only the annotations generated by this activity are deleted. Therefore, adding and deleting an activity do not require the reconstruction of the input and output annotations from scratch, this makes the updating time short.

4.2 Indexing Process Types

As mentioned previously, the p-graphs registered in the repository are compiled into their respective *process types* that are indexed using three hash tables: *Processes*, *Activities* and *FlowDependencies*.

The *Processes* hash table contains the descriptions of the p-graphs registered in the repository, such as the names, the number of their activities, and their storage path. It is indexed by the identifier assigned to the p-graphs. This identifier is auto-generated by incrementing a counter every time that a new p-graph is added to the repository.

The *Activities* hash table stores the descriptions of the activities of the p-graphs registered in the repository. It is indexed by the identifiers assigned to activities that are built by concatenating the identifier of the p-graph to which they belong and a unique identifier distinguishing each activity within the p-graph to which it belongs (generated in the same way as the identifier of the p-graphs). Each entry of this table stores the name, inputs, and outputs of an activity.

The *FlowDependencies* hash table contains the *flow dependencies* specified by the p-graphs registered in the repository. It is indexed by the signatures of these *flow dependencies* that are built as follows.

Let us consider a *flow dependency* $fd_{(i,j)} = (a_i, a_j, T_{(i,j)}, M_{(i,j)})$ occurring in a p-graph, and Aid_i and Aid_j be the identifiers assigned to the activities a_i and a_j . The signature of $fd_{(i,j)}$ is built by concatenating the identifiers of the activities (Aid_i and Aid_j) and the type of the flow dependency ($T_{(i,j)}$): “ $Aid_i.Aid_j.T_{(i,j)}$ ”. The signature of each *flow dependency* is unique since there is only one *dependency flow* between each pair of activities. Each entry of the *FlowDependencies* hash table records the multiplicity and the identifier of the p-graph to which it belongs.

As the activity index, the construction of the index for *process types* is done incrementally, by inserting the description of the p-graph and its activities, and its *flow dependencies* in *Processes*, *Activities*, and *flow dependency* hash tables.

5 Process Retrieval

Given a query p-graph $Q = (A_Q, C_Q, E_Q)$ and a set of indexed p-graphs annotated using the same ontology O , the evaluation of Q operates in four steps as shown by Figure 3. First, the process type PT_Q of Q is established (step *Process Type Generator*) following the procedure presented in section 3. Second, the *Activity Matches Searcher* retrieves within the repository the match-candidates of each activity of Q . Next, *Process Matches Searcher* examines the match-candidates of the activities of Q and determines the set of p-graphs that potentially match Q . The result is a list of p-graphs containing at least one pair of activities similar to a pair of activities of Q and having the same *flow dependency type*. These p-graphs are ranked based on the similarity of their *process types* with PT_Q , i.e. p-graphs sharing more *flow dependencies* with Q are better ranked than those sharing less *flow dependencies*. Based on these p-graph match-candidates, the *Process Composition Searcher* tries to discover complex match candidates by composing the p-graph match-candidates. The similarities of these compositions are then evaluated. This finally leads to a ranked list of target p-graphs. Hereafter, we detail these steps.

5.1 Activity Matches Searcher

Given a query activity $A_q = (N_q, In_q, Out_q)$, the *Activity Matches Searcher* retrieves the set of activities registered in the repository that match A_q .

As mentioned above, an activity $A_t = (N_t, In_t, Out_t)$ registered in the repository is a potential match of A_q if it shares at least one direct or relaxed input/output with A_q , i.e. if it exists at least one input (resp. output) of A_t which is an input (resp. output) of A_q or one input (resp. output) of A_q is a relaxer of an input (resp. output) of A_t . The relaxation rules to be considered to generate these match candidates are defined by the user.

To select these activities, the searcher refers to the input/output annotations attached to the ontology concepts. Specifically, given the set of allowed input

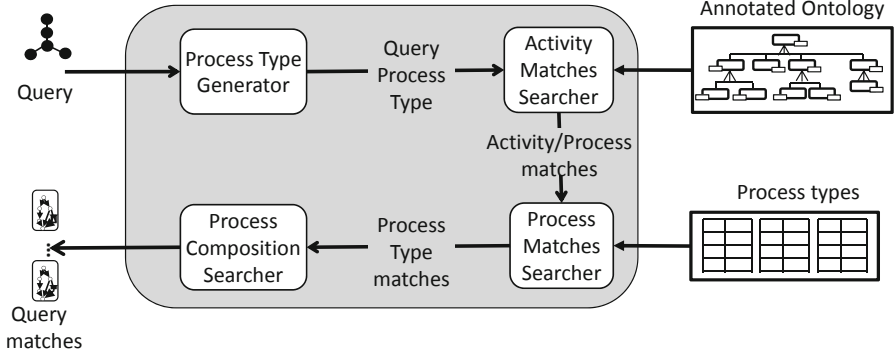


Fig. 3. P-graph retrieval steps

and output relaxation rules specified by a user denoted respectively R_I and R_O , the set of match candidates of A_q is formally described by:

$$ActCandid(A_q, R_I, R_O) = InCandid(A_q, R_I) \cup OutCandid(A_q, R_O).$$

$InCandid$ and $OutCandid$ are the functions that, respectively, compute the sets of the activities that share with A_q , at least, one direct or relaxed input and output. They are defined as follows:

$$InCandid(A_q, R_I) = \bigcup_{c \in In_q} \{Id_{a_t} | (Id_{a_t}, Id_g, \eta, ss) \in InAnnot(c), \eta \in R_I\}$$

$$OutCandid(A_q, R_O) = \bigcup_{c \in Out_q} \{Id_{a_t} | (Id_{a_t}, Id_g, \eta, ss) \in OutAnnot(c), \eta \in R_O\}$$

$InAnnot(c)$ and $OutAnnot(c)$ are the functions that retrieve respectively the input and output annotations attached to the concept c using the index built on the p-graph activities of the repository.

Once the set of match candidates is established, the similarity between A_q and each candidate has to be calculated. The similarity between A_q and a target activity A_t identified by Id_{a_t} is defined on the basis of the similarity between their inputs and outputs as stated by the following formula:

$$ActSim(A_q, Id_{a_t}) = \frac{1}{2} \left(\frac{1}{|In_t|} \sum_{c \in In_q} InSim(Id_{a_t}, c) + \frac{1}{|Out_q|} \sum_{c \in Out_q} OutSim(Id_{a_t}, c) \right),$$

where, $|Out_q|$ (resp. $|In_t|$) is the number of outputs (resp. inputs) of A_q (resp. A_t).

The function $InSim(Id_{a_t}, c)$ (resp. $OutSim(Id_{a_t}, c)$) returns the similarity of the input (resp. output) annotation attached to the concept c whose activity identifier is Id_{a_t} when it exists, zero otherwise. This is shown by the following formulas:

$$\begin{aligned} InSim(Id_{a_t}, c) &= \begin{cases} ss & \text{if } (Id_{a_t}, Id_g, \eta, ss) \in InAnnot(c) \\ 0 & \text{otherwise} \end{cases} \\ OutSim(Id_{a_t}, c) &= \begin{cases} ss & \text{if } (Id_{a_t}, Id_g, \eta, ss) \in OutAnnot(c) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Doing so, some registered activities may be retrieved even if they have a low similarity with the query activity. To keep only promising candidates, we set a similarity threshold that match-candidates have to meet to be considered as potential matches. Therefore, considering a similarity threshold ρ , the set of matches of an activity A_q is: $ActMatches(A_q, R_I, R_O, \rho) = \{Id_{a_t} | Id_{a_t} \in ActCandid(A_q, R_I, R_O), ActSim(A_q, Id_{a_t}) > \rho\}$

Other threshold functions, such as the number of fulfilled inputs and/or outputs, and/or the number of activity match candidates to select can be considered.

Once the activity matches of each activity of Q are established and ordered according to their similarity with this activity, they are passed as input to the *Process Matches Searcher*.

5.2 Process Matches Searcher

Given the sets of activity matches of a query, the *process matches searcher* evaluates the similarity between each p-graph match-candidate and Q , this leads to a ranking of these p-graphs.

The set of p-graph match-candidates of Q is defined as the set of p-graphs with which it shares at least one activity as formalized by the following formula:

$$ProCand(Q, R_I, R_O, \rho) = \bigcup_{A_q \in A_Q} \bigcup_{\substack{Id_{a_t} \in \\ ActMatches(A_q, R_I, R_O, \rho)}} PId(Id_{a_t}),$$

where $PId(Id_{a_t})$ gives the identifier of the p-graph to which the activity identified by Id_{a_t} belongs.

Subsequently, a mapping between the activities of each match-candidate T (identified by Id_T) and the activities of Q is established. This mapping contains all the correspondences found between their activities. It is formally defined as follows:

$M_{Q \leftrightarrow T} = \{(a_q, Id_{a_t}, ss) | a_q \in A_Q, Id_T = PId(Id_{a_t}), Id_{a_t} \in ActMatches(a_q, R_I, R_O, \rho), ss = ActSim(a_q, Id_{a_t})\}$, such that for any pair $(a_q^1, Id_{a_t}^1, ss_1)$ and $(a_q^2, Id_{a_t}^2, ss_2) \in M_{Q \leftrightarrow T}$, $a_q^1 \neq a_q^2$ and $Id_{a_t}^1 \neq Id_{a_t}^2$.

The similarity of this mapping is defined as follows:

$$Sim_{M_{Q \leftrightarrow T}} = \frac{1}{|A_Q|} \sum_{(a_q, Id_{a_t}, ss) \in M_{Q \leftrightarrow T}} ss.$$

As stated above, the similarity between Q and its match-candidate T is estimated in the basis of the similarity of their process types PT_Q and PT_T .

Let us consider $\Pi_{Q \leftrightarrow T}$ be the set of fully matched *flow dependencies* between PT_Q and PT_T , that is defined as follows: $\Pi_{Q \leftrightarrow T} = \{(fd_q, fd_t) | fd_q =$

$(a_1^q, a_2^q, \rho_q, m_q) \in PT_Q, fd_t = (a_1^t, a_2^t, \rho_t, m_t) \in PT_T, (a_1^q, a_1^t, ss_1) \in M_{Q \leftrightarrow P}, (a_2^q, a_2^t, ss_2) \in M_{Q \leftrightarrow P}, \rho_q = \rho_t, m_q = m_t\}$.

Let us also consider $\Pi'_{Q \leftrightarrow T}$ be the set of partially matched *flow dependencies* between PT_Q and PT_T , that includes the *flow dependencies* having the same type and having their respective activities matched but having different multiplicities. Its formal description is defined as follows: $\Pi'_{Q \leftrightarrow T} = \{(fd_q, fd_t) | fd_q = (a_1^q, a_2^q, \rho_q, m_q) \in PT_Q, fd_t = (a_1^t, a_2^t, \rho_t, m_t) \in PT_T, (a_1^q, a_1^t, ss_1) \in M_{Q \leftrightarrow P}, (a_2^q, a_2^t, ss_2) \in M_{Q \leftrightarrow P}, \rho_q = \rho_t, m_q \neq m_t\}$.

Finally, we define the similarity between Q and T as follows.

$$SimPro(Q, T) = \omega_m * Sim_{M_{Q \leftrightarrow T}} + \omega_f * \frac{|\Pi_{Q \leftrightarrow T}|}{|PT_Q|} + \omega_p * \frac{|\Pi'_{Q \leftrightarrow T}|}{|PT_Q|}$$

Weights $0 \leq \omega_m \leq 1$, $0 \leq \omega_f \leq 1$, and $0 \leq \omega_p \leq 1$ ($\omega_m + \omega_f + \omega_p = 1$) indicate the contribution of respectively, the matched activities between Q and T , fully and partially matched *flow dependencies* to establish this similarity.

It should be noted that the computation of the shared flow dependencies between a query and a target p-graphs is enhanced using the indice built on the flow dependencies of the p-graphs of the repository.

5.3 Process Composition Searcher

As mentioned before, in some cases a query cannot be satisfied by a single target p-graph, but by a set of p-graphs composed using control structures (sequence, parallelism, alternative branches). The idea is then to develop a technique that is able to propose the composition of several p-graphs as an answer. Such kind of responses are very helpful since it relieves the user of a very tedious composition task.

Informally, two p-graphs can be composed when their respective activities are matched against subgraphs that are disjoint. Figure 4 shows a query and a set of its match candidates. The match candidates T_1 and T_2 are good candidates for composition since their respective activities are matched against subgraphs that are disjoint, while the composition of T_3 and T_4 is not possible since the subgraphs of the query covered by these matches overlap. In other words, a set of p-graphs can be composed if they are mapped to subgraphs of the query that are composable in sequence or using connector nodes of types XOR or AND. This is formalized by Definition 8. This definition states that two p-graphs are composable when the type of the *flow dependencies* occurring between each pair of their respective match activities in the query are of the same type.

There is an exception to this rule that occurs when the two p-graphs are matched against two parts of the query that are in sequence. The exception states that if the *flow dependencies* occurring between the activities of the query matched to a p-graph G_1 and the activities matched to another p-graph G_2 is of type *PATH*, it may exist at most one pair of activities $a_1 \in G_1$ and $a_2 \in G_2$ having a *flow dependency* of type *SEQ*.

Definition 8. Process composition. Let us consider a query p-graph $Q = (A_Q, C_Q, E_Q)$ and a set of its match-candidates: $T_1 = (A_1, C_1, E_1), \dots, T_k = (A_k, C_k, E_k)$. Let $M_{Q \leftrightarrow T_1}, \dots, M_{Q \leftrightarrow T_k}$ be the mappings found between Q and respectively T_1, \dots, T_k . Let $M_{Q \rightarrow T_1} = \{a_q | (a_q, a_{T_1}, ss_1) \in M_{Q \leftrightarrow T_1}\}, \dots, M_{Q \rightarrow T_k} = \{a_q | (a_q, a_{T_k}, ss_2) \in M_{Q \leftrightarrow T_k}\}$ be the sets of the mapped nodes of Q following respectively the mappings $M_{Q \leftrightarrow T_1}, \dots,$ and $M_{Q \leftrightarrow T_k}$. P-graphs $T_1, \dots,$ and T_k are composable to answer the query Q iff:

- $M_{Q \rightarrow T_1} \cap \dots \cap M_{Q \rightarrow T_k} = \emptyset$
- for each pair $M_{Q \rightarrow T_i}$ and $M_{Q \rightarrow T_j}$, and each pair of activities $a_i^1, a_i^2 \in M_{Q \rightarrow T_i}$, and each pair of activities $a_j^1, a_j^2 \in M_{Q \rightarrow T_j}$: $fd(a_i^1, a_j^1).Type = fd(a_i^2, a_j^2).Type$

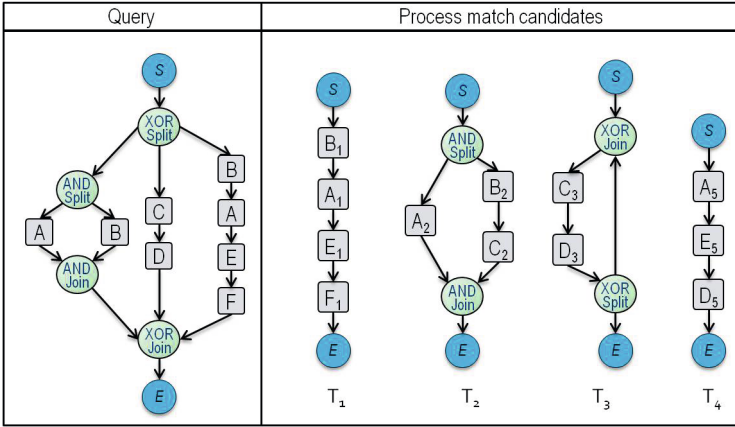


Fig. 4. Process composition example

The composition of a set of p-graphs results in the fulfillment of other *flow dependencies* specified by the query that are not satisfied by one p-graph taken alone. Therefore, the set of fulfilled flow dependencies of the composition of two p-graphs T_i and T_j is defined as follows: $I(T_i, T_j) = \{(a_1, a_2, \gamma, m) | a_1 \in M_{Q \rightarrow T_i}, a_2 \in M_{Q \rightarrow T_j}\}$, where, γ is the type of the *flow dependency* between each pair (a_1, a_2) (it is unique since T_1 and T_2 are composable), m is its multiplicity which is also the same of all the pairs of activities because the composition rule allows only the composition of p-graphs that are mapped to disjoint sub-blocks.

Let $T_s = \{T_1, \dots, T_k\}$ be the set of p-graphs to be composed in order to answer the query Q , the set of *flow dependencies* of the query satisfied by the composition of the p-graphs of T_s is:

$$I_Q = \bigcup_{i=1}^{|T_s|} \bigcup_{j=i}^{|T_s|} (I(T_i, T_j)).$$

Therefore, the similarity between Q and T_s is defined as follows:

$$SimCompo(Q, T_s) = \sum_{i=1}^{|T_s|} SimPro(Q, T_i) + \omega_f * \frac{|I_Q|}{|PT_Q|}, \text{ where } \omega_f \text{ is as defined in section 5.2}$$

6 Implementation and Experiments

We implemented our technique on top of a platform for matching p-graphs [4] and experimentally evaluated it. One of the problems we faced to conduct our experiments is the lack of a public benchmark over which we can test our technique, and against which we could compare its result. To overcome this, we built

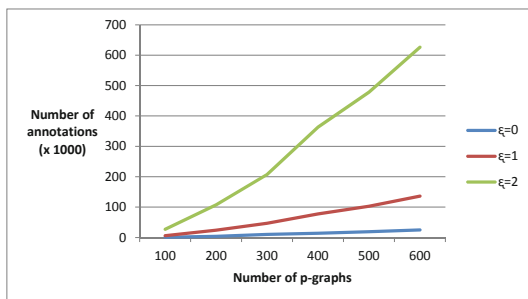


Fig. 5. Number of annotations of the ontology Vs the number of p-graphs and ξ

up our own test collection that we generated by considering the mismatches that most often occur in real life p-graphs and a set of p-graph characteristics that could impact the performances of our technique. The collection contains 623 p-graphs annotated using an ontology containing 500 concepts. On average each p-graph contained 19 activities with a minimum of 2 and a maximum of 83 activities. The average size of an activity name is 2 words with a minimum of 1 and a maximum 8 words.

We randomly extracted from this collection 30 query p-graphs and 300 p-graph targets. We then manually evaluated the similarity between each query and the targets in a 1-7 Likert scale and we subsequently ranked the targets according to their similarity with each query. Details about the methodology we followed to build this collection and its characteristics can be found in [8].

The first experiments were dedicated to the study of the size of the indexes according to the number of indexed p-graphs and the relaxation degree (ξ). From Figure 5, we can see that the number of annotations generated when adding p-graphs is approximately linear with respect to the number of indexed p-graphs (the tendency is the same whatever the value of ξ). In addition, the number of generated annotations for the same number of indexed p-graphs increases, as expected, with the increasing of the value of ξ . Nevertheless, the number of annotations remains reasonable for a repository containing 600 p-graphs (630k

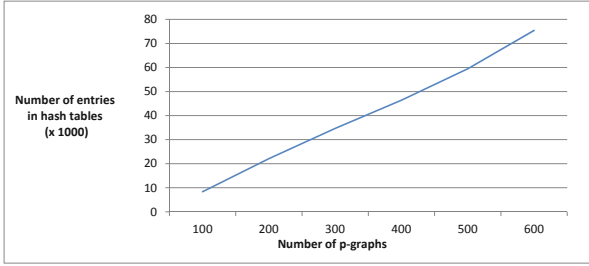


Fig. 6. Sizes of the hash tables Vs the number of p-graphs and ξ

when $\xi = 2$). From Figure 6, we can see that the size of the hashtable index is also approximately linear with respect to the number of indexed p-graphs. For instance, the indexing of 600 p-graphs required 77k entries in the hashtables, and on average, the indexing of a p-graph required 128 entries. The size of the indexes could become prohibitive for repositories containing thousands of p-graphs, but it supports the indexing of current repositories that, in majority of cases, contain less than 1000 p-graphs. It should be noted that the time for indexing 600 p-graphs is about 300 seconds. Although the indexing of a large number of p-graphs at the same time may require considerable time, it is not penalizing since it is done in an off-line preprocessing step.

Figure 7 shows the effect of varying the number of indexed p-graphs and the relaxation degree ξ (from 0 to 2) on the average and maximum times spent for answering the queries of our benchmark. The results show that query answering is done within low times. For example, the maximum and average time for answering a query are respectively equal to 37 ms and 95 ms when the number of indexed p-graphs is equal to 600 and ξ is equal to 2 (the worst case in our experiments). These results also show that the time of query answering increases

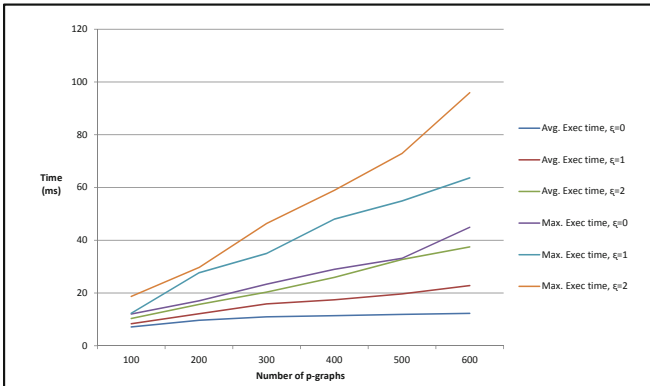


Fig. 7. Query evaluation times vs the number of p-graphs and ξ

with the increasing of the number of indexed p-graphs, and the tendency of this increasing is approximatively linear (whatever the value of ξ).

Finally, we evaluated the quality of the rankings found by our technique with respect to the number of the indexed p-graphs and the value of ξ . The indexes are built only on p-graphs for which a manual evaluation was made (300 p-graphs). The effectiveness of the rankings is evaluated using the well known NDCG formula formalized in Definition 9. The results of this experiment are shown by the graphic of Figure 8. As shown by this graphic, the results obtained by our technique are very satisfactory. For instance, the NDCG obtained when $\xi = 2$ and the number of indexed p-graphs is 300 is equal to 0.81. We also observed that the value of NDCG slightly decreases when the number of indexed p-graphs increases. However, the decreasing is negligible when $\xi = 2$.

Definition 9. NDCG measure. Let $\Psi = [P_1, P_2, \dots, P_n]$ be the ranking found by an algorithm for a given query Q_k , and $\delta^{(k,i)}$ be the similarity degree between Q_k and PM_i given by an expert. Let Z_n be the DCG corresponding to the manual (best) ranking. The effectiveness of the ranking Ψ is: $NDCG_n = \frac{1}{Z_n} \sum_{i=1}^n \frac{2^{\delta^{(k,i)}} - 1}{\log_2(i+1)}$.

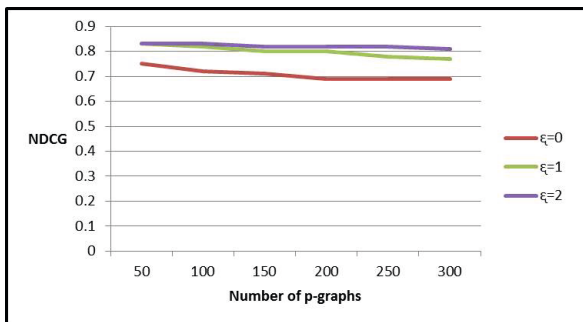


Fig. 8. NDCG Vs the number of indexed p-graphs and ξ

7 Related Work

To the best of our knowledge, the only works addressing the problem of process retrieval in repositories are [17,18,12,1].

In [17], an index build on top of a RDBMS relates a process model to a set of partial traces of length N . A feature-based filtering technique is proposed in [18] to search a collection of processes specified using process graphs. The target processes are ranked according to the number of shared features with the query. Features are small subgraphs that may be: the first and last activities, sequence of a given length, a split node and its successors, and a join node with its predecessors.

A visual query language extending BPMN is proposed in [1] to query graph-based process collection. The processes are stored in a relational database that

stores process models nodes (activities and gateway), edges, and paths. The approach is extended in [2] to handle differences of vocabularies that may occur between the query and the stored processes.

Recently, another query language is proposed in [12] that allows specifying the behavioral relationships (two activities can be in sequence, parallel or exclusive branches) that must fulfill the processes to retrieve. The relationships that occur in the stored processes are indexed using an inverted list that relies each behavioral relationship to the set of processes where it occurs.

Many other works [3,10] addressed the problem of measuring the similarity between two processes. These methods sequentially traverse all the processes of the repository and compare each process against the query. The majority of these algorithms are NP-complete and therefore they do not suit for searching similar processes of a query within a repository of processes.

To summarize, we proposed a fast similarity search technique that allows retrieving within a repository the processes the more similar to the query, while the above techniques allow only exact structural matches. Other novel feature of our approach is that it allows proposing a composition of processes in the repository to answer a user query.

8 Conclusion

We propose in this paper an effective and fast similarity search technique that allows retrieving within a process repository, the processes the most similar to the query. Moreover, our approach allows proposing the composition of processes in the repository to answer his query. To do so, we use an abstraction function that represents a process model as a finite set of representative flow dependencies, which are indexed along with the process activities. We implemented our algorithms in a larger platform for matching process models [4] and experimentally evaluated it. Experiments show that our technique has very good execution times.

To reduce the size of indexes, mainly for managing very large repositories, we are currently investigating methods of ontology encoding that reduce the size of the annotated ontology. We also work on methods that allow a more compact representation of the process type relation.

References

1. Awad, A.: Bpmn-q: A language to query business processes. In: EMISA, pp. 115–128 (2007)
2. Awad, A., Polyvyanyy, A., Weske, M.: Semantic querying of business process models. In: EDOC, pp. 85–94 (2008)
3. Becker, M., Laue, R.: A comparative survey of business process similarity measures. *Computers in Industry* 63(2), 148–167 (2012)
4. Corrales, J.C., Grigori, D., Bouzeghoub, M., Burbano, J.E.: Bematch: a platform for matchmaking service behavior models. In: EDBT, pp. 695–699 (2008)

5. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 498–516 (2011)
6. Eshuis, R., Grefen, P.: Structural matching of bpm processes. In: Fifth European Conference on Web Services, Halle (Saale), Germany, pp. 171–180 (2007)
7. Euzenat, J., Shvaiko, P.: *Ontology Matching*. Springer, Heidelberg (2007)
8. Gater, A.: *Process matching and discovery*. PhD thesis, University of Versailles (2012)
9. Gater, A., Grigori, D., Bouzeghoub, M.: Complex mapping discovery for semantic process model alignment. In: *IIWAS* (2010)
10. Gater, A., Grigori, D., Bouzeghoub, M.: A graph-based approach for semantic process model discovery. In: Sakr, S., Pardede, E. (eds.) *Graph Data Management: Techniques and Applications*, pp. 223–233. Information Science Reference (2011)
11. Grigori, D., Corrales, J.C., Bouzeghoub, M., Gater, A.: Ranking bpm processes for service discovery. *IEEE T. Services Computing*, 178–192 (2010)
12. 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)
13. Klusch, M., Fries, B., Sycara, K.: Automated semantic web discovery with owls-mx. In: *AAMAS 2006* (2006)
14. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring Acyclic Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) *BPM 2010*. LNCS, vol. 6336, pp. 276–293. Springer, Heidelberg (2010)
15. Sakr, S., Al-Naymat, G.: Graph indexing and querying: a review. *International Journal of Web Information Systems* 6(2), 101–120 (2010)
16. Valery, C.: Fuzzy semantic distance measures between ontological concepts. In: *NAFIPS*, pp. 635–640 (2004)
17. Wombacher, A., Mahleko, B., Fankhauser, P.: A grammar-based index for matching business processes. In: *ICWS 2005* (2005)
18. 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, Part I*. LNCS, vol. 6426, pp. 60–77. Springer, Heidelberg (2010)

A Framework for Cost-Aware Cloud Data Management

Verena Kantere

Cyprus University of Technology
verena.kantere@cut.ac.cy

Abstract. The emerging world of offering information services through the cloud necessitates the coalescence of existing research and business technologies into the provision of all-inclusive solutions for data management. This paper proposes a framework that can support cost-aware data management in the cloud. Users and cloud providers can use the framework to receive and provide services that comply with agreements on data service cost and requirements and allow for profit while being efficient in terms of performance. The proposed framework includes modules that incorporate the notion of monetary cost in current data management, but also modules that take optimization decisions for future data management taking into account both monetary cost and performance. The framework dictates the design of a middleware application that can be plugged on top of a cloud data management system. Such a middleware receives the user's workload and preferences for cost and query performance and controls data management so that the user is satisfied and the cloud provider is viable and, furthermore, profitable. An initial realization of part of the framework as a middleware application has already been constructed, tested and published with promising results.

Keywords: Cloud data services, cloud data management, cost-aware data management, cloud economics.

1 Introduction

The new trend for service infrastructures in the IT domain is called cloud computing, a style of computing that allows Internet users of any expertise to access information services on the web. Information, as well as software is permanently stored in Internet servers and probably cached temporarily on the user side.

Outsourcing the archiving and manipulation of large persistent datasets, such as scientific data of large scale, small-and-medium enterprise data as well as personal datasets gains more and more credence. The general requirement of users is to manage efficiently the data with a dynamic demand for additional or reduced computational support, while investing as little as possible time and money. Unfortunately, current commercial data management tools are insufficient to meet these requirements especially while they are not tailored for specific data and users. Also, they are usually incapable to support the unprecedented scale, rate, and complexity of big data collection and processing.

The cloud computing paradigm seems the right choice for designing infrastructures and moreover applications that can meet the above data management expectations.

The cloud alleviates the burden of data management from the users by offering transparent data services for remuneration. Emerging IT business in cloud computing [1, 2, 3] is an effort to offer such services. Such a cloud provider necessitates various technological capabilities. Most importantly, it is required that cloud data services run with minimal capital expenditure, but, nonetheless, can support efficiently multi-user tenancy.

A cloud data service provider offers its services on cloud databases. Such databases support the archiving of user data in the cloud employing caching techniques. Data that reside in the cloud, i.e. cloud data, can benefit from the offered cloud data services. Users are customers of the cloud that consume its resources as a utility service. Specifically, the users can query the cloud data, paying the price for what they use. User payment is employed for coverage of short and long-term costs. Short-term cost refers to the respective query execution, and long-term cost to the self-preservation of the cloud infrastructure and improvement of the cloud services.

We propose a generic framework that aims to coalesce the existing research and business technologies into the provision of an all-inclusive solution for the management of data in the cloud. The framework is designed to provide transparent cost-aware data management on top of a cloud database, whatever the latter is: a traditional DBMS deployed on a cloud infrastructure or a new prototype of a cloud DBMS. The purpose of the proposed framework is to provide an economy solution for a cloud data management system that is inherently bound to the technical solutions. Currently, our focus is on scientific data, since their variety, volume and extreme data management requirements makes them the best candidates for research and experimentation. The framework, however, is generic and applicable to any kind of data that may need cloud management.

The framework aims to support an economy for a cloud infrastructure that handles structured data, i.e. data that are stored in databases. Following the business line in cloud computing, the proposed economy employs a cost model that takes into account all the available resources in a cloud, such as disk space and I/O operations, CPU time and network bandwidth. The economy is self-tuned to the policies that aim to: (i) high quality of individual query services, (ii) increasing overall quality of query services, and (iii) cloud profit. The experience of data management is accumulated and quantified by a regret scheme in order to assist the improvement of cloud services by building new data structures, (i.e. cached data and indexes). The cost of new data structures is amortized to prospective users based on a model that predicts future service consumption. Finally, cloud profit is ensured and maximized using an appropriate pricing scheme. The framework may include an optional module that comprises a query planner and query router that enable the servicing of sets or sequences of queries. The cloud economy can be extended in order to handle such query combinations. In this way, the cloud framework can serve in an optimal manner even demanding tasks of data analysis (which are very common in scientific data management). Finally, associative modules that take care of risk management and estimate data service correlations are necessary in order to use the framework in real cloud environments.

An initial application of a simplified form of the framework is already designed, developed, tested and published in [4, 5, 6]. This experience has proved the applicability and effectiveness of the framework even in a simplified form. Our goal is to extend our existing work with the general and elaborated form of the framework presented in this paper.

2 Related Work

Current research on cloud computing considers an infrastructure that comprises a set of independent edge caches that cooperate in order to deliver web content. Content sharing among caches [7, 8, 9] involves (i) content retrieval from sibling caches instead from the server, (ii) routing and sharing of content updates, and (iii) sharing cache resources, in order to achieve efficient collaborative data placement/replacement/update/lookups etc. Being more compliant with the business domain, this proposal focuses on self-tuned cloud caches that share resources, rather than self-organization of independent caches. Concerning the management of scientific data, existing research solutions [10], consider network bandwidth to be the only important resource, and, therefore, the sole basis for cost computation. However, cloud businesses usually prorate cost to more types of resources. For instance, GoGrid [8] gives network bandwidth for free. A self-tuned cache [11] reduces query execution costs adaptively to the workload. As a step further towards commercial applications, we propose an economy that takes into consideration the overall cost of the infrastructure beyond network bandwidth: disk I/O, storage and CPU.

Cloud computing is the natural dilation of grid computing [12], as it enables an integrated collaborative use of high-end computing owned and managed by multiple organizations. Grid databases [13] are federated database servers over a grid, which are viewed as a virtual database system through a service federation middleware [14]. Querying the grid data guaranteeing high performance is one of the key issues for distributed queries across large datasets. This issue is inherited in cloud computing, and becomes more complicated, since it is augmented with the issue of providing an accounting service that supports a payment scheme for cloud usage. We take this challenge and we propose a framework of data-aware cloud economy that fills the essential gap of tuning the provision of data management services for remuneration.

Related to the proposed framework for cloud economy is the research on auctioning and accounting systems. Accounting in wide-area networks that offer distributed services have long been the target of research in computing. Mariposa [15] discusses an economic model for querying and storage in a distributed database system. In Mariposa clients and servers have an account in a network bank and users allocate a budget to each of their queries. The processing mechanism aims to service the query in the allotted budget by executing portions of it on various sites. The latter place bids for the execution of query parts, and the bids are accumulated in query brokers. The decision of selecting the most appropriate bids is delegated to the user. In contrast, our cloud proposal can recommend to the user an efficient but also profitable for the cloud query plan. In the spirit of Mariposa, a series of other works have proposed solutions for similar frameworks [16, 17, 18, 19, 20]. These focus on job scheduling and bid negotiation, which are orthogonal issues to those that are tackled by the proposed framework.

Concerning the special case of scientific data, their management has been a challenge until now. The need for in-depth analysis on huge amounts of data has increased the demand for additional computational support. Unfortunately, current commercial data management tools are incapable of supporting the unprecedented scale, rate, and complexity of scientific data collection and processing. Independently of the categories that scientific data belong to, their management is essentially divided into the

following coarse phases: workflow deployment, management of metadata, data integration, data archiving and finally data processing [21]. All these phases suffer from tremendous data management problems that concern automation, online processing, data and process integration and file management [22].

The proposed framework enables the leverage of the management of scientific data by a cloud provider, achieving transparency of data management solutions that are efficient and cheap. Currently, scientists need to tightly collaborate with computer engineers in order to develop custom solutions that efficiently support data storage and analysis for each different experiment [23, 24]. Beyond the fact that constant collaboration of multidisciplinary scientists and engineers is hard, time and effort consuming, the experience gained by such collaboration is not inherited widely to the scientific community, so that next generation experimental setups can benefit from it. It is absolutely necessary to develop generic solutions for storage and analysis of scientific data that can easily be extended and customized. Developing generic solutions is feasible, since there are low-level commonalities in the way that experimental data are represented or analyzed [21, 24]. Therefore, frequently, scientific data processing encompasses generic procedures. Current research, however, has not proposed any solution for the support of such processes.

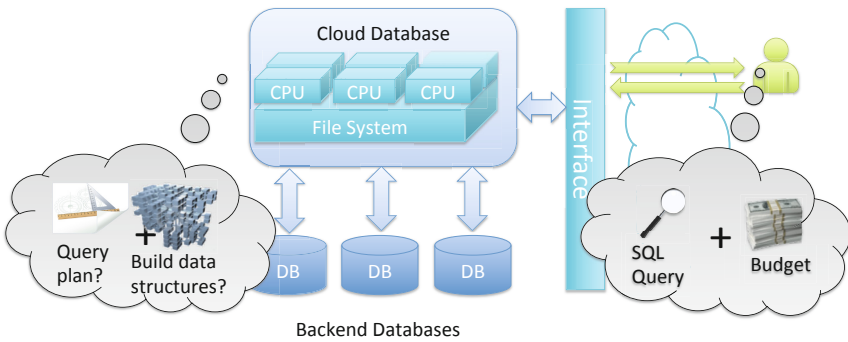


Fig. 1. Operation of a cloud data service provider

3 Operation of a Cloud Data Service Provider

In a cloud environment, there are the users and probably owners of data on one hand and the cloud data service provider on the other. In the general case, the data reside permanently in backend databases. The cloud database supports caching of these data in order to offer efficient query processing. Users pose queries to the cloud, which are charged in order to be served. The goal of the cloud is to provide efficient query processing at low price, while being profitable. Query performance is measured in terms of execution time. Naturally, query execution is accelerated with the number of available data structures, i.e. indexes and cached data. Rationally, the faster the execution, the more expensive the service is. The user defines her preferences concerning the service of her query by indicating the budget she is willing to spend on the query, according to the respective execution time that the cloud can provide.

The cloud receives the query and the budget/execution-time preferences of the user and produces respective alternative query plans. The cost of these query plans is estimated and juxtaposed to the preferences of the user. If the cheapest plan is to execute the query on the backend database (i.e. the permanent storage of the data), the cloud outsources query execution. Otherwise, the query is executed on the cloud database, either on data that is already cached or by caching data from their permanent storage. If there are query plans that the user can afford, according to her budget, then the cloud picks up the most appropriate one of them, w.r.t. the supported policies. If the cost of alternative query plans is over the user budget, the user is presented with the alternative options and can choose which one she is willing to pay for, if there is such one. The profit from each query service is credited to the cloud account and is employed in order to build data structures that can improve the query services. The cost of the new structures is amortized to prospective users that will consume services that employ these structures. Fig. 1 shows graphically the operation of the cloud data service provider.

3.1 Cloud Data Services

The cloud offers services on the data that the user can access. These services are summarized as follows:

1. **Data storage design:** The user may request from the cloud to implement either a user pre-defined or a cloud design for the storage of her data in the cloud database.
2. **Query execution:** The user may request the execution of a query or the execution of a whole workload. The cloud has to create alternative query plans and select one for execution. Furthermore, the cloud has to dynamically parallelize the execution on the cloud infrastructure.
3. **Query optimization:** The cloud has to optimize query execution by creating and maintaining dynamically and automatically (i.e. without the intervention of the user) data structures, such as indexes, cached data columns and data views.

All the above services incur an infrastructure and administration cost that, through their provision, is alleviated from the user and burdens the cloud data service provider. The proposed framework aims at handling this economic burden in the best possible way, such that the provider remains viable and furthermore profitable.

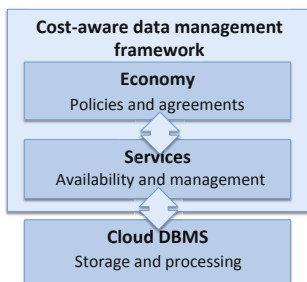


Fig. 2. Collaboration of the framework and the cloud DBMS

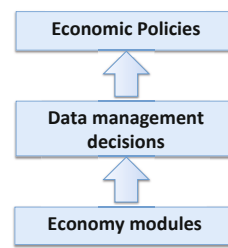


Fig. 3. Internal operation flow in the proposed framework

3.2 Cloud Economic Policies

As mentioned, a query plan is selected according to supported policies. Our framework supports the following policies, which can lead to a viable and even more a profitable operation of the cloud data service provider.

1. **Individual user satisfaction:** The service provider should keep each user satisfied with the data services she receives. User satisfaction is achieved if the following requirements are satisfied:
 - Respect user preferences for query execution: The quality of the data services that the user receives should comply with her preferences for query execution. For example, a user that requests a fast or cheap query execution, should receive query execution that is fast or cheap, respectively.
 - Avoid over-charging data services: Each service should be priced close to its actual cost, i.e. the cloud intention is to provide cost-competitive services.
 - Respect user budget constraints: The user should receive services that are always priced in her declared budget.
2. **Data service improvement:** The cloud provider should be able to offer improved services as time goes by. Service improvement can be sought along two directions:
 - Offer broader and appropriate services: The cloud has to increase the variety of offered services with time. This means that it should build more data structures, which can be used in offered query execution plans. Moreover, the cloud has to change or discontinue services that are no longer useful or popular.
 - Offer services at a lower price: Ideally, as time goes by, the cloud should be able to offer the same services, e.g. the same data structures, at a lower price.
3. **Cloud profitability:** A commercial cloud provider is a business and as such it aims at making profit. The latter can be guaranteed by the following:
 - Set optimal price: Services should be priced above their actual cost in a way that the overall cloud profit is maximized.
 - Schedule optimal availability: The cloud has the option of discontinuing services that are not popular and creating services that are. This means that data structures that are not used often in user query execution should be evicted from the cloud DBMS and data structures that are used often should be constructed and maintained. Fig. 2 and Fig. 3 show conceptually how the framework sits on top of a cloud DBMS and what is the overall internal operation the framework.

3.3 Cloud Layering Architecture

As it is widely known, the cloud computing paradigm dictates the offer of services on three levels, namely the infrastructure (IaaS), the platform (PaaS) and the software (SaaS). In case of cloud data service provision, the infrastructure is the cloud cluster, which includes CPUs, disks, memory and I/O and network communication; the platform is the cloud DBMS deployed on the infrastructure and the software is a specific database application. A cloud data service provider may offer all three levels as an

all-inclusive service, i.e. such a provider may own the infrastructure and manage both the cloud DBMS and the database applications on top of it. Alternatively, the cloud data service provider may not own the infrastructure, but rent the latter from an IaaS provider, and manage only the cloud DBMS and the database applications. Even more, the cloud data service provider may rent the cloud database from a PaaS provider. In any of the three cases, the offered data services incur a cost that is either the direct cost of operating the cloud infrastructure, or the cost of renting it through mediators. Fig. 4 shows the 3 alternative architectures. Our framework takes the infrastructure cost as an input, no matter if this originates from operating or renting the infrastructure. Therefore, the framework is applicable to any of the three layering architectures.

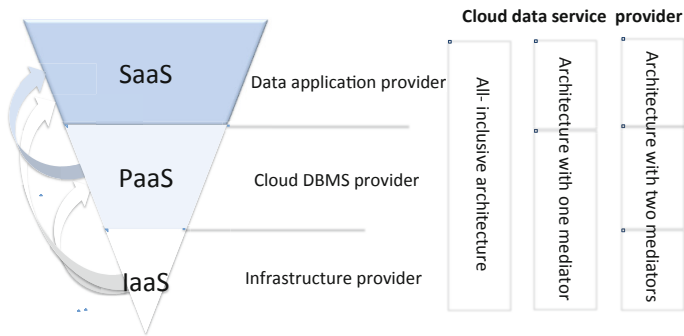


Fig. 4. Overall architecture and mediation of the cloud data service provider

4 Cost-Aware Data Management

The cloud data service provider needs to perform traditional data management, i.e. data management that aims at performance, while taking into consideration cost restrictions. The latter are input to the provider, which has to decide what to output as an offered service. The input can be broken down to four categories:

1. **Data requests:** This refers to what the user asks for. Usually, it is query execution, but it can be a workload execution or other data tasks, such as data replication.
2. **Data updates:** This refers to the rate and extent of data update. When data is updated in the backend databases (in case it is permanently stored out of the cloud), the updated data has to be reloaded into the cloud database. Moreover, data update incurs an update of all relevant data structures.
3. **Infrastructure cost:** This refers to the cost of operating or renting the infrastructure, namely CPU, disk, I/O operations and network.
4. **User budgets:** This refers to how much the users are willing to spend on monetary compensation for the data services they receive.

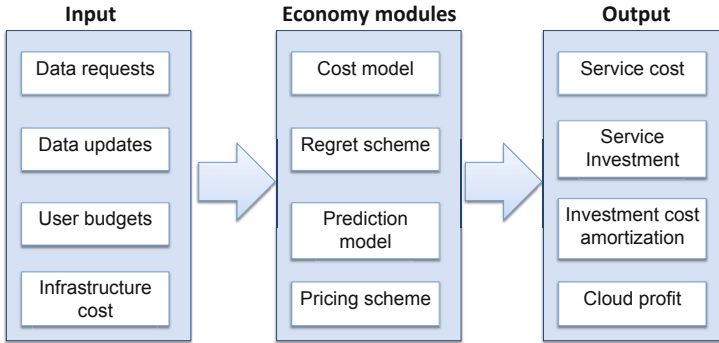


Fig. 5. Operation of the proposed framework

Considering the above input, the cloud has to decide what are the services it should offer, which can be broken down to two decisions: (i) how to service each individual request and (ii) what should be the available services. In order to take these decisions, the cloud has to answer to the following questions:

1. What is the cost of a possible service?

Assuming that there is a pool of possible services to be offered alternatively for the fulfillment of a data request, the cloud needs to decide which one complies best with all the cost restrictions.

2. In which services should money from the cloud account be invested?

The cloud needs to create pools of services that can serve alternatively or complementary data requests. For example two query plans may each include a different index, and these indexes can be used alternatively or in combination to serve the same query. To create these services the provider needs to invest cloud money, which will cover the cost of operating the infrastructure.

3. What is the depreciation of a service since it becomes available?

In order to keep the cloud data service provider viable, the cloud money invested in the creation of possible services needs to be recuperated from user payments. The cloud needs to know how much and for how long it can recuperate the cost.

4. How can the cloud make profit?

The cloud data service provider is usually a business, and, as such, it aims not only to be economically viable, but also profitable. Thus, it is essential for the provider to know if and how much profit it can make from the offered services.

To answer the above questions our framework comprises four basic generic modules: (i) a service cost model, (ii) a data management regret scheme, (iii) a prediction model for service consumption and (iv) a service pricing scheme. These are presented in the following. The operation of the proposed framework is shown in Fig. 5.

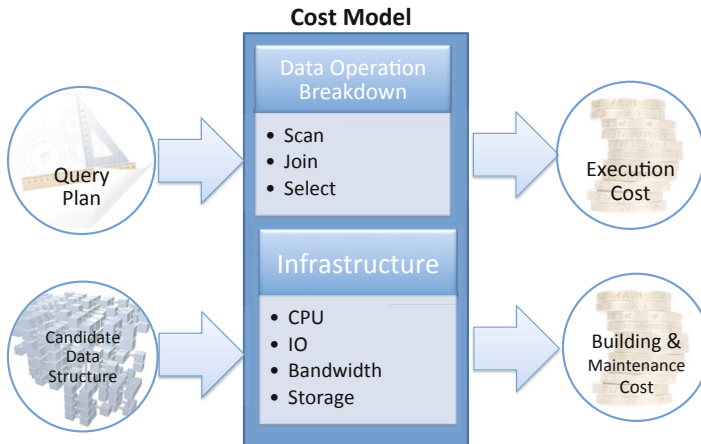


Fig. 6. Operation of the cost model

4.1 Service Cost Model

A request for query execution received by the provider is distributed to the appropriate CPU node(s), or to the backend databases (the actual distribution and execution algorithm depends on the cloud DBMS and is a black box to framework). It is essential for the economy of the cloud provider to estimate correctly the utility cost of the cloud infrastructure. Based on such estimations, the cloud can decide the compensation requested from the users for the offered query services. The cost model takes as input either (i) a query execution plan or (ii) a data structure that is to be built from scratch or to be updated. For the query plan, the cost model estimates what is the execution cost and for a data structure it estimates what is (a) the cost for building and (b) the cost of maintaining the structure in the cloud DBMS. In order to estimate the total cost, the model takes into account (i) what are the necessary primitive data operations for each input (e.g. scan, join, select, project) and (ii) how much of the infrastructure will be used (CPU, I/O operations, network bandwidth and disk space). The operation of the cost model is shown in Fig. 6.

It is safe to assume that the cloud provides unlimited amount of storage space, CPU nodes, and very high-speed intra-cloud networking. Also, the CPU nodes are usually all identical to each other. Compared to TCP bandwidth on Internet, the intra-cloud bandwidth is orders of magnitude faster and we can ignore the overhead associated with it. We assume that the storage system is based on a clustered file system, where the disk blocks are replicated and stored close to the CPU nodes accessing them. Also, we assume that the virtual disk is a shared resource for all the CPU nodes in the cloud DBMS (share-all architecture).

Since the cloud DBMS considers many alternative possible query plans, i.e. design configurations for executing a query, accurate and fast estimation of the cost associated with each plan is very important. The execution time of a query is estimated based on a respective query plan. Currently, in our work we have only considered

plans that run thoroughly in the backend or in the cloud DBMS. Concerning queries that run in the cloud DBMS completely, the estimation of the execution cost is determined based on a plan that is suitable for the cloud. Therefore, the implementation of the cost model module needs to be aware and take into account statistics on query execution on the specific cloud DBMS. Such statistics can be requested by the cost model from the cloud DBMS based on query templates. Then, specific queries can be matched with the appropriate template and their execution cost can be estimated based on the statistics for the template.

As mentioned, the cost model is used to estimate the cost of building or maintaining (i.e. updating) a data structure, too. The latter can be cached data columns from the permanent data storage, views and indexes on cached columns. For the estimation of the building and maintenance cost of a data structure, the cost model has to use statistics on creating similar structures, too. These statistics can be collected by the cloud DBMS optimizer during a training period, or interleaved training periods.

4.2 Data Management Regret Scheme

The data management regret scheme accumulates and quantifies the experience of the cloud provider in taking data management decisions in order to help the provider to take better, in terms of cost and performance, decisions in the future. In effect, this means that the cloud aims to execute in the future similar data requests to current ones in a cheaper and faster manner. This can be achieved if the cloud builds and maintains appropriate data structures and utilizes the right parts of the infrastructure (i.e. how many CPUs and how much memory to use for servicing the data requests, and how much disk to use for data replication).

Essentially, the absence of a data structure in the cloud DBMS prohibits the provider from offering services, i.e. query execution plans, which employ this structure. Also, having to rent (or switch on, depending on the cloud provider architecture and mediation) more infrastructure means a ‘cold start’ for caches and a delay in CPU and disk utilization. The goal of the regret scheme is to monitor how many times the provider has not been able to offer a query plan (or a data operation in general) that would satisfy the user preferences in the best possible way due to the unavailability of a data structure or part of the infrastructure. To achieve this, query plans have to be broken down into basic plans and compared to each other with respect to the data structures they comprise and the infrastructure (especially the parallelization degree, i.e. number of CPUs used). This comparison leads to the formulation of a ‘regret’ for the unavailability of data structures and infrastructure, which has to be quantified and accumulated, so that it can be an indication that the investment in building and offering this data structure is a good or bad data management decision. The regret scheme can also include the decision making part for creating data structures or adding infrastructure, based on the accumulated regret. This decision making has to be based on the policies presented in Section 3.2, with a customizable prioritization. The operation of the regret scheme is shown in Fig. 7.

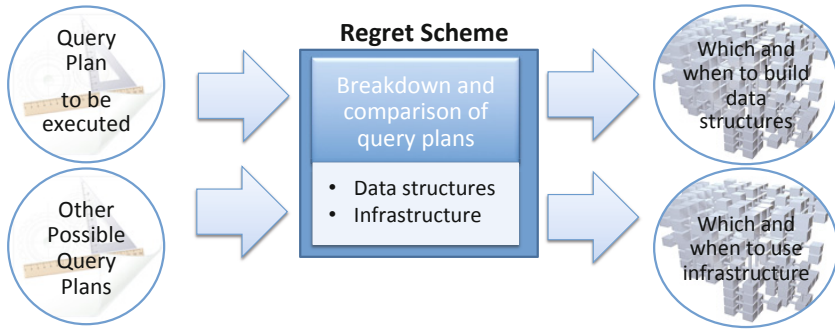


Fig. 7. Operation of the regret scheme

4.3 Prediction Model

The regret leads to the improvement of data services (see Fig. 8). Yet, the cost of investment in new data structures is paid from the credit in the cloud account. The intention of the provider is to remain economically viable, by amortizing this cost to prospective users that receive data services that use the new inventory. The goal of amortization is to reduce the individual cost of these services. Cost reduction increases the potential that the offered services are cheaper and well-within the user's budget. In such a case, the services become competitive and have a high chance of being offered instead of alternative services (for example the faster and the cheaper a query plan is, the most probable it is to be selected for execution, among alternative query plans). The provider benefits from the difference of actual cost and offered user compensation; therefore, the provider increases its credit, which gives the opportunity for more investments directed by regret, leading to even more quality data services.

The framework comprises a prediction model for the amortization of the building and maintenance cost of new data structures to prospective data requests that will be executed using them. If data structures are employed in the service of a lot of prospective data requests, then its building cost can be longer amortized, making individual payments smaller. The opposite holds for structures that are predicted to be used by few prospective queries. The prediction should be ongoing in order to fix errors using feedback from real incoming workload.

There has been notable work on workload prediction in grid and super-computer environments [25, 26, 27, 28]. These works deal with the problem of workload modeling for the production of synthetic workloads employed for the evaluation of real systems. They focus on the site execution locality [25] or the temporal locality [27] of workload characteristics such as runtime, memory and CPU usage etc [26]. These works are orthogonal to the prediction model we need and focus on workload characteristics of the data level. Specific works on modeling job arrivals [28, 29] can be complementary to the discussed prediction model, since they can provide the workload distribution for special grid environments. Structure usage prediction has been studied in other areas, such as the processor cache-line access and survivability prediction [30]. One proposal is a trace-based mechanism that predicts when a cache-line

has been last accessed [31]. Another is a time-based mechanism that predicts the death of a block after a specific timeout period [32]. A third one is a counting-based predictor [33] that predicts the line to be dead after a fixed number of accesses. The prediction model included in the proposed framework should be on the lines of the last one, extended, however, to utilize the computing and storage power available to full processors compared to the limited power on a cache controller.

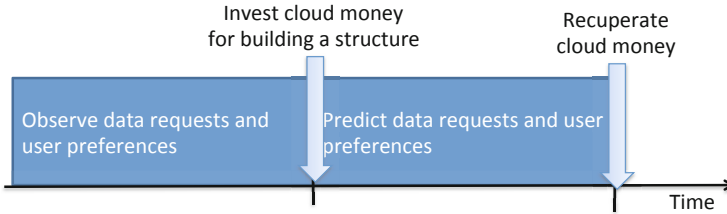


Fig. 8. Observation is followed by prediction of data requests and user preferences

4.4 Pricing Scheme

The cloud provider makes profit from selling its services at a price that is higher than the actual cost. Setting the right price for a service is a non-trivial problem, because when there is competition the demand for services grows inversely but not proportionally to the price. There are two major challenges when trying to define an optimal pricing scheme for the cloud caching service. The first is to define a simplified enough model of the price-demand dependency, to achieve a feasible pricing solution, but not an oversimplified model, which is not representative. For example, a static pricing scheme cannot be optimal if the demand for services has deterministic seasonal fluctuations. The second challenge is to define a pricing scheme that is adaptable to (i) modeling errors, (ii) time-dependent model changes, and (iii) stochastic behavior of the application. For instance, service demand may depend in a non-predictable way from factors external to the cloud application, such as socioeconomic situations.

Pricing schemes were proposed recently for the optimal allocation of grid resources in order to increase revenue [34], or to achieve an equilibrium of grid and user satisfaction [35], assuming knowledge of the demand for resources or the possibility to vary the price of a resource for different users. Similarly, dynamic pricing for web services [36] focuses on scheduling user requests. Moreover, dynamic pricing for the provision of network services [37, 38] aims at achieving a game-theoretic equilibrium through price control among competitive Internet Service Providers. The problem of revenue management through dynamic pricing is well-studied [39]. Based on the rationale that price and demand are dependent qualities, numerous variations of the problem have been tackled, for instance businesses that sell products to retailers [40], seasonal products [41], stochastic demand [42]. Data services necessitate a new dedicated pricing scheme since they are distinguished from consumable products in two major ways: (i) they are not exhausted while they are consumed and (ii) the demand for a specific service pauses while this is not available.

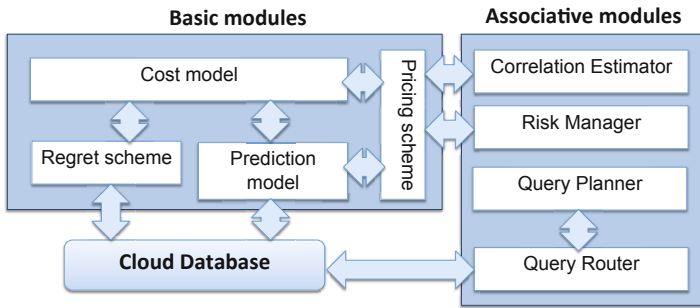


Fig. 9. Modules of the cost-aware data management framework

5 Associative Framework Modules

Section 4 describes the basic modules of the proposed framework in order to achieve cost-aware data management in a cloud data service provider. Beyond the basic modules, the framework should include some associative modules that are necessary in order to make the framework applicable and effective in a real cloud environment:

1. **An estimator of the correlation of offered data services:** All three modules, the regret scheme, the prediction model and the pricing scheme, make decisions about offered data services. These decisions may be ineffective or even wrong if essential correlations between data services are ignored. All three modules could benefit from information on such correlation, input by a separate associative module that is able to estimate it.
2. **A middleware for workload execution:** The basic modules operate and take decisions on a query-per-query basis. Yet, in a real cloud data management environment users would most probably like to receive services for execution of large workloads and not for individual queries. A middleware that takes care of planning the execution of a group or even a workflow of data requests and routes these to the cloud DBMS is necessary.
3. **A risk manager:** All basic modules take decisions based on cost and performance estimations. Naturally, estimations are expected to be wrong occasionally. In such cases, the respective decisions are wrong and have a negative effect to the cloud provider operation, and to its economic viability and profitability. We need to have some knowledge about the extent of such negative effects and relate them with recovery and calibrating procedures. A module that can take all such information into account, and estimate and manipulate the risk they incur for the provider, is necessary in a real cloud environment.

Fig. 9 shows all the modules of the framework.

5.1 Estimator of Data Service Correlation

It is expected that the employment of data structures in the provision of data services will exhibit utility correlations. Therefore the regret scheme but, most importantly, the pricing model should take into account that the data structures may compete or collaborate during query execution. For example, consider the query:

```
select A from T where B = 5 and C = 10
```

Out of the set of candidate indexes to run the query efficiently, indexes $I_b = T(B)$, $I_c = T(C)$, and $I_{bc} = T(BC)$ are most important, since they can satisfy the conditions in the 'where' clause. If the cloud DBMS uses I_{bc} , then the indexes I_b and I_c , will never be used, since I_{bc} can satisfy both conditions. Therefore, the presence of I_{bc} has a negative impact on the demand for I_b and I_c . Alternatively, if the cloud DBMS uses I_b , then I_c can also improve query performance via index intersections, hence increasing the profit for the cloud. Therefore, indexes I_b and I_c have positive impact on each other's demand. Thus, it is necessary to provide a measure that estimates demand correlations among data structures. We form three requirements for this measure.

1. The measure is able to capture both competitiveness and collaborativeness of data structures.
2. The computational complexity is low so that its performance is satisfying even in the presence of a big number of data structures.
3. The measure computes normalized values, in order to provide fair comparison.

Recently, the authors of [43] proposed a technique that computes the correlation between indexes. Yet, this technique does not satisfy any of the above requirements.

Beyond measuring the correlation of data structures, the module could also measure the correlation of the performance of different CPUs or data partitions on the cloud disk. Since data services utilize data structures for accelerating data search, as well as CPU and data partitions for parallelization of execution, we can use the correlation measurements of all the latter in order to classify data structures and parts of the infrastructure. Such classifications can be used to form basic data services and build complex ones on top of them. The correlation of the latter can be, therefore, based on the correlation of basic data services that are involved.

5.2 A Middleware for Workload Execution

In a real cloud environment, we expect users to ask for the service of a workload and not just individual queries. Such a workload can be a group or even a workflow of data requests, in which queries can be interleaved by algorithmic processing. Such a query workflow may represent an experimental analysis on scientific data and may be accompanied by data computing that has to be performed between queries. Thus, it is necessary to provide in the proposed framework a middleware that takes as input a set of queries and routes these queries into the cloud appropriately, in parallel and sequential combinations. The middleware architecture can enable the realization of analysis workflows by providing interaction of queries. The challenge in designing the middleware is to achieve a functionality that resembles that of a declarative

programming language. The analysis workflow should be received by the middleware as a program by an interpreter. The comprised queries would correspond to the parameterized functions of the program that have to be executed in a specific (partial) order.

The middleware includes an overall query planner that schedules the queries involved in a workflow or, even more, in many workflows. The planner has to compute, update and use estimations for the overall cost and performance. Using such estimations, the planner enables efficient parallel processing of independent queries. For this task we can employ experience in job scheduling and query optimization.

5.3 Incorporating Risk Management

All basic framework modules operate on the basis of various estimations. Yet, it is very probable that the real operation of the cloud provider would prove these estimations wrong, in which case the provider runs the risk to become economically non-profitable, and furthermore, non-viable. The uncertainty of the behavior of such an environment comes from several sources: unknown changes in service availability (e.g. data updates), unpredictable extremes of service requests (spikes), and unpredictable failures of cloud hardware. Thus, it is necessary to include in the framework an associative module that handles the risk that originates from these factors. The risk module estimates the risk based on statistics and includes tailored risk management procedures for the rest of the modules. Having a separate module to handle the risk allows the implementation, testing and customization of various risk management techniques, without affecting the design of the basic modules. Achieving the realization of an effective risk module will allow the design of a wide range of SLAs between the users and the cloud data service provider.

6 Current State of Work and Future Plans

In [4] we make an initial proposal of an economic model suitable for a self-tuned cloud cache. We have defined the cloud infrastructure for data that are massively collected and queried, such as scientific data and we have studied the characteristics of query workloads that are amenable to caching, and, therefore, economically suitable to be served by a cloud data management system. Typically, the queries must have two properties: first, they have data access locality, i.e. they mostly target a specific part of the data; second, queries have temporal locality, i.e. similar queries are posed close in time. We make an initial proposal of a cost model that takes into account all possible query and infrastructure expenditure. Our experimental study proves that the proposed solution is on the right track and viable for a variety of workloads and data.

In [5] we present a prediction model that estimates the survivability of the new data structure in the cloud DBMS, which is based on two factors: (i) usefulness of the structure, i.e. the probability that the structure is employed in data services w.r.t. time, and (ii) the stability of the structure, i.e. the probability that data related to the structure are not updated w.r.t. time.

In [6] we present a pricing scheme that is suitable for a cloud cache. The scheme is based on a novel price-demand model designed specifically for a cloud cache. There are two major challenges in designing such model: First, we need a model that represents the reality in a simplified, in order to be computable at runtime with thousands of parameters, but not over-simplified manner, in order to capture the factors that determine the price-demand relation. Together with the pricing scheme we also showed a first effort in designing a correlation estimator for data structures.

We consider all the above work to be the first step towards the exploration of the applicability of the proposed framework in a cloud data service provider. We intend to revisit, extend and generalize the current work so that the modules work for sets or workflows of queries or data requests beyond queries. Also, most of the current work does not take into account correlations of data services and parallelization of execution on many machines. Furthermore, we have not worked yet on the middleware for workload execution and the risk manager. Finally, the above work was produced in parts that were designed and tested separately. We need to go a step further and extend the work so that all modules are units that constitute a black box for the outside environment, yet they collaborate transparently, as in the proposed framework.

Until now our experimental study has been based on real and synthetic datasets and query workloads. The real datasets and workloads were from SDSS 2006 [44]. In the future we intend to perform experimental studies on a diversity of data and queries that will allow for a thorough testing of the framework. Furthermore, we can use these real datasets and workloads in order to produce synthetic ones that will allow for a thorough sensitivity testing. Roughly, synthetic datasets should vary: (i) selectivity, (ii) joins, and (iii) aggregation of query operations; and by (i) varying the value range of existing attributes or entity properties, and (ii) adding more attributes or entities. The general goal of our methodology is to perform thorough sensitivity experimentation on all aspects of the proposed data-aware cloud economy.

7 Conclusion

This paper proposes a generic framework for cost-aware data management in a cloud environment. The framework comprises basic and associative modules that each take care of one aspect of integrating cost-aware and traditional, performance-aware, data management. The modules are designed to give input and output to each other and to the cloud DBMS that executes the user data requests. In this way, each one of them can be realized in a manner that is suitable for the cloud application at hand, without interfering with the realization of the rest. The whole framework can be plugged on top of a cloud DBMS. We have already promising results from a first effort to realize the basic modules of the framework for a cloud cache.

References

1. <http://aws.amazon.com/ec2>
2. <http://www.gogrid.com>

3. <http://code.google.com/appengine>
4. Dash, D., Kantere, V., Ailamaki, A.: An Economic Model for Self-Tuned Cloud Caching. In: ICDE, pp. 1687–1693 (2009)
5. Kantere, V., Dash, D., Gratsias, G., Ailamaki, A.: Predicting cost amortization for query services. In: ACM SIGMOD, pp. 325–336 (2011)
6. Kantere, V., Dash, D., Francois, G., Kyriakopoulou, S., Ailamaki, A.: Optimal Pricing for a Cloud Cache. The IEEE TKDE, Special Issue on Cloud Data Management 23(6), 1345–1358 (2011)
7. Ramaswamy, L., Liu, L., Iyengar, A.: Cache clouds: Cooperative caching of dynamic documents in edge networks. In: ICDCS, pp. 229–238 (2005)
8. Ramaswamy, L., Liu, L., Iyengar, A.: Scalable delivery of dynamic content using a cooperative edge cache grid. IEEE TKDE 19(5) (2007)
9. Bhattacharjee, S., Calvert, K.L., Zegura, E.W.: Self-organizing wide-area network caches. In: IEEE Infocom, pp. 752–757 (1998)
10. Malik, T., Burns, R.C., Chaudhary, A.: Bypass caching: Making scientific databases good network citizens. In: ICDE, pp. 94–105 (2005)
11. Wang, X., Burns, R.C., Terzis, A., Deshpande, A.: Network-aware join processing in global-scale database federations. In: ICDE, pp. 586–595 (2008)
12. Foster, I.: What is the grid? a three point checklist (2002), <http://www-fp.mcs.anl.gov/foster/articles/whatisthegrid.pdf>
13. Nieto-Santisteban, M.A., Gray, J., Szalay, A.S., Annis, J., Thakar, A.R., Omullane, W.J.: When database systems meet the grid. In: CIDR, pp. 154–161 (2005)
14. Watson, P.: Databases and the grid. Grid Computing: Making The Global Infrastructure a Reality, Technical Report (2001)
15. Stonebraker, M., Aoki, P.M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A.: Mariposa: A wide-area distributed database system. VLDB J. 5(1) (1996)
16. Wellman, M.P., Walsh, W.E., Wurman, P.R., Mackie-Mason, J.K.: Auction protocols for decentralized scheduling. Games and Economic Behavior 35, 2001 (1998)
17. Ernemann, C., Hamscher, V., Yahyapour, R.: Economic Scheduling in Grid Computing. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 128–152. Springer, Heidelberg (2002)
18. Moreno, R., Alonso-Conde, A.B.: Job Scheduling and Resource Management Techniques in Economic Grid Environments. In: Fernández Rivera, F., Bubak, M., Gómez Tato, A., Doallo, R. (eds.) Across Grids 2003. LNCS, vol. 2970, pp. 25–32. Springer, Heidelberg (2004)
19. Kradolfer, M., Tombros, D.: Market-based workflow management. International Journal of Cooperative Information Systems 7 (1998)
20. Chen, C., Maheswaran, M., Toulouse, M.: Supporting co-allocation in an auctioning-based resource allocator for grid systems. In: IPDPS, pp. 89–96 (2002)
21. The Office of Science Data-Management Challenge. Report from the DOE Office of Science Data-Management Workshops (March-May 2004)
22. Ailamaki, A., Kantere, V., Dash, D.: Managing scientific data. Communications of ACM 53(6), 68–78 (2010)
23. Gray, J., Szalay, A.S., Thakar, A., Stoughton, C., van Berg, J.: Online Scientific Data Curation, Publication, and Archiving. CoRR cs.DL/0208012 (2002)
24. Gray, J., Liu, D.T., Nieto-Santisteban, M.A., Szalay, A.S., DeWitt, D.J., Heber, G.: Scientific Data Management in the Coming Decade. CoRR abs/cs/0502008 (2005)
25. Feitelson, D.G.: Locality of sampling and diversity in parallel system workloads. In: ICS, pp. 53–63 (2007)

26. Li, H., Groep, D.L., Wolters, L.: Workload Characteristics of a Multi-cluster Supercomputer. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 176–193. Springer, Heidelberg (2005)
27. Li, H., Muskulus, M., Wolters, L.: Modeling correlated workloads by combining model based clustering and a localized sampling algorithm. In: ICS, pp. 64–72 (2007)
28. Minh, T.N., Wolters, L.: Modeling Parallel System Workloads with Temporal Locality. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2009. LNCS, vol. 5798, pp. 101–115. Springer, Heidelberg (2009)
29. Minh, T.N., Wolters, L.: Modeling job arrival process with long range dependence and burstiness characteristics. In: CCGRID, pp. 324–330 (2009)
30. Calder, B., Grunwald, D.: Next cache line and set prediction. In: ISCA, pp. 287–296 (1995)
31. Lai, A.-C., Fide, C., Falsafi, B.: Dead-block prediction & dead-block correlating prefetchers. In: ISCA, pp. 144–154 (2001)
32. Hu, Z., Kaxiras, S., Martonosi, M.: Timekeeping in the memory system: predicting and optimizing memory behavior. In: ISCA, pp. 209–220 (2002)
33. Kharbutli, M., Solihin, Y.: Counter-based cache replacement and bypassing algorithms. *IEEE Transactions in Computing* 57(4), 433–447 (2008)
34. Sulistio, A., Kyong Hoon, K., Buyya, R.: Using revenue management to determine pricing of reservations. In: *IEEE e-Science*, pp. 396–405 (2007)
35. Allenor, D., Thulasiram, R.K., Thulasiraman, P.: A Financial Option Based Grid Resources Pricing Model: Towards an Equilibrium between Service Quality for User and Profitability for Service Providers. In: Abdennadher, N., Petcu, D. (eds.) GPC 2009. LNCS, vol. 5529, pp. 13–24. Springer, Heidelberg (2009)
36. Lin, Z., Ramanathan, S., Zhao, H.: Usage-based dynamic pricing of Web services for optimizing resource allocation. *Inf. Systems and E-Business Management* 3(3), 221–242 (2005)
37. Masuda, Y., Whang, S.: Dynamic Pricing for Network Service: Equilibrium and Stability. *Management Science* 45(6), 857–869 (1999)
38. Cao, X.-R., Shen, H.-X., Milito, R., Wirth, P.: Internet pricing with a game theoretical approach: concepts and examples. *ACM Transactions on Networking* 10(2), 208–216 (2007)
39. Bitran, G.R., Caldentey, R.: An overview of pricing models for revenue management. *MSOM* 5(3), 203–229 (2003)
40. Ghose, A., Choudhary, V., Mukhopadhyay, T., Rajan, U.: Dynamic pricing: A strategic advantage for electronic retailers. In: ICIS, p. 28 (2003)
41. You, P.-S., Chen, T.C.: Dynamic pricing of seasonal goods with spot and forward purchase demands. *Comput. Math. Appl.* 54(4), 490–498 (2007)
42. Gallego, G., van Ryzin, G.: Optimal Dynamic Pricing of Inventories with Stochastic Demand over Finite Horizons. *Management Science* 40(8), 999–1020 (1994)
43. Schnaitter, K., Polyzotis, N., Getoor, L.: Modeling index interactions. In: VLDB, pp. 1234–1245 (2009)
44. <http://www.sdss.org>

Event-Driven Actors for Supporting Flexibility and Scalability in Service-Based Integration Architecture

Huy Tran and Uwe Zdun

Software Architecture Research Group
University of Vienna, Austria
firstname.lastname@unvie.ac.at

Abstract. Service-based software systems are often built by incorporating functionalities from other software systems or platforms. A widely used approach in practice is to introduce an intermediate integration layer for hiding the complexity and heterogeneity of the integrated systems or platforms. However, existing approaches introduce limited support for the flexibility of the integration architecture. It is challenging to alter the integration architecture, e.g., due to some exceptions or unanticipated situations such as peak loads or emergencies, because of rigid dependency structures in the integration architecture defined at design or deployment time. In this paper, we propose DERA as a novel approach that exploits event-driven architecture concepts for enhancing the flexibility and scalability of service-based integration architectures. Our approach provides primitive concepts that can easily be analyzed with tools or be used to depict a current snapshot of the integration architecture using graphical notations close to the intuitive perception of stakeholders. We show the applicability of DERA through an industrial case study in the field of software platform integration and evaluate the scalability of our approach.

Keywords: Event-driven architecture, event actors, services, service-based integration architecture, flexibility, scalability, substitutability.

1 Introduction

Service-oriented architectures (SOA) provide efficient means for exposing functionality of software systems or platforms in terms of services with standardized interfaces. Service-based systems can be built by incorporating services provided by other systems or platforms. Instead of directly dealing with the heterogeneity and variety of the integrated systems or platforms, an intermediate integration layer is often introduced for bringing systems or platforms into the service-oriented world and providing the required service integration and composition logic [15]. Figure 1 illustrates a high-level overview of a simplistic platform integration layer design. There is a considerable amount of existing approaches for realizing the integration layer, such as using composite services, enterprise service buses, messaging infrastructures, or business processes.

A certain flexibility is often required at the level of the integration layer to not only support rapid tailoring and customization of the integration architecture but also enable the ability to deal with some exceptions or unanticipated situations such as peak loads

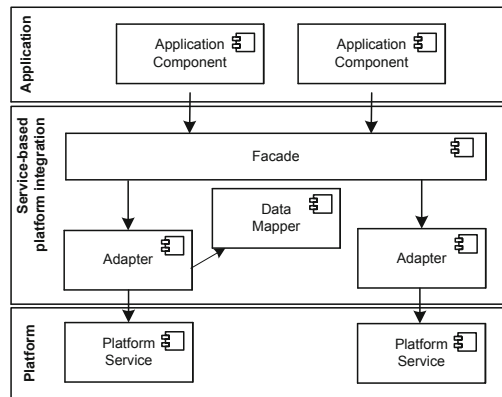


Fig. 1. Overview example for service-based platform integration architectures

or emergencies. The mentioned existing approaches introduce only limited support for the flexibility of the integration architecture through dynamic bindings of the constituting elements (e.g., dependency injection or aspect weaving techniques are used in some of the mentioned approaches) [7,9,8]. Altering the integration architecture is very challenging because of the rigid dependency structures in the integration architecture defined at design or deployment time. Such rigid dependencies also might cause scalability problems as sophisticated algorithms and coordination techniques are required to efficiently schedule and distribute integration and composition logic [14,4].

Event-driven communication styles are potential solutions for facilitating high flexibility, scalability, and concurrency of distributed systems [18]. Due to the intrinsic loose coupling of the participants, the event-based architectural style is used in many large-scale distributed software systems today. The advantages of event-driven communication styles have been extensively exploited in a considerable amount of work proposing different combinations of event-driven architectures (EDA) with business process management and SOAs [12,25]. To the best of our knowledge, none of the existing approaches has exploited EDA for resolving the aforementioned problems in service-based integration architectures. Another issue hindering the use of EDA is that software architects and developers often find the event-driven communication style unintuitive (compared to other paradigms such as remote procedure calls or messaging) and large architectures with many event actors and numerous events hard to comprehend.

In this paper, we propose DERA as a novel approach leveraging the event-driven architecture style to enable flexibility of integration architectures for supporting various kinds of runtime evolution and dynamic adaptation while minimizing the non-deterministic nature of event-based applications. In particular, our approach encapsulates constituting elements (e.g., components, functions, adapters, proxies) using event actors with explicit interfaces. It exploits the event-based communication style to loosen the dependencies among the actors. The interfaces of the actors, described in terms of incoming and outgoing events, are formally specified and constrained to, on the one hand, enable the support for changing actors at runtime (e.g., replacing actors or changing their execution order). On the other hand, well-defined interfaces

of actors also reduce the non-deterministic behavior in EDAs. The proposed graphical notations of the DERA concepts can be used to visually depict a current snapshot of the event-driven service integration architecture that is closer to the stakeholders' perception than studying only the event actors' inputs and outputs and their event processing rules. In this paper, we present concepts and formalizations to establish the foundation of our approach as well as a prototypical implementation. We show the applicability of our approach using an industrial case study in the field of service platform integration. Finally, we evaluate the performance and scalability of our approach, as good scalability is one of the central promises of EDAs and a central integration layer is a potential bottleneck in an integration architecture. We can show that our approach offers linear scalability and has only a moderate performance overhead compared to a hard-coded Java implementation with rigid dependencies among the actors.

The paper is structured as follows. Section 2 introduces the case study. We present the DERA (Dynamic Event-driven Actors) approach in Section 3. In Section 4, we describe a prototypical implementation of DERA, revisit the case study and elaborate on how DERA can help to improve the flexibility in the case study, and report on the evaluation of DERA's performance and scalability. We compare to related work in Section 5, and finally we conclude and discuss future work in Section 6.

2 Case Study

We illustrate the application of the concepts presented in this paper in an integration scenario of a warehouse operator application extracted from an industrial case study in the field of service platform integration¹. In the context of this application, there are three different domain-specific service platforms, namely, a yard management system (YMS), a warehouse management system (WMS), and a remote monitoring system (RMS) that provide a wide variety of services. The warehouse operator application shall utilize these services to perform various necessary tasks that are triggered by events such as the arrival of a truck carrying products to be stored in the warehouse. The three platforms shall be integrated using an service-based platform integration layer akin to the one in Figure 1, and the warehouse operator application shall use the services of the integration layer to access the services of all three platforms.

We present in Figure 2 a schematic view of one scenario of the warehouse operator application. The lifelines of the sequence diagram represent the proxy components which are responsible for interacting with the service platforms involved in the scenarios. One of the crucial requirements for the scenario is that we want to be able to alter the composition logic of the warehouse operator in a flexible manner at runtime. For instance, the warehouse operator might not need to perform `getFreeDock` because the arriving truck is specially assigned to a dedicated dock in the yard. Another example is that the warehouse operators need to call the warehouse staff to prepare for unloading products in the truck. We will revisit and analyze this scenario in Section 4.

¹ http://indenica.eu/fileadmin/INDENICA/user_upload/d51-casestud.pdf

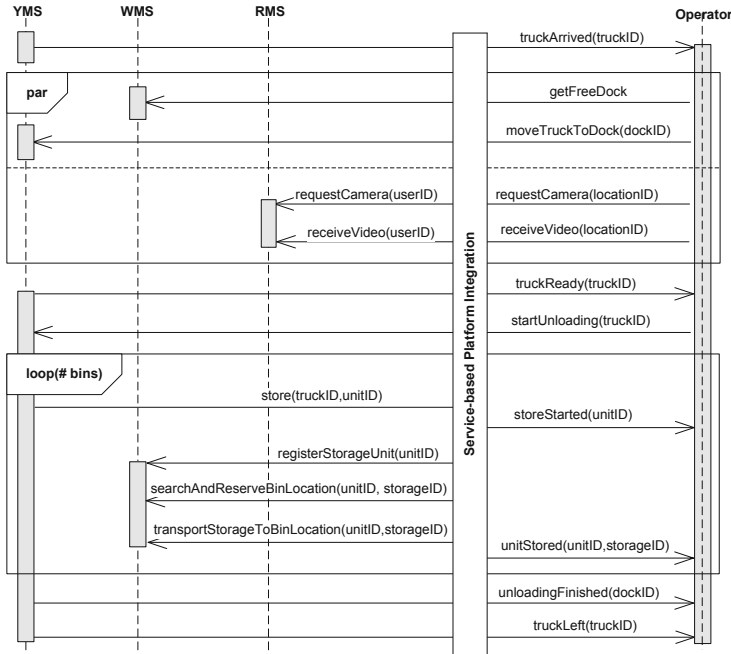


Fig. 2. The behavior of the unloading scenario

3 Dynamic Event-Driven Actors (DERA)

3.1 DERA Primitive Concepts

Figure 3 depicts the meta-model of the primitive concepts forming a DERA system. The semantics of these basic event actors are provided in Table 1. The central notions of DERA are *events* and *event actors*. An event can be considered essentially as “any happening of interest that can be observed from within a computer” [20] (or a software system). An example of an event from the business domain is the arrival of a purchase order. An event is associated with one or more `ObjectReferences` for obtaining references to data sources managed in `ObjectPools`. Events may also have some attributes such as their unique identifier, correlation identifiers, timing attributes, and so on.

An *event type* is a representation of a class of events that share a common set of attributes. An *instance* of an event type is a concrete occurrence of that event type that has a unique identifier and is instantiated with concrete values of the event type’s attributes. An example of an event type are incoming customer orders, whilst the corresponding instance of this type is an incoming order from Alice. Given a concrete event e , `typeof(e)` will be used to denote the event type of e .

The encapsulation of a particular computational unit (e.g., an executable function, a component, a proxy, or an adapter) that performs a concrete task, for instance, executing composition logic of a number of service invocations, performing the role of a service

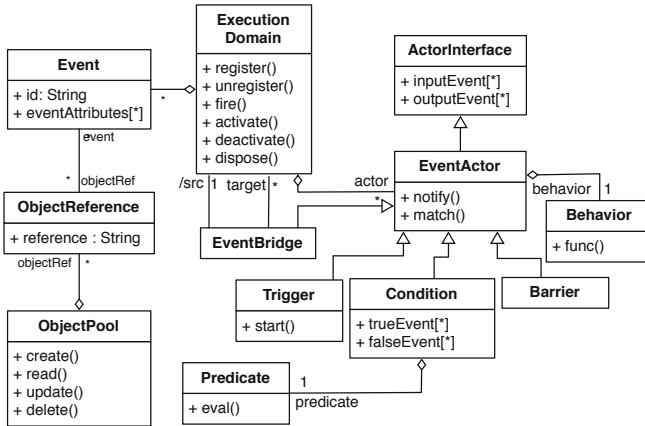


Fig. 3. Meta-model for the primitive DERA concepts and their relationships

adapter or proxy, accessing and transforming data, to name but a few, is an *event actor* (or *actor* for short). Each actor has a well-defined interface represented by the ActorInterface class. The actor’s interface defines a set of events that the actor awaits (aka input events) and a set of events that the actor will emit after finishing its execution (aka output events). Particular behavior of each actor is defined through a concrete instance of the Behavior class. The execution of an actor is triggered by the arrival of any of the input events. At the end of its execution, the actor will emit *all* of its output events that, in turn, may trigger the executions of other actors.


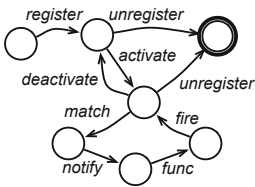

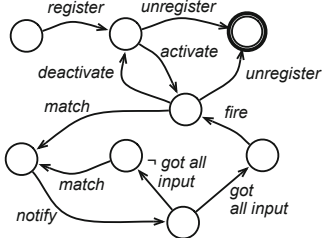

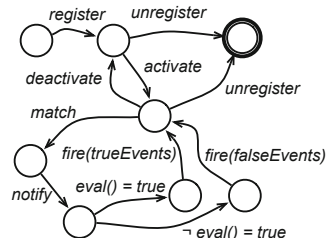

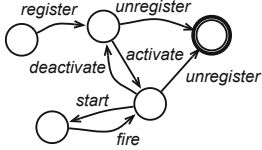

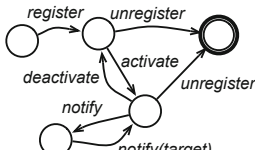
In Table 1, we present the graphical notations of the DERA actors that can be used to visually depict a snapshot of a DERA system at a particular point in time. A formal behavior definition for each type of actor is given as a Finite State Machine (FSM). The transitions between two states represents the occurrence of DERA operation invocations such as (un)register, (de)activate, notify, match, func, fire, and so forth, (see Definition 2 below). Any snapshot of a DERA system can easily be transformed into an FSM or another formal representation (like a Petri Net) to perform formal analysis of the system snapshot. The DERA prototype is based on Java. Developing DERA applications using the Java APIs is rather tedious as they offer a lower abstraction level than DERA system models. For this reason, we additionally provide DeraDSL, a domain-specific language (DSL) for supporting the model-driven specification of DERA systems. The second column of Table 1 shows the code required for defining each of the actor type in DeraDSL. Note that DeraDSL’s constructs will be mapped to the meta-model depicted in Figure 3 and consequently to the Java implementation of DERA for execution. We explain DeraDSL in detail in Section 4.1.

Event actors are defined based on well-defined event interfaces:

Definition 1 (Event actor interface). An interface \mathcal{I}_x of a DERA event actor x can be described by a 2-tuple $(\bullet x, x\bullet)$, where $\bullet x$ is a set of input events expected by x and $x\bullet$ is a set of output events to be emitted by x ($\bullet x$ and $x\bullet$ can be empty sets).

The benefits of specifying well-defined interfaces for DERA event actors are manifold. Firstly, they enable us to *conceptually* capture a snapshot of current state of the DERA

Table 1. Notations and formal definitions of behaviors of DERA actors

Actor	Notation	DeraDSL construct	Formal behavior definition
EventActor		EventActor <name> <input [inputevents]<br=""/> output [outputEvents] func [actor-behavior]	
Barrier		Barrier <name> <input [inputevents]<br=""/> output [outputEvents]	
Condition		Condition <name> <input [inputevents]<br=""/> when-true [trueEvents] when-false [falseEvents] eval [predicate]	
Trigger		Trigger <name> output [outputEvents]	
EventBridge		EventBridge <name> target [targetDomains]	

system and derive a directed graph that comprises event actors connected via their inputs and output events at design time or runtime. As a result, we are able to support monitoring and analysis of important properties such as reachability (safety or deadlock checking), liveness, performance, quality of services, of DERA systems described in terms of such graphs. On the other hand, well-defined interfaces also enable us to support changes at runtime, e.g., substituting an event actor by another with a compatible interface or changing the execution order of event actors by substituting an event actor by another having the same input events but different output events.

3.2 DERA Architecture

Figure 4 shows an overview of the DERA tool-chain on the left-hand side and the DERA runtime architecture on the right-hand side. DeraDSL will be described in Section 4.1. In this section, we focus on the DERA runtime architecture.

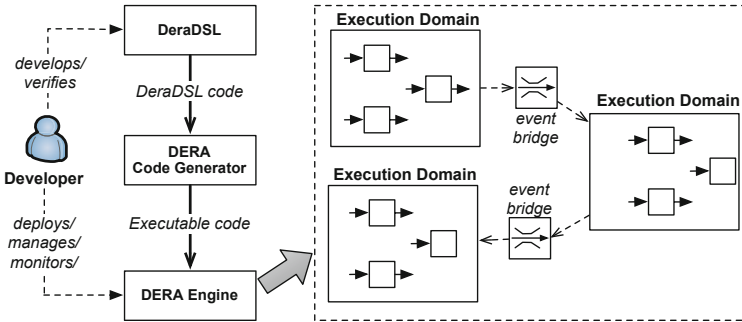


Fig. 4. Overview of DERA development toolchain and system architecture

DERA is designed so that each event actor only concentrates on its own task, its well-defined interfaces defining its input events and the events it is going to emit. There are no tight dependencies between two particular event actors except “*virtual connections*” established via the event-based communications. This is realized using the notion of *event channels* which are abstractions used for delivering events among communicating parties. All actors in an execution domain of a DERA system are connected to the same channel, and all events are published via the channel. All events published on a channel are consumed by all actors registered for the channel. Hence, actors are loosely coupled. This loose coupling leads to the flexibility and scalability of DERA. That is, event actors can be distributed for better load balancing or performance optimization purposes without requiring any sophisticated distribution algorithms.

Event channels are also used as a means to implement logical *execution domains*. Execution domains are useful for supporting runtime governance activities such as deployment, management, and monitoring, as they can be used to group event actors and events. An execution domain might host one or many DERA applications while a certain DERA application can span across several execution domains. Two DERA execution domains can be connected by EventBridges which are special event actors

responsible for forwarding events from a DERA execution domain to another (see Figure 4). The original function of event bridges may also be extended with extra features such as enriching or transforming the content of events.

Definition 2 (DERA system). A DERA system \mathcal{S} can be described by a 4-tuple $(\mathcal{E}, \mathcal{A}, \mathcal{C}, \mathcal{O})$, where

1. \mathcal{E} is a finite set of events.
2. \mathcal{A} is a finite set of event actors. Each event actor $x \in \mathcal{A}$ has a well-defined interface $\mathcal{I}_x(\bullet x, x\bullet)$, where $\bullet x \subseteq \mathcal{E}$ and $x\bullet \subseteq \mathcal{E}$ and can perform a particular function.
3. \mathcal{C} is an event channel that is responsible for delivering an event received from a certain event actor exactly once to all other actors registered to the channel. An event actor $x \in \mathcal{A}$ consumes an incoming event $e \in \mathcal{E}$ from the channel \mathcal{C} only if $\text{match}(x, e) = \text{true}$ (for definition of $\text{match}()$ see below). The channel is assumed to be reliable, i.e., no message is lost or altered.
4. \mathcal{O} is a set of basic operations, including (but not limited to)
 - $\text{register}(x)$, where $x \in \mathcal{A}$, indicates that the actor x is registered to the event channel \mathcal{C} .
 - $\text{unregister}(x)$, where $x \in \mathcal{A}$, indicates that the actor x unregisters to the event channel \mathcal{C} .
 - $\text{fire}(x, E)$ or $\text{fire}(x, e)$, where $x \in \mathcal{A}$, $E \subseteq \mathcal{E}$, and $e \in \mathcal{E}$, indicates that the actor x fires a set of events E or a single event e , respectively.
 - $\text{match}(x, e)$, where $x \in \mathcal{A}$, $e \in \bullet x$, returns true if the event e matches the interface of the actor x and false otherwise.
 - $\text{func}(x)$, where $x \in \mathcal{A}$, denotes a concrete task or an arbitrary user-defined behavior of the event actor x .
 - $\text{deactivate}(x)$ and $\text{activate}(x)$, where $x \in \mathcal{A}$: $\text{deactivate}(x)$ is used for putting the execution of x on hold and $\text{activate}(x)$ is used for resuming its execution, for instance, after a $\text{deactivate}(x)$.

The interface operations $\text{register}()$ and $\text{unregister}()$ are mainly used for the management of a DERA execution domain, and $\text{deactivate}()$ and $\text{activate}()$ deal with lifecycle management. The operation $\text{func}()$ is the placeholder where one can put in a certain user-defined behavior such as invoking a service, accessing a database, or opening and reading a local file. Please note that the execution of the operation $\text{func}()$ is not allowed to change the event actor's interface (i.e., input and output events) or emit new events. The main goal of this constraint is to reduce the non-deterministic nature of the event-based communication styles employed in DERA. Changing an event actor's input and output events must be explicitly declared through its interface. As a result, we can *conceptually* establish the dependencies between event actors by observing their interface descriptions. The dependencies can be used for many important tasks such as monitoring and verifying properties of DERA systems and applications.

Definition 3 (Event matching). Given a DERA system \mathcal{S} described by the 4-tuple $(\mathcal{E}, \mathcal{A}, \mathcal{C}, \mathcal{O})$. Let x be an event actor ($x \in \mathcal{A}$) having an interface $\mathcal{I}_x = (\bullet x, x\bullet)$. An event $e_1 \in \mathcal{E}$ matches the interface \mathcal{I}_x if and only if there exists at least one event $e_2 \in \bullet x$ such that e_1 and e_2 are of the same event type (i.e., $\text{typeof}(e_1) = \text{typeof}(e_2)$).

Definition 4 (DERA application). A DERA application Φ running in a DERA system S can be described by a 4-tuple $(A, E, E_{start}, E_{finish})$, where

- $A \subseteq \mathcal{A}$ is a finite set of event actors constituting the functionality of the application;
- $E \subseteq \mathcal{E}$ is a finite set of events published and consumed by the event actors of A ;
- $E_{start} \subseteq \mathcal{E}$ is a finite set of events that indicate the start of the application;
- $E_{finish} \subseteq \mathcal{E}$ is a finite set of events that indicate the end of the application.

```

module eu.indenica.casestudy.yms
domain YMS {
  Trigger y1 output [ymsTruckArrived]
  EventActor y2 input [facadeMoveTruckToDock] output [ymsMoveTruckToDockFinished]
    func [MoveTruck]
  EventActor y3 input [ymsMoveTruckToDockFinished] output [ymsTruckReady]
    func [CheckTruckReadyForUnloading]
  Barrier y4 input [ymsTruckReady, facadeStartUnloading] output [ymsStartedUnloading]
  EventActor y5 input [ymsStartedUnloading, ymsUnloadingNotFinished] output [ymsStore]
    func [StoreUnit]
  Condition y6 input [ymsStore] when-true [ymsUnloadingFinished]
    when-false [ymsUnloadingNotFinished]
  EventActor y7 input [ymsUnloadingFinished] output [ymsTruckLeft] func [CheckTruckInDock]
  Application YMS start-with [ymsTruckArrived] end-with [ymsTruckLeft]
}

```

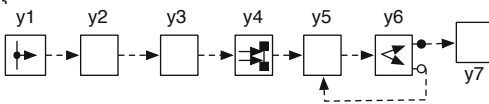


Fig. 5. DERA application and its corresponding graphical representation

We present in Figure 5 an excerpt of the composition logic of the proxy component of the integration architecture that is responsible for interacting with the YMS platform in the scenario shown in Figure 2. The proxy component is described using the programming constructs provided by DeraDSL. Given the specification of the actors and their interfaces, we can build an intuitive graphical representation of a snapshot of the application using the notation from Table 1. Note that the dashed lines among event actors are not real dependencies but virtual connections achieved by analyzing the inputs and outputs of the actors captured in the snapshot of the application.

3.3 Event Actor Substitution

There are several studies in programming languages (especially object-oriented programming) and component-based systems, on the substitutability of data types, objects, and components [22,17]. The event actor substitution in approach is based on the well-known Liskov substitution principle [17]. There are three crucial features of DERA actors that enable us to substitute a certain actor at runtime. Firstly, each actor x explicitly exposes a well-defined event-based interface $\mathcal{I}_x(\bullet, x, x\bullet)$. Secondly, the interactions among actors are loosely coupled through event-based communication. Thirdly, the encapsulation of a computational unit that performs a concrete task (i.e., the operation $\text{func}()$ of an `EventActor`) is not allowed to alter the interface. These features allow us to substitute an actor by one or a set of other actors that introduce (1) an interface which is *compatible* to the original actor's interface (called strong substitution below) or (2) a different interface (called weak substitution below).

Strong substitution occurs when the developers have to replace or upgrade an existing functionality with a different (e.g., better or bug-fixed) version while preserving the overall structure and behavior of the DERA application. It is achieved by defining a new actor y with the same interface as x . x can be replaced with y using the operation `deactivate(x)` to temporarily put the execution of x on hold, and using the operations `register(y)` and `activate(y)` to enable the execution of y .

Definition 5 (Strong substitution). *Let x be an event actor having an interface $\mathcal{I}_x(\bullet x, x\bullet)$. An event actor y posing an interface $\mathcal{I}_y(\bullet y, y\bullet)$ is said to be a strong substitution for x if all the following constraints are satisfied:*

1. *Type requirement: y must be of the same type or a subtype of x . That is, if x is of type `EventActor`, then y must be of type `EventActor` or a subtype of `EventActor`.*
2. *Input requirement (applied for every event actor x , such that $\bullet x \neq \emptyset$): $\bullet y = \bullet x$, i.e., y has to be able to accept the same input events as x does.*
3. *Output requirement (applied for every event actor x , such that $x\bullet \neq \emptyset$): $y\bullet = x\bullet$, i.e., y has to fire the same output events as x does.*

Weak substitution occurs when the developers want to alter the structure and behavior of one or all running instances of a DERA application, for instance, skipping some tasks to deal with exceptions or unanticipated circumstances such as peak loads and emergencies or adding new functionalities. This is difficult to achieve with many existing integration architectures due to rigid dependency structures. We can support the required flexibility in DERA by substituting existing event actors with newly defined event actors posing different interfaces. This kind of substitution is called *weak substitution* as it is not going to preserve the original structure and behavior.

Definition 6 (Weak substitution). *In a DERA application $\Phi = (A, E, E_{start}, E_{finish})$, let $x \in A$ be an event actor having an interface $\mathcal{I}_x(\bullet x, x\bullet)$. A weak substitution y for x can be achieved by relaxing one or more of the conditions for strong substitutions.*

To illustrate possible relaxations of strong substitution conditions, let us consider the following examples:

1. *Type requirement: x and y can be instances of a) the same or b) different types.*
2. *Input requirement (applied for every event actor x , such that $\bullet x \neq \emptyset$): there are no constraints on the input, but a potential case may be one of the following:*
 - a) $\bullet x \subseteq \bullet y$, i.e., y is able to be triggered by more input events than x ,
 - b) $\bullet y = \bullet x \cap (\bigcup z\bullet, \forall z \in A \wedge z \neq x)$, i.e., y only considers to accept a subset of x 's input events that are going to be emitted by other event actors,
 - c) $\bullet x \cap \bullet y = \emptyset$
3. *Output requirement (applied for every event actor x , such that $x\bullet \neq \emptyset$): there are no constraints on the output, but a potential case may be one of the following:*
 - a) $x\bullet \subseteq y\bullet$, i.e., y can emit more events than x .
 - b) $y\bullet = x\bullet \cap (\bigcup \bullet z, \forall z \in A \wedge z \neq x)$, i.e., y only considers to fire a subset of x 's output events that are going to be consumed by other event actors.
 - c) $x\bullet \cap y\bullet = \emptyset$

Weak substitutions lead to different levels of changes ranging from light or moderate adjustments (e.g., 2(a), 2(b), 3(a), and 3(b)) to significant and disruptive alterations (e.g., 2(c) and 3(c)) of event actors' interfaces. In some situations, these changes may become undesirable as they can result in anomalies such as dead tasks, deadlocks, or livelocks. It would be unrealistic to require the developers to ensure that a certain substitution must lead to an expected and sound state of the running DERA applications. Instead, exploiting powerful reasoning mechanisms based on formal methods such as process algebras [16,19] or Petri-nets [21] can help developers to analyze a certain runtime snapshot of DERA applications to detect potential flaws and correct the substitution before applying it. Also, existing approaches on behavior inheritance [5] or on using change patterns for preserving certain system properties [24] can be leveraged in the context of DERA to enhance the soundness of actor substitutions. Studying the applicability of those mechanisms is one of our planned future works.

4 Implementation – Case Study Revisited – Evaluation

4.1 DERA Implementation

A prototypical implementation of the DERA concepts has been developed to show the feasibility of our approach and implement the case study presented in the next section. In our implementation, we have defined an abstraction layer covering all primitive concepts of DERA presented in Figure 3. This layer is independent from the underlying technologies. The implementation layer realizes the concepts of the abstraction layer using a particular technology, in our case the Java Concurrency Utilities packages². These packages, which are available in Java JDK 1.5 and later, provide sufficient concurrency utilities. In particular, in our DERA implementation, the event-driven communication style is realized using asynchronous event callbacks. The creation and execution of event actors are managed using the built-in `ExecutorService` with fixed thread pools and the synchronization among actors is done through the `synchronized()` construct.

The DERA `EventChannels` represent a means for delivering events among communication parties and can be realized by asynchronous communication styles. Thus, it is possible to incorporate existing powerful libraries and frameworks such as Java Message Service³ or PADRES [11] in the DERA implementation. For the evaluation purpose presented in the next section, we opted for implementing DERA `EventChannels` based on pure Java asynchronous multi-threading. Each `EventChannel` has a number of event distributors mapped to a pool of light-weight Java threads. These event distributors receive and deliver events following a round-robin scheduling strategy.

The DERA `EventBridges` used to enable DERA systems to support distributed event processing (see Figure 4) have been realized using REST Web services. REST services are a good match for DERA's flexible and scalable architecture because of the use of the scalable and stateless concepts of the Web in the REST architectural style. In the DERA implementation, the REST services are used for transmitting events between two (possible distributed) execution domain *A* and *B* that are connected via an

² <http://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/index.html>

³ <http://jcp.org/en/jsr/detail?id=343>

`EventBridge`. The `EventBridge` is realized using a REST Service that is connected as an event actor to domain *B*. An event actor on domain *A* acts as a REST client forwarding all events raised within *A* to the REST service. The REST service delivers all received events to *B*. A bidirectional bridge can be established by adding a second REST service to the event bridge actor on channel *A*.

As mentioned before, `DeraDSL` has been developed aiming at minimizing the noise of Java language structures and supporting efficient DERA application development. We leverage `Xtext`⁴, which is a powerful framework supporting textual language development with several advanced features, for instance, syntax coloring, code completion, validation, excellent integration with Java, and many others, to implement `DeraDSL`. Furthermore, combining `Xtext` with `Xtend`⁵ helps us on mapping `DeraDSL`'s elements onto the Java-based constructs and libraries that implement DERA. We partially described some primary elements of `DeraDSL` in Table 1 and used them to illustrate the development of DERA applications in Figure 5.

4.2 Case Study Revisited

Developing the warehouse operator application, motivated in Section 2, using the DERA prototype is fairly straightforward. First, we need to define an `EventActor` for encapsulating each task to be carried out. Then, we need to assign the input and output events of that actor. A `Barrier` may be necessary when we need to wait for more than one event arriving before some other tasks can be performed. For decisions, such as checking if there are enough storage locations in the warehouse to store all units, `Condition` is used. We show an excerpt of the `DeraDSL` code defining the warehouse operator composition logic using DERA actors in Listing 1.1. In this excerpt, the concrete definitions of the operation `func()` of the event actors are in a separate modules and can be cross-referenced. Thus, these functions are omitted in the code. The management operations such as `(de)register` and `(de)activate` are also not visible but we will discuss them in the next section. The warehouse operator requires some events from the integration facade, namely, `PlatformFacade`, defined in the lower part of the code excerpt. Thus, we create an event bridge named `OperatorToFacade` for delivering events from the `WarehouseOperator` domain to the `PlatformFacade` domain. Likewise, the event bridge `FacadeToOperator` is defined in the `PlatformFacade` domain for sending events back.

Even though `DeraDSL` can help on better formulating DERA elements, the code shown in Listing 1.1 is still not close to the developers' perception. At a certain stage, for instance, after finishing development or before deploying, a snapshot of a DERA system and application can be taken and visually depicted using the graphical notations described in Table 1. Accordingly, we can establish an equivalent intuitive graphical representation, as shown in Figure 6, of the aforementioned code. The events from the `PlatformFacade` domain to the `WarehouseOperator` domain are shown for illustrating the relationship between two domains. We can see that the semantics of DERA concepts and notations are close to that of traditional conditional structures in existing

⁴ <http://xtext.org>

⁵ <http://xtend-lang.org>

programming languages or widely-used formal models such as such as process algebras [16,19] or Petri-nets [21]. As a result, existing formal analysis techniques can be leveraged (see Section for details).

```

module eu.indenica.casestudy.warehouse.operator
domain WarehouseOperator {
  EventActor TruckArrivedNotified input [facadeTruckArrived] output [
    operatorTruckArrivedNotified]
  EventActor GetFreeDock input [operatorTruckArrivedNotified] output [operatorGetFreeDock]
  Barrier b1 input [facadeGetFreeDockFinished, operatorGetFreeDock] output [
    operatorGetFreeDockFinished]
  EventActor MoveTruckToDock input [facadeGetFreeDockFinished] output [
    operatorMoveTruckToDock]
  EventActor RequestCamera input [operatorTruckArrivedNotified] output [
    operatorRequestCamera]
  Barrier b2 input [operatorRequestCamera, facadeRequestCameraFinished] output [
    operatorRequestCameraFinished]
  EventActor VideoReceiving input [operatorRequestCameraFinished] output [
    operatorReceiveVideo]
  Barrier b3 input [operatorMoveTruckToDock, facadeMoveTruckToDockFinished,
    operatorReceiveVideo, facadeReceiveVideoFinished, facadeTruckReady] output [
    operatorTruckReadyNotified]
  EventActor StartUnloading input [operatorTruckReadyNotified] output [
    operatorStartUnloading]
  Barrier b4 input [operatorStartUnloading, facadeStoreStarted] output [
    operatorStoreStartedNotified]
  EventActor StoringMonitoring input [operatorStoreStartedNotified,
    operatorStoringNotFinished] output [operatorStoringMonitoring]
  Barrier b5 input [operatorStoringMonitoring, facadeUnitStored] output [operatorUnitStored
  ]
  Condition isStoringFinished input [operatorUnitStored] when-true [operatorStoringFinished
  ] when-false [operatorStoringNotFinished]
  Barrier b6 input [operatorStoringFinished, facadeUnloadingFinished, facadeTruckLeft]
    output [operatorFinished]
  EventBridge OperatorToFacade target [eu.indenica.casestudy.facade.PlatformFacade]
  Application WarehouseOperator start-with [facadeTruckArrived] end-with [operatorFinished]
}
module eu.indenica.casestudy.facade
domain PlatformFacade {
  ...
  EventBridge FacadeToOperator target [eu.indenica.casestudy.warehouse.operator.
    WarehouseOperator]
  ...
}

```

Listing 1.1. Excerpt of the code defining the warehouse operator application

4.3 Event Actor Substitutions

In this section, we illustrate weak substitutions for the change requirements introduced in Section 2. The first requirement is to skip the execution of the task `GetFreeDock`. This can be achieved through a weak substitution, i.e., by defining a new actor `MoveTruckToDockNew` that directly consumes the event `operatorTruckArrivedNotified` emitted by the actor `app`, as illustrated in Listing 1.2.

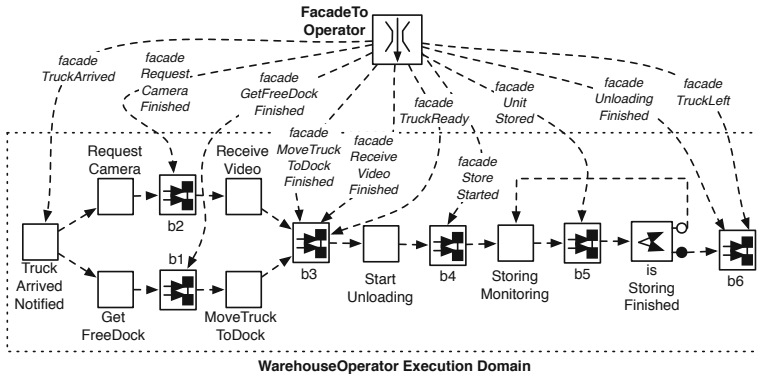


Fig. 6. The graphical representation of the warehouse operator application

```

EventActor MoveTruckToDockNew input[operatorTruckArrivedNotified] output[
operatorMoveTruckToDock]
register [MoveTruckToDockNew] // register the new actor
deactivate [MoveTruckToDock] // temporarily suspend the old actor
deactivate [b1] // we do not need this Barrier
... // verifications can be performed here to detect potential anomalies
activate [MoveTruckToDockNew] // now we can activate the new actor

```

Listing 1.2. Skipping the existing event actor GetFreeDock

In the second requirement from Section 2 the warehouse operators need to call the warehouse staff to prepare for unloading products in the truck. This implies that a new actor, namely, CallWarehouseStaff, must be executed before the StartUnloading actor. We demonstrate in Listing 1.3 how the new actor can be incorporated into the existing warehouse operator composition logic using weak substitutions.

```

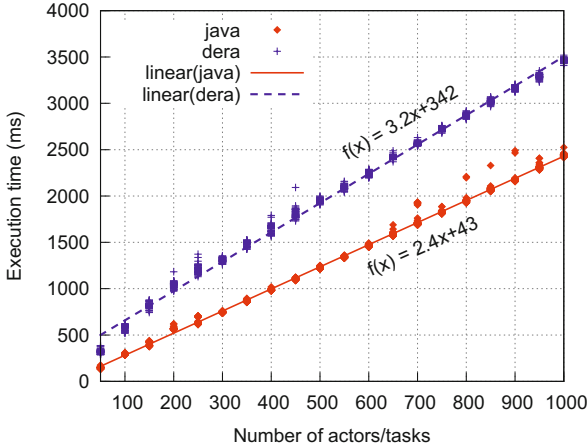
EventActor CallWarehouseStaff input[operatorStoreStartedNotified] output[
operatorCallWarehouseStaff]
EventActor StartUnloadingNew input[operatorCallWarehouseStaff] output[operatorStartUnloading]
register [CallWarehouseStaff]
register [StartUnloadingNew]
deactivate [StartUnloading]
/* verifications can be performed here to detect anomalies */
activate [StartUnloadingNew]
activate [CallWarehouseStaff]

```

Listing 1.3. Adding a new event actor CallWarehouseStaff

4.4 Performance and Scalability Evaluation

As the integration layer stands between the service-based applications and the underlying systems and platforms, it is a potential bottleneck in an integration architecture. Also the channel concept of DERA might introduce a bottleneck that could cause scalability problems. Thus, we conducted an evaluation of the performance and scalability of our approach comparing to a reference implementation based on pure hard-coded Java with rigid dependencies among the tasks. In this reference implementation we used exactly



tasks	\overline{java}	σ_{java}	\overline{dera}	σ_{dera}
50	148	8.2	323.9	23.7
100	293.9	5.1	582.4	26.4
150	403.5	22.1	819.0	31.1
200	569.9	12.7	1042.3	34.4
250	640.9	32.5	1216.0	46.9
300	745.5	6.6	1310.6	22.9
350	865.2	8.7	1484.4	28.8
400	986.4	7.6	1648.8	47.9
450	1103.5	8.6	1813.2	56.5
500	1224.9	8.8	1951.6	22.3
550	1342.1	8.0	2095.7	25.0
600	1466.5	11.2	2245.2	27.8
650	1592.4	20.0	2418.9	20.6
700	1734.8	77.0	2565.7	19.0
750	1827.1	15.6	2721.4	18.9
800	1954.8	52.3	2869.0	20.3
850	2076.6	38.8	3019.5	22.8
900	2186.4	60.8	3165.4	18.9
950	2309.9	25.5	3290.0	24.7
1000	2439.8	16.8	3464.6	22.1

Fig. 7. Evaluation of DERA scalability

the same tasks (a simple service invocation in a server running on the same machine) as in the DERA actor’s tasks. We hard-coded the integration in Java, offering no flexibility, to exactly measure the impact of DERA’s features on performance and scalability.

We evaluate the DERA implementation and the Java counterpart on the Java SE Runtime 1.6.0_u31 64-bit version on a workstation with an Intel CPU Quad-core i7 2.0 GHz and 8 gigabytes memory. To minimize the interference of the Java VM garbage collector and dynamic memory allocations during the experiments, the Java VM is set up with the following options: `-Xms512m -Xmx1024m -Xss1m`. We measured in 50 rounds the execution time of n ($n = 50, 100, \dots, 950, 1000$, respectively) DERA actors running in an execution domain with a fixed thread pool of size 8 (which is the number of CPU cores) and compare to n Java tasks running in a thread pool of the same size.

Scatter plots for the measured execution times are visualized in Figure 7 (each value of n in the 50 rounds is depicted; the values are pretty close together). We derived the two regression functions shown in Figure 7 from the data of the measurements using the least-squares linear regression method. In Figure 7, we also present the average execution time of DERA (i.e., \overline{dera}) and the Java counterpart (i.e., \overline{java}) along with their standard deviations, σ_{dera} and σ_{java} , respectively. As can be seen for the observed data, both our approach and the Java hard-coded implementation offer approximately linear scalability. Our approach introduces only a moderate performance overhead, especially when considering that (1) realistic model sizes – such as those in our industrial case study – seldom go beyond 20-50 actors and (2) the approach is indented to be used for remote service integration and each service invoked over the network requires much more time than what is spent in the integration layer.

5 Related Work

The integration layer targeted in our approach is related to dynamic service composition approaches [10,2,7]. The dynamicity of those approaches is mostly achieved by

deferring service discovery and binding to runtime. Initially, service placeholders or composition rules are prescribed in the configuration so that the enactment engine can later find, select, and combine relevant services on the fly. Moreover, most of these approaches using rigid dependency structures. However, none of the aforementioned approaches provides sufficient supports for flexibly changing or substituting arbitrary elements as in DERA. There are a few exceptions, such as the eFlow framework [6], in which modifications of composite services are allowed, but in an ad-hoc manner. As DERA actors need to interact with and incorporate various services, our approach can benefit from the advanced techniques in discovering and binding services to enhance the dynamicity of the interaction between DERA actors and corresponding services. None of these approaches considers the flexibility at runtime as in our approach.

An extensively used approach for specifying service composition are process-driven SOAs [15]. A typical process comprises a number of tasks and a control flow defining the execution order of these tasks. BPMN⁶ and BPEL⁷ are a widely used languages for describing processes. Unfortunately, they expose tight dependencies among service invocations with rigid control flows and structures. The enactment of BPMN or BPEL descriptions is usually determined at design time and very difficult to change at runtime. There are a substantial amount of efforts focusing on relaxing the rigid structures of process descriptions, and, therefore, enable a certain degree of flexibility of process execution [13,24,23]. These approaches mainly target long-running transactional systems and still suffer from the tight dependencies among the tasks. In contrast, there exists no such rigid control flows in DERA, only the *virtual* relationships among actors. Changing these virtual relationships can be straightforwardly achieved by altering actor interfaces using DERA substitution mechanisms. Our approach focuses on flexible short-running composition logic for integrating software systems and platforms. A number of approaches leverage the aspect-oriented programming paradigm for support process modifications by specifying and weaving additional modules, such as logging, auditing, and security, into the original composition logic [9,8]. However, the aspect-oriented approaches do not aim at loosening the dependency structures and provide limited supports for flexible changes at runtime.

A considerable amount of studies in the field of coordination theory are investigating methods and techniques aiming at separating computation from coordination [3] and making the interaction between components explicit in terms of coordination models and protocols. However, as far as components are still aware of these models and protocols which are stored in the components, their computation and interaction with peers is likely based on, and influenced by, this data [3]. In DERA, computational elements (i.e., actors) are totally unaware of the others. Communication between event actors is fully decoupled from the behavior of the actors.

The theoretical foundation of our work benefits from existing formal methods research in the field of subtyping. DERA substitution mechanisms are based on the studies on the substitutability of data types, objects, and components, especially the well-known Liskov substitution principle [17]. DERA extends these concepts to the domain of event-driven architectures and actors in behavior models. The implementation of

⁶ <http://www.omg.org/spec/BPMN/2.0/PDF>

⁷ <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>

DERA, in particular, the event channels, can be realized using asynchronous communication styles. Therefore, our future plan is to investigate and realize event channels using existing distributed publish-subscribe frameworks such as PADRES [11].

The concept of actor proposed in DERA is different from the *actor models* originally proposed in Agha's dissertation and follow-on studies [1]. The actors in the actor models are more complex as they encapsulate data, method, and interfaces. Moreover, in contrast to DERA systems in which event actors are totally unaware of each other, the actor models require that an actor must know the references, namely, *mail address*, of other actors to communicate with them by exchanging messages. This constraint clearly imposes tight dependencies among actors.

6 Conclusion

In this paper we present dynamic event actors (DERA) as a novel approach that exploits EDA to enable the flexibility of integration architectures and support various kinds of runtime evolution and adaptation. In particular, DERA introduces the concept of event actors with formally specified event interfaces for representing constituting elements (e.g., components, adapters, proxies). The communications and dependencies between actors are neither embedded in the actors nor prescribed in rigid dependency structures as in existing approaches. In contrast, the event-based communication style is exploited for loosening these dependencies among actors. In addition, event substitution mechanisms are proposed to enable the ability of altering DERA applications at runtime by making changes of event actor interfaces and substituting event actors. The main focus of our paper is to introduce DERA concepts and elements grounding on a sound formalization along with a prototypical implementation. The applicability of DERA has been shown through an industrial case study. The evaluation of DERA systems shows linear scalability with moderate performance overhead compared to an equivalent pure Java reference implementation.

A future work plan is to utilize existing formal reasoning methods for concurrent and distributed systems for supporting the verification of DERA system properties such as reachability, boundedness, and liveness, at important stages of development, e.g., before deploying and/or substituting actors. In addition, existing approaches for establishing reliable communication channels shall be exploited and extended in the context of DERA.

Acknowledgment. This work was partially supported by the European Union FP7 project INDENICA (<http://www.indenica.eu>), Grant No. 257483.

References

1. Agha, G.A.: ACTORS: A Model of Concurrent Computation in Distributed Systems. PhD thesis (1985)
2. Alamri, A., Eid, M., Saddik, A.E.: Classification of the state-of-the-art dynamic web services composition techniques. *Int. J. Web Grid Serv.* 2(2), 148–166 (2006)
3. Arbab, F., Talcott, C.L. (eds.): COORDINATION 2002. LNCS, vol. 2315. Springer, Heidelberg (2002)

4. Atluri, V., Chun, S.A., Mulkamala, R., Mazzoleni, P.: A decentralized execution model for inter-organizational workflows. *Distrib. and Parallel Databases* 22(1), 55–83 (2007)
5. Basten, T., van der Aalst, W.M.P.: Inheritance of behavior. *Journal of Logic and Algebraic Programming* 47(2), 47–145 (2001)
6. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
7. Chakraborty, D., Joshi, A.: Dynamic service composition: State-of-the-art and research directions. Tech. Rep. TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, USA (2001)
8. Charfi, A., Mezini, M.: AO4BPEL: An aspect-oriented extension to BPEL. *World Wide Web* 10(3), 309–344 (2007)
9. Cibrán, M.A., Verheeecke, B., Vanderperren, W., Suvéé, D., Jonckers, V.: Aspect-oriented programming for dynamic web service selection, integration and management. *World Wide Web* 10(3), 211–242 (2007)
10. D’Mello, D.A., Ananthanarayana, V.S., Salian, S.: A Review of Dynamic Web Service Composition Techniques. In: Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds.) CCSIT 2011, Part III. CCIS, vol. 133, pp. 85–97. Springer, Heidelberg (2011)
11. Fidler, E., Jacobsen, H.A., Li, G., Mankovski, S.: The PADRES distributed publish/subscribe system. In: Feature Interactions in Telecommunications and Software Systems VIII, ICFI 2005, pp. 12–30 (2005)
12. Ganesan, S., Yoon, Y., Jacobsen, H.A.: NIñOS take five: the management infrastructure for distributed event-driven workflows. In: 5th ACM Int’l Conf. on Distributed Event-based System (DEBS), pp. 195–206. ACM (2011)
13. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the propov approach. *J. Softw. Maint. Evol.* 22, 519–546 (2010)
14. Hens, P., Snoeck, M., Backer, M.D., Poels, G.: Transforming Standard Process Models to Decentralized Autonomous Entities. In: 5th SIKS/BENAIIS Conf. on Enterprise Information Systems, pp. 97–106. CEUR WS.org, Aachen (2010)
15. Hentrich, C., Zdun, U.: Process-Driven SOA: Patterns for Aligning Business and IT. Infosys Press (2012)
16. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice Hall (April 1985)
17. Liskov, B.H., Wing, J.M.: A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems* 16(6), 1811–1841 (1994)
18. Luckham, D.C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, Boston (2001)
19. Milner, R.: *Communicating and Mobile Systems: the Pi-Calculus*, 1st edn. Cambridge University Press (June 1999)
20. Mühl, G., Fiege, L., Pietzuch, P.: *Distributed Event-Based Systems*, 1st edn. Springer (2006)
21. Murata, T.: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77(4), 541–580 (1989)
22. Pierce, B.C.: *Types and Programming Languages*. The MIT Press (February 2002)
23. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: Modelling flexible processes with business objects. In: IEEE Conf. Commerce and Enterprise Computing (CEC), pp. 41–48 (2009)
24. Reichert, M., Dadam, P.: Enabling adaptive process-aware information systems with ADEPT2. In: *Handbook of Research on Business Process Modeling*, pp. 173–203. Information Science Reference (2009)
25. Tombros, D., Geppert, A.: Building Extensible Workflow Systems Using an Event-Based Infrastructure. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 325–339. Springer, Heidelberg (2000)

A Conditional Lexicographic Approach for the Elicitation of QoS Preferences

Raluca Iordache and Florica Moldoveanu

University "POLITEHNICA" of Bucharest, Romania
riordache@hotmail.com, fm@cs.pub.ro

Abstract. In a service-oriented environment, clients can usually choose between several web services offering the same functionality. The web service selection can be automated by allowing clients to specify non-functional requirements such as quality of service. Clients should also be able to indicate how to make tradeoffs when some of these requirements cannot be met. The ability to capture tradeoff preferences is critical for selecting the best fitting web service. In this paper, we propose a method of expressing non-functional preferences, which requires minimal effort on the part of the clients, but offers great flexibility in managing tradeoffs. This method leads to a simple algorithm for selecting web services, which does not require sophisticated multicriteria decision techniques.

Keywords: QoS preferences, multicriteria decision making, service selection.

1 Introduction

The emergence of services brought the world of computing in front of a new level of abstraction that is closer to the way humans naturally think and interact with their surroundings [1]. In real life, people make use of particular services, after selecting from the available alternatives the ones suiting best their requirements. In service-oriented environments, the existence of numerous web services offering the same functionality needed for a given task leaves the application designer with several candidates to choose from. At this point, analyzing the quality of the alternatives starts playing a fundamental role in the service selection. The non-functional characteristics of a web service, such as availability, cost, response time, or the supported security protocols define the Quality of Service (QoS) concept.

Although considerable research has been done in the recent years, there is currently no widely accepted approach for the QoS-aware selection of web services. This is mainly due to the various issues that have to be addressed in order to provide a complete solution. These issues include the design of suitable frameworks and architectures [2][3], which should provide ontologies for the formal specification of QoS metrics [4][5], methods of obtaining current metric values [6] and algorithms that select the best web service based on user-specified QoS criteria [7].

The ability of clients to express their QoS expectations plays a crucial role in the selection of the most suitable web service. While hard constraints are relatively easy to formulate, there is no standard way to deal with soft constraints that should reflect client's preferences in situations where no web service is capable of satisfying all QoS requirements. Most approaches are based on specifying priorities or associating weights to the different QoS dimensions. The drawback of these methods is that they cannot accurately capture user's preferences. On the other hand, more elaborate ways of specifying QoS preferences usually require considerable effort on the part of the clients. This complexity brings the risk that users don't understand the method or they are not willing to spend so much time expressing their preferences.

In this paper, we propose a conditional lexicographic method of articulating non-functional preferences, which offers great flexibility, while being easy to use and understand. It is based on the way people reason about their preferences, thus fostering its acceptance by users. This method leads to a simple algorithm for selecting web services, which does not require sophisticated multi-criteria decision techniques.

The rest of this paper is organized as follows: Section 2 presents the addressed problem of expressing the user's preferences. Section 3 outlines the related work of service selection based on multicriteria techniques. Section 4 introduces our conditional lexicographic approach illustrated by a case study of a data visualisation service. The last section concludes the paper and outlines future work directions.

2 The Problem of Expressing Preferences

Preference models can be found in various areas like psychology, mathematics, philosophical literature, in economics and game theory, in operations research and decision analysis and in various disciplines of computer science. The choices that we make are guided by our preferences. Understanding preference handling is relevant when attempting to build systems that make choices on behalf of users [8].

Of particular interest for the domain of QoS-aware service selection are the fields of multicriteria decision analysis and in particular of multiobjective optimization or Pareto optimization. Multiobjective optimization problems can also be found in various areas where optimal decisions involve tradeoffs between multiple (possibly conflicting) objectives. A Pareto optimal solution is a solution for which it is impossible to improve one objective without worsening another one. Multiobjective optimization uses a priori or a posteriori approaches, depending on the moment when the decision maker's preferences are articulated.

The best known and simplest method for preference articulation is the weighted sum method. The method uses weight values supplied by the user to describe the importance of the objectives. One drawback of this method is that the weights must both compensate for differences in objective function magnitudes and provide a value corresponding to the relative importance of an objective. Another

drawback is that it is not able to find certain solutions in the case of a non-convex Pareto curve. The authors of [9] conclude that the weighted sum method "is fundamentally incapable of incorporating complex preference information".

Lexicographic preferences is another simple method used for modeling rational decision behavior. Preferences are defined by a lexical ordering, which leads to a strict ranking. While being very easy to use, lexicographic preferences have the major drawback of being non-compensatory. An extension of this method is lexicographic semiorder, where a tradeoff is addressed in situations where there is a significant improvement in one objective that can compensate an arbitrarily small loss in the most important objective. An alternative x is considered better than an alternative y if the first criterion that distinguishes between x and y ranks x higher than y by an amount exceeding a fixed threshold [11]. The advantage of this method is that it ensures that a solution that is slightly better on the most important objective but a lot worse on the other objectives will not be selected.

3 Existing Methods and Approaches

Most work on QoS-aware service selection uses successive evaluation of different, non functional aspects in order to attribute a general "level of quality" to a service [3]. Usually the selection of the service offering the best functionality is based on either a single evaluation criterion or on a weighted sum of several quantitative evaluation criteria. These approaches have in practice major disadvantages because in most of the cases a single criterion is not enough for defining the user's QoS requirements and the weighted sum of the criteria lead to compensation problems and so to inadequate results. Many QoS attributes are qualitative parameters and cannot be used on a weighted sum evaluation. By using quantification and assigning values to the qualitative data, these parameters can be transformed in quantitative parameters. This is done usually by defining a measurement scale and then associating to each level of the scale a numerical value. For example the scale can be defined as 1-5, with 1 being very high and 5 being very low and these numbers are associated to the different qualitative attributes, by their relevance.

In practice such a quantification approach doesn't show good results in the QoS-aware service selection, and the challenge of obtaining good results lies in managing tradeoffs among QoS expectations in situations in which service requesters specify quality levels that cannot be simultaneously met.

This key problem of managing tradeoffs of the QoS preferences is addressed by [12]. The authors propose a QoS model for describing the QoS dimensions (requirements and preferences) of the service requester and the service provider. They are using fuzzy multicriteria decision analysis (MCDA) for comparing the models and ranking the competing services according to the values of their characteristics. The model describing the QoS properties is based on OMG's UML QoS Framework metamodel. MCDA methods allow the defining of weights related to criteria and also weights to interactions between criteria.

The authors of [13] propose an extension of the Web Service architecture by adding to the UDDI registry a new component called Multicriteria Evaluation Component (MEC) used for the multicriteria evaluation. This evaluation is based on a Web version of IRIS (Interactive Robustness Analysis and Parameters Inference for multicriteria Sorting Problems), which uses the ELECTRE TRI method.

Dealing with preferences, their priorities and possible tradeoffs between them has been addressed by [11] using a model of lexicographic semiorder. The work is addressing decision making based on lexicographic heuristics and ranking in order to compare a pair of alternatives.

4 The Conditional Lexicographic Approach

As mentioned before, QoS expectations can take the form of hard and soft constraints. While all hard constraints must be satisfied in order for a web service to be selected, soft constraints represent rather desirable characteristics of the chosen service. If no web service meets all soft constraints, users should have the possibility to express their tradeoff preferences, in order to allow the dynamic selection of services.

Our approach to articulate the QoS preferences is based on the observation that, when trying to find a set of rules allowing them to choose between several alternatives, people start by ranking their preferences, in accordance with their perceived importance. This action is equivalent to imposing a lexicographic order on the different criteria that have to be considered. In most situations, using such a strict hierarchy is not sufficient to capture people's real preferences. In this case, people usually introduce additional rules that change the criteria priorities when some specific condition is met.

We propose a method to establish a total order on the set of existing web service alternatives, by attaching conditions to lexicographic preferences and we introduce a preference specification language that can be used for authoring QoS preferences.

We illustrate our approach and the use of its associated specification language by considering a hypothetical company that offers data visualization services. One of these services is the generation of charts based on data sets. Instead of performing itself such tasks, the company delegates them to other business partners, which offer web services for chart generation. A service selection broker chooses the most suitable web service, based on QoS requirements formulated by clients. The chart generation web services are characterized by domain-independent QoS attributes (e.g., availability, response time) and domain-specific ones (e.g., chart type, cost per chart, number of colors, image resolution).

As client of the hypothetical company we consider a data acquisition system, which regularly sends charts depicting the state of an industrial process to a 1280 x 720 monitor capable of displaying 65536 colors. The image displayed on the monitor can be updated only at fixed intervals of 5 seconds. If a new chart is not available at the end of a 5 seconds interval, the monitor update is postponed until the next end of a 5 seconds interval.

Our preference specification language allows specifying both constraints and preferences. Constraints are declared as a list of comma separated boolean conditions that must be satisfied by the service. They are enclosed in a *constraints* block, as shown in Fig. 1.

```
constraints {
  chartType = "time series",
  cost < 10,
  availability > 0.95,
  imageResolution = "1280x720",
  responseTime < 10
}
```

Fig. 1. Hard constraints specification

The order of constraint conditions is irrelevant, but order plays a key role in the articulation of QoS preferences. For the beginning, we consider that the client provides a strict ranking of preferences. This is expressed in our specification language by using a *preferences* block that includes the comma separated list of relevant QoS attributes in the order of their importance, as shown in Fig. 2.

```
preferences {
  cost,
  availability : high,
  responseTime,
  colors : high
}
```

Fig. 2. A simplistic specification of preferences

For each QoS attribute the client should indicate the direction associated with better values. This piece of information appears after the attribute name, separated by a colon. Possible values for direction are *low* and *high*, where *low* is the default direction and can be omitted.

In the example above, *cost* is the most important QoS attribute, and services with a lower cost are considered better. However, this specification of preferences does not accurately capture client's preferences. A first problem is that selecting a web service with a response time greater than 5 seconds would result in skipping an update of the monitor. This is a serious issue, and such a scenario should be prevented even if this leads to a higher cost. The *responseTime* attribute should be ranked higher only when exactly one of the two web services compared has a value higher than 5 seconds for this attribute. If, for example, both web services considered are able to provide the chart in less than 5 seconds, the problem of missing an update does no longer exist and *responseTime* does not need a higher ranking. Conversely, if both web services compared have a *responseTime* higher

than 5 seconds, an update of the monitor will be inevitably skipped, and the actual value of this attribute is no longer of critical importance.

Another problem arises when at least one of the web services compared provides a number of colors less than 65536. Since this will lead to a loss of quality, the client may want to increase in such situations the importance of the *colors* attribute. If both web services provide a number of colors higher than 65536, the *colors* attribute becomes irrelevant, because the difference in quality cannot be detected on the available monitor.

Finally, a small difference in the values of the *cost* attribute should be ignored if the selection of the slightly more expensive web service leads to a better color quality.

In order to be able to articulate preferences for scenarios like the one above, our specification language provides four unary preference operators, which are shown in Table 1.

Table 1. Preference operators

Preference operator	Meaning
AT_LEAST_ONE *(condition)	condition(<i>service</i> ₁) OR condition(<i>service</i> ₂)
EXACTLY_ONE (condition)	condition(<i>service</i> ₁) XOR condition(<i>service</i> ₂)
ALL (condition)	condition(<i>service</i> ₁) AND condition(<i>service</i> ₂)
DIFF (attribute)	<i>service</i> ₁ .attribute – <i>service</i> ₂ .attribute

*default operator (can be omitted)

The first three operators take as argument a boolean formula, which usually involves one or more QoS attributes. The formula is evaluated twice, once for each of the web services to be compared. The two resulting boolean values are passed as arguments to the boolean operator (OR, XOR, or AND) associated with the given preference operator, in order to obtain the return value.

The preference operator **DIFF** takes as argument a QoS attribute and returns the modulus of the difference of its corresponding values from the two web services compared. Our specification language also allows the definition of virtual QoS attributes, which will be treated as genuine QoS attributes by the preference operators. This can be done by means of the *def* directive, as seen in the example below:

```
def colorDepth = log2(colors)
```

In the remainder of this paper, we use the term *preference rule* to denote an entry in the *preferences* block. As already seen, a preference rule has three components: an optional *condition*, an *attribute* indicating the QoS dimension used in comparisons and a *direction* flag stating which values should be considered better. In our specification language, the preferences corresponding to the above described scenario can be articulated as shown in Fig. 3:


```

preferences {
  [EXACTLY_ONE(responseTime > 5)] responseTime,
  [DIFF(cost) > 2] cost,
  [colors < 65536] colors : high,
  availability : high,
  responseTime,
  colors : high
}

```

Fig. 3. A more elaborate specification of preferences

The condition part of the third preference rule in the above *preferences* block (i.e., [colors < 65536]) does not explicitly specify a preference operator, which means it uses the default operator AT_LEAST_ONE.

The specification language can deal with situations where people are not fully aware of their preferences. When users notice that the current rules do not accurately capture their preferences, they can simply add a new conditional rule, thus incrementally improving the preference specification.

In what follows, we use the notation $s_1 \succ s_2$ to indicate that the web service s_1 is preferred to the web service s_2 , and the notation $s_1 \sim s_2$ to indicate that the service s_1 is indifferent to the web service s_2 (i.e., s_1 and s_2 are equally preferred). Additionally, we introduce the notation $s_1 \overset{k}{\succ} s_2$ to indicate that the web service s_1 is preferred to the web service s_2 and that the preference rule k has been decisive in establishing this relationship. We also introduce the complementary operators \prec and $\overset{k}{\prec}$, defined by the following relations:

$$s_1 \prec s_2, \text{ iff } s_2 \succ s_1$$

$$s_1 \overset{k}{\prec} s_2, \text{ iff } s_2 \overset{k}{\succ} s_1$$

An algorithm for comparing two web services based on the preferences expressed using our conditional lexicographic approach is shown in Fig. 4.

The algorithm examines all entries in the *preferences* block in the order in which they appear (line 2). If the current preference rule has no attached condition or the attached condition evaluates to true (line 6), the values corresponding to the attribute specified by this entry are compared (line 7). The *compare* function returns a numerical value that is positive if the first argument is better, negative if the second argument is better and 0 if the arguments are equal (see pseudocode in Fig. 5). If the attribute values are not equal (line 8), the algorithm returns a tuple containing the result of the current comparison and the index of the preference rule that has been decisive in establishing the preference relationship (line 9). Otherwise, the algorithm continues its execution with the next preference rule. A null return value (line 13) indicates an indifference relation between the two web services, while a not-null tuple identifies a relation of type \prec or \succ between them.

```

1. function compareServices(service1, service2, preferences)
2.   for i ← 1 .. length(preferences) do
3.     cond ← preferences[i].condition
4.     attr ← preferences[i].attribute
5.     dir ← preferences[i].direction
6.     if cond = null OR cond(service1, service2) = true then
7.       result ← compare(service1.attr, service2.attr, dir)
8.       if result ≠ 0 then
9.         return {result, i}
10.      end if
11.    end if
12.  end for
13.  return null
14. end function

```

Fig. 4. Pairwise comparison of two web services

```

function compare(attr1, attr2, dir)
  if attr1 = attr2 then
    result ← 0
  else if attr1 < attr2 then
    result ← 1
  else
    result ← -1
  end if
  if dir = high then
    result ← -result
  end if
  return result
end function

```

Fig. 5. Pairwise comparison of QoS attribute values

In a series of experiments, Tversky [10] has shown that people have sometimes intransitive preferences. Therefore, being able to capture such preferences is an important feature of our specification language. However, a consequence of allowing intransitive preferences is that the pairwise comparison of all web service alternatives is in general not sufficient to impose a total order on these services. In order to illustrate this, we use a simplified version of the preferences specified in Fig. 3. As shown in Fig. 6, the simplified version does no longer involve the QoS attribute *availability*. Therefore, this attribute is no longer relevant for the web service ranking. Although the simplified specification used for exemplification is unrealistic, it is easier to analyze and it helps us highlight the intransitivity issues. (The preference rule indexes appearing at the left side of the figure are only informative and are not part of the preference specification.)

We consider a set of 5 web service alternatives (WS_1 through WS_5) with the relevant QoS attribute values specified in Table 2.

<pre> preferences { 1. [EXACTLY_ONE(responseTime > 5)] responseTime, 2. [DIFF(cost) > 2] cost, 3. [colors < 65536] colors : high, 4. responseTime } </pre>

Fig. 6. A simplified specification of preferences used for exemplification

Table 2. Relevant QoS attribute values

	WS_1	WS_2	WS_3	WS_4	WS_5
responseTime	7.0	7.0	5.5	4.5	7.5
cost	4.0	5.0	6.5	8.0	7.5
colors	256	256	256	65536	65536

The relations identified by the pairwise comparison of the 5 web services considered in our example are depicted in Table 3, where header notations use the format i / j to indicate that the corresponding symbol in the line below represents the preference relation between the web services WS_i and WS_j .

Table 3. Pairwise comparison of the 5 web services

1/2	1/3	1/4	1/5	2/3	2/4	2/5	3/4	3/5	4/5
\sim	\succ_2	\prec_1	\succ_2	\prec_4	\prec_1	\succ_2	\prec_1	\prec_3	\succ_1

Several cases of intransitivity of preferences can be observed in the above table. A first example is given by the following relations:

$$\begin{aligned}
 WS_1 &\succ WS_3 \succ WS_2 \\
 WS_1 &\sim WS_2
 \end{aligned}$$

Although WS_1 is indifferent to WS_2 , WS_1 is preferred to WS_3 , while WS_2 is not preferred to WS_3 .

Another example is the rock-paper-scissors relationship induced by:

$$\begin{aligned}
 WS_2 &\prec WS_3 \prec WS_5 \\
 WS_5 &\prec WS_2
 \end{aligned}$$

In order to obtain a total order on the set of web service alternatives, we attach to each web service i a score vector of integer values: $V_i \in \mathbb{N}^{r+1}$, where r is the number of preference rules. The algorithm used to compute the score vectors is presented in Fig. 7, where n denotes the number of web service alternatives.

```

procedure createScoreVectors()
  for i ← 1 .. n do
    for k ← 1 .. r do
       $V_i^k \leftarrow$  number of times service  $WS_i$  is preferred to another
        web service due to decisive rule  $k$  (i.e., due to a  $\succ_k$  relation).
    end for
       $V_i^{r+1} \leftarrow$  number of times service  $WS_i$  is indifferent to another web service.
    end for
  end procedure

```

Fig. 7. Procedure to create the score vectors

For the 5 web service alternatives considered in our example, the corresponding score vectors computed with the above algorithm are presented in Fig. 8.

	\succ_1	\succ_2	\succ_3	\succ_4	\sim
WS ₁	0	2	0	0	1
WS ₂	0	1	0	0	1
WS ₃	0	0	0	1	0
WS ₄	3	0	0	0	0
WS ₅	0	0	1	0	0

Fig. 8. Score vectors of the 5 web service alternatives

Using the score vectors, we are able to provide an algorithm for the ranking of web service alternatives. This algorithm is based on the function *compareScores*, described in pseudocode in Fig. 9. Again, r is used to denote the number of preference rules. The function takes as arguments two score vectors and returns a numerical value that is positive if the web service corresponding to the first score vector is preferred, negative if the web service corresponding to the second score vector is preferred and 0 if the corresponding web services are indifferent to each other.

For each of the two corresponding web services, the function computes the number of times it has been preferred to other web services (lines 2, 3). This computation does not take into account the number of times a web service has been found to be indifferent to another one (hence the sum is taken up to the value r , not $r + 1$).

If the previously computed values $count_1$ and $count_2$ are not equal (line 4), the web service with the higher value is chosen as the better one (line 5).

Otherwise, the algorithm scans each position in the score vectors (line 7) and if it finds different values, the web service corresponding to the higher value is chosen as the better one (lines 8-10). The scanning of the values in the vector scores starts with the position corresponding to the first preference rule, because this is considered the most important one, and it ends with the position

```

1. function compareScores( $V_1, V_2$ )
2.    $count_1 \leftarrow \sum_{i=1}^r V_1^i$ 
3.    $count_2 \leftarrow \sum_{i=1}^r V_2^i$ 
4.   if  $count_1 \neq count_2$  then
5.     return  $count_1 - count_2$ 
6.   end if
7.   for  $i \leftarrow 1 .. r + 1$  do
8.     if  $V_1^i \neq V_2^i$  then
9.       return  $V_1^i - V_2^i$ 
10.    end if
11.  end for
12.  return 0
13. end function

```

Fig. 9. Function for score vector comparison

corresponding to the number of indifference relations (i.e., $r + 1$), because this is considered the least important one. If the score vectors are identical, the function returns 0 (line 12).

In contrast with the function *compareServices* presented in Fig. 4, the function *compareScores* induces a total order on the set of web service alternatives, thus allowing us to rank them accordingly. Using this algorithm, the 5 web service alternatives considered in our example will be ranked in the following order:

$(WS_4, WS_1, WS_2, WS_5, WS_3)$,

with WS_4 being the best alternative.

5 Conclusions and Future Work

In this paper we have introduced a new approach of ranking service alternatives based on the users QoS expectations. The users can define their requirements and preferences by using a simple and intuitive specification language that attaches conditions to lexicographic rules. Our approach facilitates the elicitation of preferences from clients, because it resembles the way people express trade-offs when reasoning about their preferences. The proposed method can deal with intransitive preferences and with situations where people are not fully aware of their preferences.

Our current efforts are directed toward designing and implementing a framework for dynamic web service selection that supports the handling of QoS preferences based on the approach presented in this paper. A prototype implementation of the ranking engine is available at <http://qospref.sourceforge.net/> and we plan to also offer open source implementations for the other components of our framework.

References

1. Medjahed, B., Bouguettaya, A.: *Service Composition for the Semantic Web*. Springer (2011)
2. Maximilien, E.M., Singh, M.P.: A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing* 8(5), 84–93 (2004)
3. Zeng, L., Benatallah, B.: QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* 30(5), 311–327 (2004)
4. Zhou, C., Chia, L.T., Lee, B.S.: DAML-QoS Ontology for Web Services. In: *International Conference on Web Services*, pp. 472–479 (2004)
5. Papaioannou, I.V., Tsesmetzis, D.T., Roussaki, I.G., Anagnostou, M.E.: A QoS Ontology Language for Web-Services. *Advanced Information Networking and Applications*, 101–106 (2006)
6. Zeng, L., Lei, H., Chang, H.: Monitoring the QoS for Web Services. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007*. LNCS, vol. 4749, pp. 132–144. Springer, Heidelberg (2007)
7. Day, J., Deters, R.: Selecting the best web service. In: *Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 293–307 (2004)
8. Brafman, R.I., Domshlak, C.: Preference Handling — An Introductory Tutorial. *AI Magazine*, 58–86 (2009)
9. Marler, R.T., Arora, J.S.: The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization* 41(6), 853–862 (2010)
10. Tversky, A.: Intransitivity of Preferences. *Psychological Review* 76(1), 31–48 (1969)
11. Manzini, P., Mariotti, M.: Choice by lexicographic semiorders. *Theoretical Economics* 7(1) (2010)
12. Herssens, C., Jureta, I.J., Faulkner, S.: Capturing and Using QoS Relationships to Improve Service Selection. In: Bellahsène, Z., Léonard, M. (eds.) *CAiSE 2008*. LNCS, vol. 5074, pp. 312–327. Springer, Heidelberg (2008)
13. Chakhar, S., Haddad, S., Mokdad, L., Mousseau, V.: *Multicriteria Evaluation-Based Conceptual Framework for Composite Web Service Selection. Evaluation and Decision Models: Real Case Studies*. Springer, Berlin (2011)

Goal-Based Composition of Stateful Services for Smart Homes

Giuseppe De Giacomo¹, Claudio Di Ciccio¹, Paolo Felli¹,
Yuxiao Hu², and Massimo Mecella¹

¹ SAPIENZA – Università di Roma, Italy
{degiacono,cdc,felli,mecella}@dis.uniroma1.it

² Google Waterloo, Canada
yuxiao@google.com

Abstract. The emerging trend in process management and in service oriented applications is to enable the composition of new distributed processes on the basis of user requests, through (parts of) available (and often embedded in the environment) services to be composed and orchestrated in order to satisfy such requests. Here, we consider a user process as specified in terms of repeated goals that the user may choose to get fulfilled, organized in a kind of routine. Available services are suitably composed and orchestrated in order to realize such a process. In particular we focus on smart homes, in which available services are those ones offered by sensor and actuator devices deployed in the home, and the target user process is directly and continuously controlled by the inhabitants, through actual goal choices. We provide a solver that synthesizes the orchestrator for the requested process and we show its practical applicability in a real smart home use case.

Keywords: process/service composition, smart houses/buildings, planning techniques.

1 Introduction

The promise of Web services (WSs) and of Service Oriented Architectures (SOAs) in general, coupled with the technologies and methodologies of Business Process Management (BPM), is to enable the composition of new distributed processes/solutions: (parts of) available services can be composed and orchestrated in order to realize complex processes, offering advanced functionalities to users.

Many approaches (surveyed, e.g., in [3]) have been proposed in the last years in order to address the above problem from different viewpoints. Works based on Planning in AI, such as [16,24,5,26] consider only the input/output specification of available services, which is captured by atomic actions together with their pre- and post-conditions (a notable extension is [2]), and specify the overall semantics in terms of propositions/formulas (facts known to be true) and actions, affecting the proposition values. All these approaches consider *stateless*

services, as the operations offered to clients do not depend on the past history, as services do not retain any information about past interactions. Also other works (e.g., [25,17,8,6]) consider available services as atomic actions, but, rather than on (planning-based) composition, they focus on modeling issues and automatic service discovery, by resorting to rich ontologies as a basic description mean. Many works (e.g., [15,22,1,18] consider how to perform composition by taking into account Quality-of-Service (QoS) of the composite and component services. Some works consider non classical techniques (e.g., [23] adopts learning approaches) for solving the composition problem.

There are also approaches (e.g., [12]) that consider *stateful* services, which impose constraints on the possible sequences of interactions (a.k.a., conversations) that a client can engage with the service. Stateful services raise additional challenges, as the process coordinating such services should be correct w.r.t. the possible conversations allowed by the services themselves. An interesting approach of this type is the one of [19], in which the specification is a set of atomic actions and propositions, like in planning, services are (finite-state) transition systems whose transitions correspond to action executions, which, in general, affect the truth values of propositions, and the client requests a (main) goal (i.e., a formula built from the above propositions) to be achieved, while requiring runtime failures to be properly handled by achieving a special exception handling goal. Another interesting approach is the one adopted in [4], often referred to as *Roman Model*, in which again services are abstracted as transition systems and the objective is to obtain a composite service that preserves a desired interaction, expressed as a (virtual) target service.

In this paper, we consider a notable extension of the Roman Model, where *goal-based processes* are used, instead of target services, to specify what the user desires to achieve. Such processes can be thought of as *routines* built from virtual tasks expressed declaratively simply as *goals*, which allow users to specify the desired state of affair to bring about. Such goals are organized in a control flow structure, possibly involving loops that regulates their sequencing, as well as the decision points where the user can choose the next goal to request.

Wrt [19], the main novelty proposed in this paper is allowing clients to request new goals, once the current one is achieved (by a plan). In general, such requests can be arranged as routines represented as finite-state transition systems whose transitions correspond to goal requests. These routines typically involve loops, thus ruling out naïve approaches based on (classical, conditional or conformant) planning. Indeed, not all plans that achieve a requested goal are successful: some might lead the system to states preventing future client requests fulfillment. Such *bad* plans could be recognized by taking into account *all* goals the client can request in the future, which, in the presence of loops, span over an infinite horizon (though finite-state).

The approach proposed here is strongly motivated by challenging applications in the domain of smart houses and buildings, i.e., buildings pervasively equipped with sensors and actuators making their functionalities available according to the service-oriented paradigm. In order to be dynamically configurable and

composable, embedded services need to expose semantically rich service descriptions, comprising (i) interface specifications and (ii) specifications of the externally visible behaviors. Moreover, human actors in the environment can be abstracted as services, and actually “wrapped” by a semantic description (e.g., a nurse offering medical services). This allows them to be involved in orchestrations and to collaborate with devices, to reach certain goals. See, e.g., [14,11,7].

We envision a user that can express processes she would like to have realized in the house, in the form of routines consisting of goals (e.g., states of the house she would like to have realized); an engine automatically synthesizes the right orchestration of services able to satisfy the goals. Users can interact with the house through different kinds of interfaces, either centralized (e.g., in a home control station) or distributed, and embedded in specific interface devices. Brain Computer Interfaces (BCIs) allow also people with disabilities to interact with the system. Using such interfaces, users issue specific goals to the system, which is, in turn, expected to react and satisfy the request.

In this paper, we detail this approach. We provide a framework for composition of goal-oriented processes from available (non-atomic) services (Section 2). We present a case study where the framework is applied in a real smart home setting (Section 3). We provide an effective solver (Section 4), which synthesizes an orchestrator that realizes the target goal-oriented processes, by detailing sub-processes that fulfil the various goals at the various point in time. Our solver is sound and complete, and far more practical than other solutions proposed in literature, also because it easily allows for exploiting heuristic in the search for the solution. We show the effectiveness of our solver with some experiments in our use case (Section 5). We conclude the paper with a brief discussion on further work (Section 6).

2 Framework

We assume that the user acts on an environment that is formalized as a possibly nondeterministic *dynamic domain* \mathcal{D} , which provides a symbolic abstraction of the world that the user acts in. Formally, a *dynamic domain* is a tuple $\mathcal{D} = \langle P, A, D_0, \rho \rangle$, where:

- $P = \{p_1, \dots, p_n\}$ is a finite set of *domain propositions*. $D \in 2^P$ is a *state*;
- $A = \{a_1, \dots, a_r\}$ is the finite set of *domain actions*;
- $D_0 \in 2^P$ is the *initial state*;
- $\rho \subseteq 2^P \times A \times 2^P$ is the *transition relation*. We freely interchange notations $\langle D, a, D' \rangle \in \rho$ and $D \xrightarrow{a} D'$ in \mathcal{D} .

Intuitively, a dynamic domain models an environment whose states are described by the set P of boolean propositions, holding all relevant information about the current situation. For instance, the state of a room can be defined by the light being on or off and the door being open or closed, using two propositions *light_on* and *door_open*. By convention, we say that if one of such propositions is in the current state of \mathcal{D} , then it evaluates to \top (true), otherwise it is \perp (false). Hence,

a propositional formula φ over P holds in a domain state $D \in 2^P$ ($D \models \varphi$) if φ evaluates to \top when all of its propositions occurring in D are replaced by \top . However, such domain can not be manipulated directly, i.e., domain actions can not be accessed directly by the user: they are provided through *available services*. The idea is that, at each moment, a service offers a set of possible actions, and the user can interact with the domain \mathcal{D} *only* by means of available services.

Given a dynamic domain \mathcal{D} , a *service* over \mathcal{D} is a tuple $\mathcal{B} = \langle B, O, b_0, \varrho \rangle$, where: (i) B is the finite set of service states; (ii) O is the finite set of service actions over the domain, i.e., $O \cap A \neq \emptyset$; (iii) $b_0 \in B$ is the service initial state; (iv) $\varrho \subseteq B \times O \times B$ is the service transition relation. We will interchange notations $\langle b, a, b' \rangle \in \varrho$ and $b \xrightarrow{a} b'$ in \mathcal{B} .

As a service is instructed to perform an action over \mathcal{D} , both the service and the domain evolve *synchronously* (and possibly *nondeterministically*) according to their respective transition relations. So, for a domain action to be carried out, it needs to be both compatible with the domain and (currently) available in some service. However, services can also feature *local* actions, i.e., actions whose execution does not affect the domain evolution. For instance, a service representing a physical device might require to be switched on to use all its functionalities, a fact that is not captured by \mathcal{D} alone. To define formally this idea, we introduce the notion of executability for actions of a service $\mathcal{B} = \langle B, O, b_0, \varrho \rangle$: given \mathcal{B} in its own service state b and a domain \mathcal{D} in domain state D , action $a \in O$ is said to be *executable* by \mathcal{B} in b iff (i) it is available in b , i.e. $b \xrightarrow{a} b'$ in \mathcal{B} for some state $b' \in B$ and (ii) it is either a local action ($a \notin O \cap A$) or it is allowed in D , i.e., there exists a domain state D' such that $D \xrightarrow{a} D'$. Notice that services are loosely-coupled with the domain they are interacting with: new services can be easily added to the systems and modifications to the description of the underlying domain do not affect them.

Example 1. Consider a dynamic domain $\mathcal{D} = \langle P, A, D_0, \rho \rangle$ describing (among other components) a simple door as in Figure 2a. A domain proposition $doorIsOpen \in P$ is used to keep its state, and the door can be either closed or opened executing domain actions $\{doClose, doOpen\} \subseteq A$. However, the door can only be managed through a service $doorSrv = \langle \{\text{open}, \text{closed}\}, \{\text{doOpen}, \text{doClose}\}, \text{open}, \varrho \rangle$ where ϱ is such that $\text{open} \xrightarrow{\text{doClose}} \text{closed}$ and $\text{closed} \xrightarrow{\text{doOpen}} \text{open}$. Assume that $doorSrv$ is in its state open , and the current domain state D to be such that $doorIsOpen \in D$ (i.e., it evaluates to true in D). As soon as action doClose is executed, both the $doorSrv$ service evolves changing its state to closed and, *synchronously*, the domain evolves to a state D' such that $doorIsOpen \notin D'$ (i.e., it evaluates to false in D').

Given a dynamic domain \mathcal{D} and a fixed set of available services over it, we define a *dynamic system* to be the resulting global system, seen as a whole: it is an abstract structure used to capture the interaction of available services with the environment. Formally, given a dynamic domain \mathcal{D} and a set of available services $\mathcal{B}_1, \dots, \mathcal{B}_n$, with $\mathcal{B}_i = \langle B_i, O_i, b_{i0}, \varrho_i \rangle$, the corresponding *dynamic system* is the tuple $\mathcal{S} = \langle S, \Gamma, s_0, \emptyset \rangle$, where:

- $S = (B_1 \times \dots \times B_n) \times 2^P$ is the set of system states;
- $\Gamma = A \cup \bigcup_{i=1}^n O_i$ is the set of system actions;
- $s_0 = \langle \langle b_{10}, \dots, b_{n0} \rangle, D_0 \rangle \in S$ is the system initial state;
- $\vartheta \subseteq S \times (\Gamma \times \{1, \dots, n\}) \times S$ is the system transition relation such that
 - $\langle \langle b_1, \dots, b_n \rangle, D \rangle \xrightarrow{a,i} \langle \langle b'_1, \dots, b'_n \rangle, D' \rangle$ is in ϑ iff:
 - (i) $\langle b_i, a, b'_i \rangle \in \varrho_i$;
 - (ii) for each $j \in \{1, \dots, n\}$, if $j \neq i$ then $b'_j = b_j$.
 - (iii) if $a \in A$ then $\langle D, a, D' \rangle \in \rho$, otherwise $D' = D$;

(i)-(ii) require that only one service \mathcal{B}_i moves from its own state b_i to b'_i performing action a , and (iii) requires that, if the action performed is not a local action, the domain evolves accordingly. Indeed, the set of system operations Γ includes operations local to services, i.e. whose execution, according to ϑ , does not affect the domain evolution. We stress the fact that a dynamic system does not correspond to any actual structure: it is a convenient representation of the interaction between the available services and the domain. Indeed, a dynamic system captures the joint execution of a dynamic domain and a set of services where, at each step, only one service moves, and possibly affects, through operation execution, the state of the underlying domain. The evolutions of a system \mathcal{S} are captured by its *histories*, hence *\mathcal{S} -histories*. One such history is a finite sequence of the form $\tau = s^0 \xrightarrow{a^1, j^1} s^1 \dots s^{\ell-1} \xrightarrow{a^\ell, j^\ell} s^\ell$ of length $|\tau| \doteq \ell + 1$ such that (i) $s^i \in S$ for $i \in \{0, \dots, \ell\}$; (ii) $s^0 = s_0$; (iii) $s^i \xrightarrow{a^{i+1}, j^{i+1}} s^{i+1}$ in \mathcal{S} , for each $i \in \{0, \dots, \ell - 1\}$. We denote with $\tau|_k$ its k -length (finite) prefix, and with \mathcal{H} the set of all possible \mathcal{S} -histories. Given a dynamic system \mathcal{S} , a *general plan* is a (possibly partial) function $\pi : \mathcal{H} \rightarrow \Gamma \times \{1, \dots, n\}$ that outputs, given an \mathcal{S} -history, a pair representing the action to be executed and the index of the service which has to execute it. An *execution* of a general plan π from a state $s \in S$ is a possibly infinite sequence $\tau = s^0 \xrightarrow{a^1, j^1} s^1 \xrightarrow{a^2, j^2} \dots$ such that (i) $s^0 = s$; (ii) $\tau|_k$ is an \mathcal{S} -history, for all $0 < k \leq |\tau|$; and (iii) $\langle a^k, j^k \rangle = \pi(\tau|_k)$, for all $0 < k < |\tau|$. When all possible executions of a general plan are finite, the plan is a *conditional plan*. The set of all conditional plans over \mathcal{S} is referred to as Π . Note that, being finite, executions of conditional plans are \mathcal{S} -histories. A finite execution τ such that $\pi(\tau)$ is undefined is a *complete execution*, which means, informally, that the execution cannot be extended further. In the following, we shall consider only conditional plans.

We say that an execution $\tau = s^0 \xrightarrow{a^1, j^1} s^1 \dots s^{\ell-1} \xrightarrow{a^\ell, j^\ell} s^\ell$ of a conditional plan π , with $s^i = \langle \langle b_1^i, \dots, b_n^i \rangle, D^i \rangle$:

- *achieves* a goal ϕ iff $D^\ell \models \phi$
- *maintains* a goal ψ iff $D^i \models \psi$ for every $i \in \{0, \dots, \ell - 1\}$

Such notions can be extended to conditional plans: a conditional plan π *achieves* ϕ from state s if all of its complete executions from s do so; and π *maintains* ψ from s if all of its (complete or not) executions from s do.

Finally, we can formally define the notion of (*goal-based*) *target process* for a dynamic domain \mathcal{D} as a tuple $\mathcal{T} = \langle T, \mathcal{G}, t_0, \delta \rangle$, where:

- $T = \{t_0, \dots, t_q\}$ is the finite set of *process states*;
- \mathcal{G} is a finite set of goals of the form *achieve ϕ while maintaining ψ* , denoted by pairs $g = \langle \psi, \phi \rangle$, where ψ and ϕ are propositional formulae over P ;
- $t_0 \in T$ is the *process initial state*;
- $\delta \subseteq T \times \mathcal{G} \times T$ is the *transition relation*. We will also write $t \xrightarrow{g} t'$ in \mathcal{P} .

A target process \mathcal{T} is a transition system whose states represent *choice points*, and whose transitions specify pairs of *maintenance* and *achievement* goals that the user can request at each step. Hence, \mathcal{T} allows to combine achievement and maintenance goals so that they can be requested (and hence fulfilled) according to a specific temporal arrangement, which is specified by the relation δ of \mathcal{T} . Intuitively, a target process \mathcal{T} is *realized* when a conditional plan π is available for the goal couple $g = \langle \phi, \psi \rangle$ chosen from initial state t_0 and, upon plan's completion, a new conditional plan π' is available for the new selected goal, and so on. In other words, all potential target requests respecting \mathcal{T} 's structure (possibly infinite) have to be fulfilled by a conditional plan, which is meant to be executed starting from the state that previous plan execution left the dynamic system \mathcal{S} in (initially from s_0). Since the sequences of goals actually chosen by the user can not be foreseen, a realization has to take into account all possible ones: at any point in time, all possible choices available in the target process must be guaranteed by the system, i.e., every legal request needs to be satisfied. We are going to give a formal definition of this intuition [9] in the remainder of this section.

Let \mathcal{S} be a dynamic system and \mathcal{T} a target process. A *PLAN-simulation relation*, is a relation $R \subseteq T \times S$ such that $\langle t, s \rangle \in R$ implies that for each transition $t \xrightarrow{\langle \psi, \phi \rangle} t'$ in \mathcal{T} , there exists a conditional plan π such that: (i) π *achieves ϕ and maintains ψ* from state s and (ii) for all π 's possible complete executions from s of the form $s \xrightarrow{\pi(\tau|_1)} \dots \xrightarrow{\pi(\tau|\ell)} s^\ell$, it is the case that $\langle t', s^\ell \rangle \in R$. A plan π *preserves R* from $\langle t, s \rangle$ for a given transition $t \xrightarrow{\langle \psi, \phi \rangle} t'$ in \mathcal{T} if requirement (ii) above holds. Also, we say that a target process state $t \in T$ is *PLAN-simulated* by a system state $s \in S$, denoted $t \preceq_{PLAN} s$, if there exists a *PLAN-simulation relation* R such that $\langle t, s \rangle \in R$. Moreover, we say that a target process \mathcal{T} is *realizable* in a dynamic system \mathcal{S} if $t_0 \preceq_{PLAN} s_0$. When the target process is realizable, one can compute *once for all (offline)* a function $\Omega : S \times \delta \rightarrow \Pi$ that, if at any point in time the dynamic system reaches state s and the process requests transition $t \xrightarrow{\langle \psi, \phi \rangle} t'$ of \mathcal{T} , outputs a conditional plan π that its execution starting from s (i) achieves ϕ while maintaining ψ and (ii) preserves \preceq_{PLAN} , i.e., it guarantees that, for all possible states the system can reach upon π 's execution, all target transitions outgoing from t' (according to δ) can still be realized by a conditional plan (possibly returned by the function itself). Such function Ω is referred to as *process realization*.

We can now formally state the problem of concern: *Given a dynamic domain \mathcal{D} , available services $\mathcal{B}_1, \dots, \mathcal{B}_n$, and a target process \mathcal{T} , build, if it exists, a realization of \mathcal{T} in the dynamic system \mathcal{S} corresponding to \mathcal{D} and $\mathcal{B}_1, \dots, \mathcal{B}_n$.* In previous work [9,10], a solution to a simplified variant of our problem has been

proposed¹. Here, as discussed above, we explicitly distinguish between dynamic domain and available services, thus obtaining a different, more sophisticated problem. Nonetheless, the techniques presented there still apply, as we can reduce our problem to that case. This allows us to claim this result:

Theorem 1 ([9]). *Building a realization of a target process \mathcal{T} in a dynamic system \mathcal{S} is an EXPTIME-complete problem.*

3 Case Study

Here we present a case study, freely inspired by a real storyboard of a live demo held in a smart home located in Rome, whose houseplant is depicted in Figure 1.

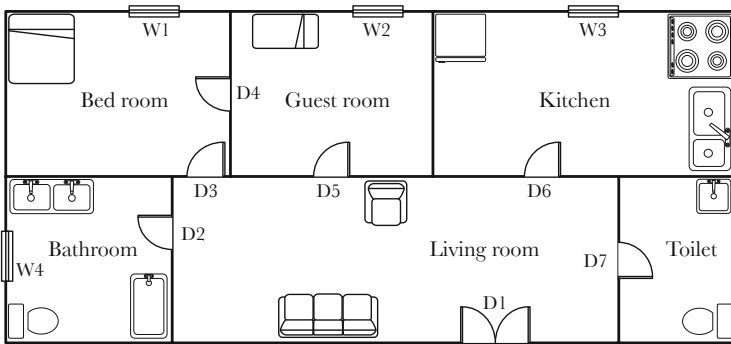


Fig. 1. The smart home plant

The home is equipped with many devices and a central reasoning system, whose domotic core is based on the framework and solver described in this paper, in order to orchestrate all the offered services. Imagine here lives Niels, a man affected by Amyotrophic Lateral Sclerosis (ALS). He is unable to walk, thus he needs a wheelchair to move around the house. The other human actors are Dan, a guest sleeping in the living room, and Wilma, the nurse. At the beginning of the story, Niels is sleeping in his automated bed.

The services the system can manage are the `bedService`, i.e., an automated bed, which can be either `down` or `up`; the `doorNumService`, i.e., the doors, for $Num \in \{1, 7\}$ (Figure 2a); the `alarmService`, i.e., an alarm, that can be either `set` or `not`; the `lightRoomService`, i.e., light bulbs and lamps, for each `Room` in Figure 1; the `kitchenService`, i.e., a cooking service with preset dishes (Figure 2c); the `pantryService`, i.e., an automated pantry, able to check whether ingredients are in or not and buy them, if missing (see Figure 2b); the `bathroomService`,

¹ In particular, in [9,10] services are modelled directly in terms of restrictions on the domain.

i.e., a bathroom management system, able to warm the temperature inside and fill or empty the tub (Figure 2e); and finally the `tvService`, i.e., a TV, either on or off.

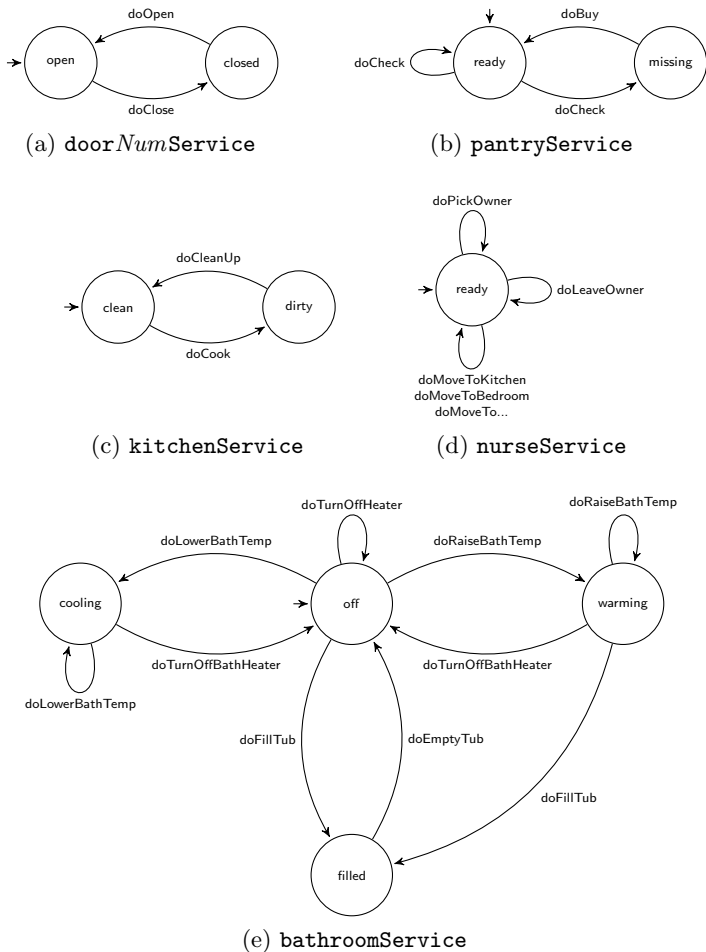


Fig. 2. Case study services

Finally, we consider a very particular service, that we call `nurseService`: it is Wilma, the nurse, who is in charge of moving Niels around the house. Despite the fact that an analogous service could be provided by some mechanical device, we refer to an human to illustrate how actors can be abstracted as services as well, wrapped by a semantic description. All services are depicted in Figure 2, except for `lightRoomService`, `bedService`, `alarmService` and `tvService` that have very simple on/off behaviors. As described in Section 2, a dynamic domain state

is a subset of 2^P , where $P = \{p_1, \dots, p_n\}$ is a finite set of boolean domain propositions. In order to express that, e.g., the bathroom temperature is mild, we could make use of a grounded propositional letter such as *bathroomTemperatureIsMild*. Nevertheless, we would have a grounded proposition for each value that the sensed temperature may assume (*bathroomTemperatureIsHot*, etc.) with the implicit constraint that only one of them can be evaluated to \top at a time (and all the others to \perp). Thus, for sake of simplicity, here we make use of statements of the form “*var = val*” (e.g., “*varBathroomTemperature = warm*”). We call *var* the domain variable; *val* can be equal to any expected value which *var* can assume. Using such abbreviations we can phrase concepts like “a domain variable *var* is set to the *val* value” to easily refer to a transition in the dynamic domain moving from the current state to a following one where the proposition *var = val* holds. For the sake of readability, actions are identified by the do-prefix (e.g., *doRing*).

Now we comment the case study. All services affect, through their actions, the related domain variables representing the state of the context. As an example, consider Figure 2e. Action *doRaiseBathTemp* causes the *bathroomService* to reach *warming* state, and affects the domain setting *varBathroomTemperature* either to (i) *mild* if it was equal to *cold*, or (ii) *warm*, if previously *mild*. However, we can imagine also *indirect* effects: e.g., the *door4Service* and *door5Service*’s *doOpen* actions trivially turn the *varDoor4* and *varDoor5* domain variables from *closed* to *open* and, at the same time, change the *varGuestDisturbed* domain variable from *false* to *true*, since, as depicted in Figure 1, they lead to the guest room, which we supposed Dan, the guest, to sleep in. The dynamic domain constrains the execution of service actions, allowing *executable* transitions only to take place (as explained in Section 2). For instance, consider the *doPick* and *doLeave* actions in *nurseService*: they represent Wilma taking and releasing Niels’ wheelchair. Even if they are always available according to the service’s description (Figure 2d), they are allowed by the domain iff *varPositionOwner* and *varPositionNurse* are equal (i.e., iff $\bigvee_{r \in Rooms} (varPositionOwner = r \wedge varPositionNurse = r)$ for $Rooms = \{livingRoom, bedRoom, bathRoom, guestRoom, toilet, kitchen\}$). Further on, it is stated that you can activate the *doPick* transition only if *varOwnerPicked = false* (conversely, activate the *doPick* only if *varOwnerPicked = true*) and, when *varOwnerPicked = true*, *doMoveToRoom* causes both *varPositionOwner* and *varPositionNurse* to be set to the same *Room*. As an example of interaction between services, consider *kitchenService* and *pantryService*. As depicted in Figure 2, they do not have any action in common. Though, cooking any dish (namely, invoking *doCook* action on the *kitchenService* service) is *not* possible if some ingredients are missing (i.e., if *varIngredients = false*). The *pantryService* can buy them (indeed, *doBuyIngredients* sets *varIngredients = true*), but only after the execution of a check (*doCheckIngredients*). The evolution of *doCheckIngredients* is constrained by the *varIngredients* domain variable: if *varIngredients = false*, then the next state of *pantryService* is missing (and the *doBuyIngredients* action *executable*), otherwise it remains in the ready state.

Such comments motivate the advantages of decoupling services and dynamic domain as in the framework: the evolution of the system is not straightforward from the inspection of services or dynamic domains alone. Indeed, a service represents the behavior of a real device or application plugged in the environment, and it is distributed by vendors who do not know the actual context in which it will be used. The same service could affect (or be affected by) the world in different ways, according to the environment it is interacting with.

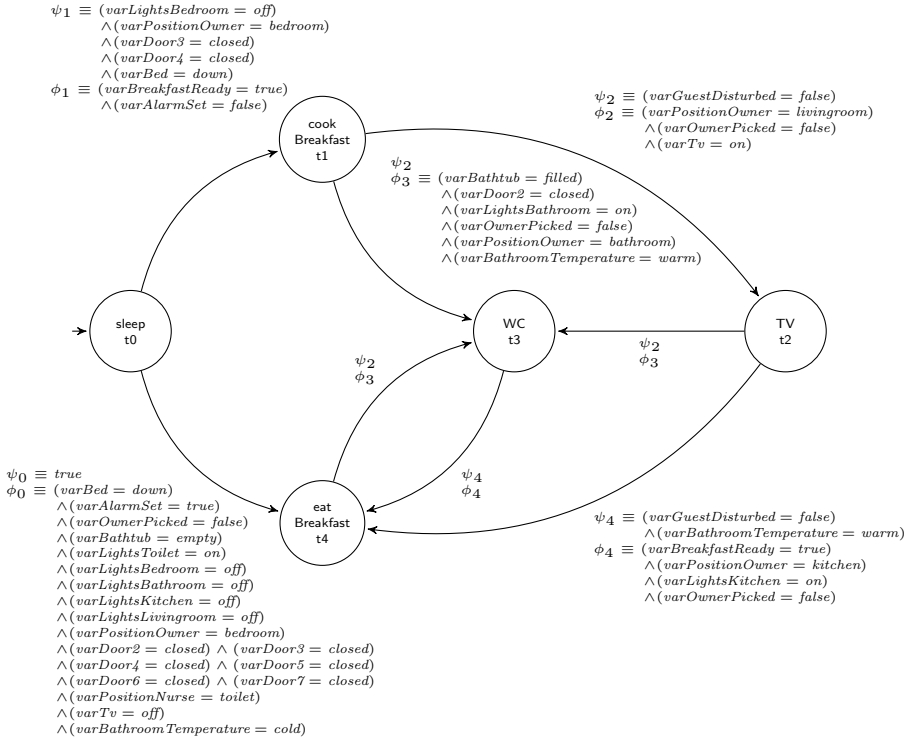


Fig. 3. The sample target process

Next we turn to the target process itself, shown in Figure 3, representing what Niels wants to happen, when waking up in the morning. First, the home system must let Niels get awoken only after the breakfast is ready: this is the aim of the first transition, where the reachability goal is to have $(\text{varBreakfastReady} = \text{true}) \wedge (\text{varAlarmSet} = \text{false})$, while all conditions that make Niels sleep comfortable must be kept: $(\text{varBed} = \text{down}) \wedge (\text{varLightsBedroom} = \text{off}) \wedge \dots$. Then, once the alarm rang out, we let Niels decide whether he prefers to have a bath or to watch TV (and optionally have a bath afterwards). In both cases, we do not want to wake up Dan ($\psi_2 \equiv (\text{varGuestDisturbed} = \text{false})$). Niels can successively have breakfast, but we suppose that further he can go back to the bathroom and eat a little more again how many times he wishes:

this is the rationale beneath the formulation of: $\psi_4 \equiv (\text{varGuestDisturbed} = \text{false}) \wedge (\text{varBathroomTemperature} = \text{warm})$. Finally, Niels can get back to the bed room. The transition from the `eatBreakfast` (`t4`) state to the `sleep` (`t0`) one has no maintenance goal (i.e., $\phi_0 = \text{true}$), whereas the reachability goal is just to reset the domain variables to their initial setup.

4 Solver

As we can see from the case study above, goal-based processes can be used to naturally specify the behavior of complex long-running intelligent systems. In order to apply this framework to real applications, however, we need a practical and efficient solver for such composition tasks. The solution in [9], [10] reduces the composition problem to LTL synthesis by model checking. As a result, an efficient model checker and the approach is viable in practice only if large computational resources are available; on typical hardware for the smart home applications, only simple examples can be solved with that approach.

In light of the success of heuristic search in classical planning, it is interesting to ask whether this problem can be more efficiently solved by a direct search method. In this paper, we pursue this idea, and propose a novel solution to the composition problem based on an AND-OR search in the space of execution traces of incremental partial policies. Intuitively, the search keeps a partial policy at each step, and simulates its execution, taking into account all possibilities of the goal requests and the nondeterministic effects of the actions. If no action is specified for some situation yet, the policy is augmented by trying all possible actions for it. Starting from an empty policy, this process is repeated until either a valid policy is found that works in all contingencies, or all policy extensions are tried yet no solution is found. In this process, the nondeterministic goal requests and action effects are handled as “AND steps,” whereas the free choice of actions during expansion represents an “OR step.” Figure 4 shows the Prolog code of the body of our solver.

The composition starts from an empty policy [] with the initial goal state and initial world state S_0 , and simulates (while incrementally building) its execution, until all possible goal requests in the target process can always be achieved by some policy C (line 2). In our implementation, we always assume that the initial goal state is 0. To handle all the possible evolutions of the system from goal state T and world state S , the `compose/4`² predicate first finds all goal requests GL that originate from T , and augments the current policy C_0 to obtain C_1 that handles these requests (lines 4–6). This is done by `compGoals/5`, which represents the first AND step in the search cycle. It recursively processes each goal request in the list GL by using `planForGoal/8` (lines 8–11). Notice that the policy is updated in each recursive step with the intermediate variable C in line 11. The predicate `planForGoal/8` essentially performs conditional planning with full observability and nondeterministic effects (lines 13–20). Lines 14 and

² According to the Prolog convention, we use the syntax `<name of the predicate>/<arity>`.

```

0: % planner.pl - a generic solver for goal-based process composition.
1: % Usage: call compose(C) to find a realization C.
2: compose(C) :- initial_state(S0), compose(0, S0, [], C).
3:
4: % compose(T, S, C0, C1) compose for goal state T.
5: compose(T, S, C0, C1) :-
6:     findall((M, G, T'), goal(T, M, G, T'), GL), !,
7:     compGoals(T, GL, S, C0, C1).
8:
9: % compGoals(T, GL, S, C0, C1) compose for all goal requests in GL.
10: compGoals(_, [], _, C, C).
11: compGoals(T, [(M, G, T')|GL], S, C0, C1) :-
12:     planForGoal(T, M, G, T', S, [], C0, C),
13:     compGoals(T, GL, S, C, C1).
14:
15: % planForGoal(T, M, G, T', S, H, C0, C1)
16: % update policy for a specific goal.
17: planForGoal(_, _, _, _, S, H, _, _) :- member(S, H), !, fail.
18: planForGoal(_, M, _, _, S, _, _, _) :- \+ holds(M, S), !, fail.
19: planForGoal(T, _, _, T', S, _, C, C) :- member((T, T', S, _), C), !.
20: planForGoal(_, _, G, T, S, _, C0, C1) :- holds(G, S), !,
21:     compose(T, S, C0, C1).
22: planForGoal(T, M, G, T', S, H, C0, C1) :-
23:     bestAct(G, A, S), next_states(S, A, SL),
24:     tryStates(T, M, G, T', SL, [S|H], [(T, T', S, A)|C0], C1).
25:
26: % tryStates(T, M, G, T', SL, H, C0, C1)
27: % compose for all progressed world states.
28: tryStates(_, _, _, _, [], _, C, C).
29: tryStates(T, M, G, T', [S|SL], H, C0, C1) :-
30:     planForGoal(T, M, G, T', S, H, C0, C),
31:     tryStates(T, M, G, T', SL, H, C, C1).

```

Fig. 4. Prolog implementation of our search-based solver

15 prevent the found partial policy from containing deadlocks or violating the maintenance goal. Line 16 detects visited states in achieved goals so that they can be realized in the same way, and thus no further search is needed. Line 17 checks whether the current achievement goal has been realized, and if so, it goes on to recursively compose for the next goal state. Finally, Lines 18–20 capture the last case where no action is associated to the current situation, in which case the current policy needs expansion to handle it. This is done by the OR step of the search cycle, which proposes a best candidate action with the predicate `bestAct/3`, and planning goes on for the resulting world states.

Since the actions are nondeterministic, it means that executing an action may lead to multiple possible states, and the policy we find must work for them all. In our algorithm, `tryStates/8` handles all these states by recursing into `planForGoal/8` with updated policy for each state, which represents the second AND step in the search cycle (lines 22–25).

Recall that the exploration of a search branch may fail in Lines 14 and 15, due to a deadlock and violation of a maintenance goal, respectively. When either case occurs, the program backtracks to the most recent predicate with a different succeeding assignment, which is always `bestAct/3`. From there, the next best action is proposed and tried, and so on. If all the possible actions have been tried, yet none leads to a valid policy, the program backtracks to the next most recent `bestAct/3` instance, and the same process is performed similarly.

Notice that our algorithm is applicable to any goal-based process composition task, as the predicates `initial_state/1`, `holds/2`, `bestAct/3`, `next_states/3` and `goal/4` behave according to the actual target goal-based process and its underlying dynamic environment which are specified using a problem definition language detailed below. It is not hard to see that the our algorithm strategically enumerates all valid policies, generating *on-the-fly* action mappings for *reachable* situations only. The algorithm can be shown to be sound and complete.

Theorem 2 (Soundness and completeness). *Let \mathcal{T} be a target goal-based process and \mathcal{S} its underlying dynamic system. If `compose(C)` succeeds, then \mathcal{C} is a realization of \mathcal{T} in \mathcal{S} . Moreover, if \mathcal{T} is realizable in \mathcal{S} , then `compose(C)` succeeds.*

This algorithm can be used for solving small composition problems even with a simple enumeration-based implementation of `bestAct/3`. However, as the problem size grows, this naïve implementation quickly becomes intractable, due to the large branching factor and deep search tree. Therefore, some intelligent ordering is needed for the succeeding bindings of `bestAct/3`, in order to make our solver efficient for large composition tasks. In our implementation of the solver³,

³ The code of both solver and case study, as well as the experimental results, are available at the URL: http://www.dis.uniroma1.it/~cdc/pubs/CoopIS2012-Code_Tests.zip

we make use of the well-known *delete-relaxation heuristics* [13], although other heuristics in classical planning could be adapted as well.

The delete-relaxation heuristics for a state is computed by solving a relaxed goal-reachability problem where all negative conditions and delete effects are eliminated from the original planning problem. It can be shown that the relaxed problem can always be solved (or proven unsolvable) in polynomial time. If a relaxed plan is found, then the number of actions in the plan is used as a heuristic estimation for the cost of achieving the goal from the current state; otherwise it is guaranteed that no plan exists to achieve the goal from the current state, so it is safe to prune this search branch and backtrack to other alternatives. In our implementation, when choosing the best action, `bestAct/3` first sorts all legal actions according to the optimistic goal distance of their successor states using the delete-relaxation heuristics⁴, and unifies with each of the actions in ascending order when the predicate is (re-)evaluated. Notice that the heuristics only changes the ordering of branch exploration in the search tree, with possible sound pruning for deadends, and does not affect the correctness guarantee of our algorithm.

For our solver, a problem specification is a regular Prolog source file which contains the following components:

- the instruction to load the solver `:- include(planner).`
- a list of primitive fluents, each fluent `F` specified by `prim_fluent(F).`
- a list of primitive actions, each action `A` specified by `prim_action(A).`
- action preconditions, one for each action `A`, by `poss(A,P).` where `A` is the action, and `P` is its precondition formula.
- conditional effects of actions of the form `causes(A,F,V,C).` meaning that fluent `F` will take value `V` if action `A` is executed in a state where condition `C` holds.
- initial assignment of fluents of the form `init(F,V).` where `F` is the fluent and `V` is its initial value.
- the process by a list of goal transitions of the form `goal(T,M,G,T').` where `T` and `T'` are the source and target goal states of the transition, `M` is the maintenance goal, and `G` the achievement goal. By default, the initial goal state is always 0.

5 Experiments on the Case Study

In order to test the efficiency of the solution presented in Section 4, we conducted some experiments based on the case study of Section 3.

Given the dynamic system \mathcal{S} and the target process \mathcal{T} described in Section 3, we considered both \mathcal{T} and its restrictions $\mathcal{T}_{i \in \{1, \dots, 5\}}$, shown in Figure 5, where states and goals refer to the ones depicted in Figure 3.

⁴ In our experiments, we also take into account the conjunction of the maintenance goals of the next goal state, so that states violating future maintenance goals are pruned earlier, leading to further gain in efficiency.

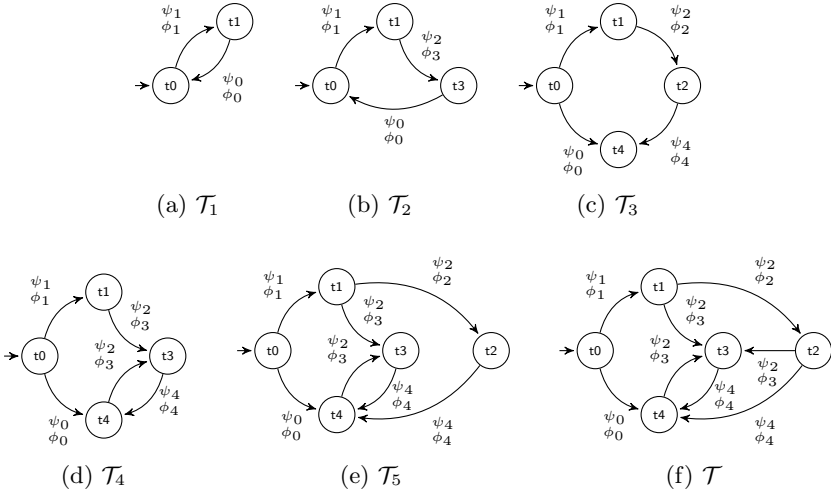


Fig. 5. Test target processes

	Trans.'s	Time [sec]			Std. Dev. (σ)	Coeff. of Var. [%] (σ/\mathcal{M})
		Mean (\mathcal{M})	min (m)	Max (M)		
\mathcal{T}_1	2	1.166	1.15	1.19	0.011	9.219
\mathcal{T}_2	3	60.688	60.54	60.82	0.094	1.545
\mathcal{T}_3	4	30.448	30.39	30.61	0.088	2.896
\mathcal{T}_4	5	83.497	83.26	83.88	0.195	2.331
\mathcal{T}_5	7	180.894	180.29	182.05	0.523	2.889
\mathcal{T}	8	238.841	238.38	239.19	0.291	1.219

Fig. 6. Test results

Hence, for each \mathcal{T}_i , we run the solver 10 times in a SWI-Prolog 5.10.2 environment, on top of an Intel Core Duo 1.66 GHz (2 GB DDR2 RAM, Ubuntu 10.04) laptop.

We gathered the results listed in Figure 6. The solution for the complete problem was found in about 239 seconds. Performances for simpler formulations followed an almost linear trend with respect to the input dimension, measured in terms of number of transitions in the target process (see Figure 7a). Figure 7b⁵ shows that such results are quite reliable, since the Coefficient of Variation (i.e., the ratio between the Standard Deviation σ and the Mean Value \mathcal{M}) is fair little (ca. 2%, excluding the first value, which is not significant, being the solution for that instance computed in too few milliseconds) and keeps constant as the \mathcal{M} value grows. The performances are notable, especially if compared to the previous tests. Indeed, we ran a solver based on model checking techniques, built on top of TLV (version 4.18.4, see [20]), on the laptop mentioned above.

⁵ There, the base for the Logarithm of the Mean Time \mathcal{M} is the least \mathcal{M} value.

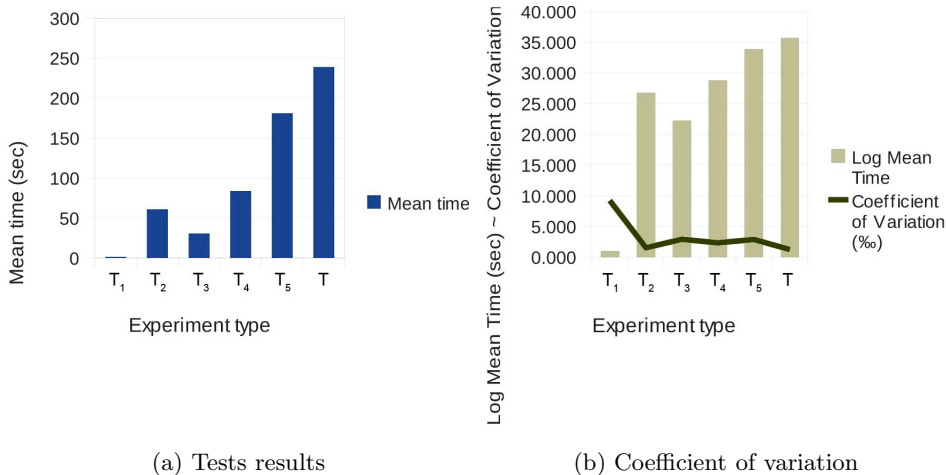


Fig. 7. Graphical representation of the test results

Notice that such a solver requires the usage of high computational resources, which are not affordable in a smart home scenario. Indeed, on our laptop it took more than 24 hours to terminate, whereas the solver presented in this paper is returned a solution within less than 4 minutes.

6 Conclusions

In this paper we have proposed an approach for composing goal-based processes on the basis of available services, which stems from the real needs of a smart home scenario, in which available services are based on sensors, actuators and equipments of the home. The approach and the solver we develop turned out to be effective in practice as the case study and the experiments demonstrate. Future work include the investigation of the case of multiple users (e.g., all inhabitants of the home) asking for different simultaneous goal-based processes. Preliminary ideas can be found in [21,10].

Acknowledgements. This work has been partly supported by the EU projects SM4All (FP7-224332), ACSI (FP7-257593) and Greener Buildings (FP7-258888), and by the Italian AriSLA project Brindisys. Yuxiao Hu would like to thank his PhD supervisor Hector Levesque for useful discussions and financial support.

References

1. Baligand, F., Rivierre, N., Ledoux, T.: A Declarative Approach for QoS-Aware Web Service Compositions. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSSOC 2007. LNCS, vol. 4749, pp. 422–428. Springer, Heidelberg (2007)

2. Beauche, S., Poizat, P.: Automated Service Composition with Adaptive Planning. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSSOC 2008. LNCS, vol. 5364, pp. 530–537. Springer, Heidelberg (2008)
3. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Formal Methods for Service Composition. *Annals of Mathematics, Computing and Teleinformatics* 1(5), 1–10 (2007)
4. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Service Composition based on Behavioural Descriptions. *International Journal of Cooperative Information Systems* 14(4), 333–376 (2005)
5. Blythe, J., Ambite, J. (eds.): Proc. of ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services (2004)
6. Cardoso, J., Sheth, A.P.: Introduction to Semantic Web Services and Web Process Composition. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 1–13. Springer, Heidelberg (2005)
7. Catarci, T., Di Ciccio, C., Forte, V., Iacomussi, E., Mecella, M., Santucci, G., Tino, G.: Service Composition and Advanced User Interfaces in the Home of Tomorrow: The SM4All Approach. In: Gabrielli, S., Elias, D., Kahol, K. (eds.) AMBI-SYS 2011. LNICST, vol. 70, pp. 12–19. Springer, Heidelberg (2011)
8. Curbera, F., Sheth, A., Verma, K.: Services Oriented Architectures and Semantic Web Processes. In: ICWS 2004 (2004)
9. De Giacomo, G., Patrizi, F., Sardiña, S.: Agent Programming via Planning Programs. In: AAMAS 2010 (2010)
10. De Giacomo, G., Felli, P., Patrizi, F., Sardiña, S.: Two-player Game Structures for Generalized Planning and Agent Composition. In: AAAI 2010 (2010)
11. Di Ciccio, C., Mecella, M., Caruso, M., Forte, V., Iacomussi, E., Rasch, K., Querzoni, L., Santucci, G., Tino, G.: The Homes of Tomorrow: Service Composition and Advanced User Interfaces. *ICST Trans. Ambient Systems* 11(10-12) (2011)
12. Hassen, R.R., Nourine, L., Toumani, F.: Protocol-Based Web Service Composition. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSSOC 2008. LNCS, vol. 5364, pp. 38–53. Springer, Heidelberg (2008)
13. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research* 14, 253–302 (2001)
14. Kaldeli, E., Warriach, E.U., Bresser, J., Lazovik, A., Aiello, M.: Interoperation, Composition and Simulation of Services at Home. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSSOC 2010. LNCS, vol. 6470, pp. 167–181. Springer, Heidelberg (2010)
15. Klein, A., Ishikawa, F., Honiden, S.: Efficient QoS-Aware Service Composition with a Probabilistic Service Selection Policy. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSSOC 2010. LNCS, vol. 6470, pp. 182–196. Springer, Heidelberg (2010)
16. McIlraith, S., Son, T.: Adapting GOLOG for Composition of Semantic Web Services. In: KR 2002 (2002)
17. Medjahed, B., Bouguettaya, A., Elmagarmid, A.: Composing Web Services on the Semantic Web. *Very Large Data Base Journal* 12(4), 333–351 (2003)
18. De Paoli, F., Lulli, G., Maurino, A.: Design of Quality-Based Composite Web Services. In: Dan, A., Lamersdorf, W. (eds.) ICSSOC 2006. LNCS, vol. 4294, pp. 153–164. Springer, Heidelberg (2006)
19. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: IJCAI 2005 (2005)
20. Pnueli, A., Shahar, E.: The TLV System and its Applications. Tech. rep., Department of Computer Science, Weizmann Institute, Rehovot, Israel (1996)

21. Sardiña, S., De Giacomo, G.: Realizing Multiple Autonomous Agents through Scheduling of Shared Devices. In: ICAPS 2008 (2008)
22. Schuller, D., Miede, A., Eckert, J., Lampe, U., Papageorgiou, A., Steinmetz, R.: QoS-Based Optimization of Service Compositions for Complex Workflows. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 641–648. Springer, Heidelberg (2010)
23. Wang, H., Zhou, X., Zhou, X., Liu, W., Li, W., Bouguettaya, A.: Adaptive Service Composition Based on Reinforcement Learning. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 92–107. Springer, Heidelberg (2010)
24. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.S.: Automating DAML-S Web Services Composition Using SHOP2. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 195–210. Springer, Heidelberg (2003)
25. Yang, J., Papazoglou, M.: Service Components for Managing the Life-cycle of Service Compositions. *Information Systems* 29(2), 97–125 (2004)
26. Zhao, H., Doshi, P.: A Hierarchical Framework for Composing Nested Web Processes. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 116–128. Springer, Heidelberg (2006)

Automated Risk Mitigation in Business Processes

Raffaele Conforti¹, Arthur H.M. ter Hofstede^{1,2,3},
Marcello La Rosa^{1,2}, and Michael Adams¹

¹ Queensland University of Technology, Australia
{raffaele.conforti, a.terhofstede, m.larosa, mj.adams}@qut.edu.au

² NICTA Queensland Lab, Australia

³ Eindhoven University of Technology, The Netherlands

Abstract. This paper proposes a concrete approach for the automatic mitigation of risks that are detected during process enactment. Given a process model exposed to risks, e.g. a financial process exposed to the risk of approval fraud, we enact this process and as soon as the likelihood of the associated risk(s) is no longer tolerable, we generate a set of possible mitigation actions to reduce the risks' likelihood, ideally annulling the risks altogether. A mitigation action is a sequence of controlled changes applied to the running process instance, taking into account a snapshot of the process resources and data, and the current status of the system in which the process is executed. These actions are proposed as recommendations to help process administrators mitigate process-related risks as soon as they arise. The approach has been implemented in the YAWL environment and its performance evaluated. The results show that it is possible to mitigate process-related risks within a few minutes.

1 Introduction

Business processes in various sectors such as financial, healthcare and oil&gas, are constantly exposed to a wide range of risks. Take for example the BP oil spill in 2010 which resulted in an environmental disaster, or the fraud at Société Générale in 2008, which led to a €4.9B loss.

A *process-related risk* measures the likelihood and the consequence that something happening will impact on the process objectives [29]. Failing to address process-related risks can result in substantial financial and reputational consequences, potentially threatening an organization's existence, like in the case of Société Générale. There is thus an increasing need to better manage business process risks, as also highlighted by legislative initiatives like Basel II [7] and the Sarbanes-Oxley Act.¹ Organizations are attempting to incorporate process-related risks as a distinct view in their operational management, seeking effective ways to *control* such risks. However, whilst conceptually appealing, to date there is little guidance as to how this can be concretely done.

In previous work [12], we presented a mechanism to model risks in executable business process models and detect them as early as possible during process execution. Unfortunately, detecting a risk in time is often not enough to avoid the negative outcome associated. A *prompt* risk mitigation should be taken to restore the process instance to

¹ www.gpo.gov/fdsys/pkg/PLAW-107publ204

a safe state, before the instance progresses any further. Moreover, taking the *right* mitigation at the right time may make the difference between success and failure. In fact, the number of possible ways a process-related risk may be mitigated is potentially very large that it is difficult for a process administrator to take the right decision at the right time, without any support. One has to consider all mitigations that are possible, given the current state of the process instance (including a snapshot of the associated data and resources), and the context in which the instance is running, i.e. the state of other running instances, to make such a decision. For example, in order to mitigate the risk of a process instance A to run overtime, a mitigation may entail to reallocate resources from a process instance B (potentially of another process) to A.

In light of the above, in this paper we propose a technique for automatically mitigating process-related risks. Since a process instance may be affected by multiple risks at the same time, we treat this problem as a *multi-objective optimization problem*. A solution to this problem is a variant of the risky process instance obtained by applying a sequence of mitigation actions, in order to reduce the risks' probability down to a tolerable level, or in the best case, to annul the risks altogether. Mitigation actions include control-flow aspects (e.g. skipping a task to be executed), process resources (e.g. reallocating a resource to a different task), and data (e.g. rolling back an executed task to restore its input data). To explore the potentially large solution space, we use dominance-based Multi-Objective Simulated Annealing (MOSA) [28]. At each run, the algorithm generates a small set of solutions similar to the original process instance but with less risks. It stops when either a maximum number of *non-redundant* solutions (i.e. solutions proposing different mitigations) is found or a given timeframe elapses. This approach is not meant to replace human judgement. Instead, it aims to support process administrators in deciding what mitigations to take, by reducing the number of feasible options, and consequently the time needed to take a decision.

We defined the mitigation actions in collaboration with an Australian risk consultant. To prove the feasibility of this approach, we implemented these actions and the MOSA algorithm on top of the YAWL system. We instantiated a set of process models from the logistics [34] and screen business [25] domains that are affected by one or more risks, and executed a series of tests to mitigate such risks. The tests show that the technique can find a set of possible solutions within a few minutes of computation, and that in all cases the associated risks are mitigated.

The rest of this paper is organized as follows. Section 2 introduces the required background concepts in the context of an example. Section 3 describes the proposed technique to mitigate process risks which is then evaluated in Section 4. Section 5 covers related work and Section 6 concludes the paper.

2 Background and Running Example

Our technique for risk mitigation is part of a holistic approach for managing process-related risks throughout the process lifecycle. Accordingly, the four phases of the traditional BPM lifecycle (Design, Implementation, Enactment and Analysis) [16] are each extended to incorporate elements of risk, as shown in Fig. 1. First, in a *Risk Identification* phase, the process model to be designed is analyzed for potential risks. Established risk analysis methods such as Fault Tree Analysis [11] or Root Cause Analysis [21]

can be employed in this phase. The output of this phase is a set of risks, each expressed as a *risk condition* that describes the set of events that lead to a potential fault occurrence. Then, in the Design phase, these high-level risk conditions are mapped to process model-specific aspects. The result of this phase is a risk-annotated process model. Next, in the Implementation phase, these conditions are linked to workflow-specific aspects, such as the content of data variables and resource allocation states. The model is then executed by a risk-aware process engine in the Enactment phase.

To evaluate risk conditions in this phase, we need to consider the current state of all running instances of any process (and not only the instance for which we are computing the risk condition), the resources that are busy and available, and the values of the data variables being created and consumed. Moreover, we need to consider historical data, i.e. the archived execution data of all previous instances of the process. Finally, the Diagnosis phase involves risk monitoring and controlling, which can trigger changes in the current process instance, to *mitigate* the likelihood of a fault occurring, or in the underlying process model, to *prevent* a given risk from occurring in future instances. This risk mitigation phase is the focus of this paper.

Let us now consider an example process for which we have defined several risks, to understand how risk conditions can be formulated in terms of process model elements. These conditions will provide input for the risk mitigation technique presented in the next section. The example process, shown in Figure 2, describes a *Payment* subprocess of an order fulfillment process, inspired by the VICS industry standard for logistics [34]. This standard is endorsed by 100+ companies worldwide, with a total sales volume of \$2.3 Trillion annually [34]. The example process begins after freight has been picked up by a carrier and deals with the payment of shipment costs. First, a Shipment Invoice is produced for costs related to a specific order. If payment has been made in advance, a Finance Officer simply issues a Shipment Remittance Advice to the customer specifying the amount paid. Otherwise, the Finance Officer issues a Shipment Payment Order, which requires approval by a Senior Finance Officer (a superior of the Finance Officer) who may request amendments be made by the Finance Officer that issued the Order. After the document is finalized and the customer has paid, an Account Manager can process the payment. If the customer underpays, the Account Manager issues a Debit Adjustment, the customer makes a further payment and the payment is reprocessed. If a customer overpays, the Account Manager issues a Credit Adjustment. In the latter case and in the case of correct payment, the Payment subprocess completes.

In this process, we can identify various faults that may occur during execution. For example, a Service Level Agreement (SLA) may establish that the process (or one of its tasks) may not last longer than a Maximum Cycle Time *MCT* (e.g. 5 days), otherwise

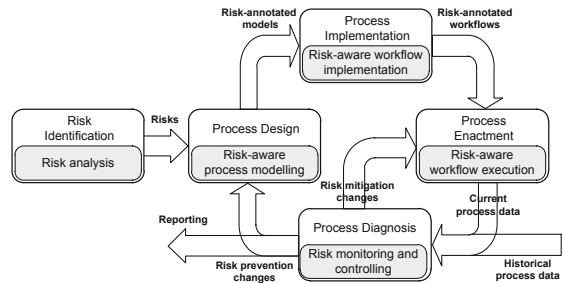


Fig. 1. Risk-aware BPM lifecycle

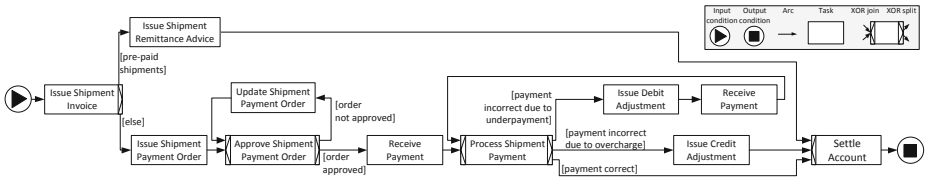


Fig. 2. Order-Fulfillment: Payment subprocess (using the YAWL [19] notation)

a pecuniary penalty may be incurred. 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_c expired to that point. 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 based on past executions, which can be computed using the approach in [2]. 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.

A 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 a potential for *approval fraud*, a violation of a four-eyes principle across different instances of the Payment subprocess. To detect this risk we first have to check that there is an order, say order o of customer c , to be approved. 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 customer c 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 did not approve an order for c during the last d days are unavailable.

Finally, a third fault relates to a situation where a process instance executes a given task too many times, typically via a loop. Not only could this lead to a process slow-down, but also to a “livelock” if the task is in a loop whose exit condition is deliberately never met. In general, given a task t , a maximum number of allowable executions of t per process instance, $MAE(t)$, can be fixed as part of the service-level agreement for t . In our example, this fault may occur 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 at run-time, we need to check if: i) the Update task is currently being performed for order o ; and ii) it is likely that the task will be repeated within the same process instance. The probability that the number of times a task will be repeated within the same instance is computed by dividing the number of instances where the *MAE* for the task has been reached by 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. If the probability to exceed $MAE(t)$ is greater than a tolerance value for t , e.g. 60%, we notify the risk to the user.

In the next section we propose a set of mitigation actions that can be performed “on-the-fly” on a running process instance in order to mitigate its risks.

3 Approach

In this paper we deal with the problem of automatically mitigating one or more business process risks for a specific running process instance (*case* for short), without raising other business process risks for the same case. This problem belongs to the family of multi-objective optimization problems, and we propose the use of simulated annealing for finding a Pareto-optimal solution, or a set of such solutions.

The Process Risk Simulated Annealing (PRSA) algorithm is an application of the DBMOSA [28] algorithm where at each iteration a new solution is discovered through the use of one or more random mitigation actions. The algorithm proposes a solution, or mitigation, as a sequence of elementary mitigation actions. A “behavioral cost” (cost for short) is associated with each mitigation action, and measures how deeply an action affects the process instance to which it is applied. For example, allocating a different resource to a work item has a lower cost than skipping a task that has to be executed. The total cost of a solution is the sum of the costs of each mitigation action used in that solution. A good solution to the PRSA algorithm is one that reduces the likelihood of a risk under its threshold, keeping the total cost as low as possible.

When comparing solutions that have the same cost, a solution that fully mitigates a risk is better than one that mitigates that risk because its risk condition is no longer evaluable. And in turn, this solution is better than one that does not mitigate the risk at all. Finally, if two solutions mitigate the same risk, we privilege the one that yields the lowest risk probability. Given two solutions a, b we say that a dominates b if it mitigates the same risks mitigated by b with a lower total cost. As result, we define them as mutually non-dominating if neither one dominates the other.

Below we describe the more elementary mitigation actions that can be used to create a solution, and how they affect a process case. We use the YAWL language as a reference language to define the mitigation actions, since this language has a formal foundation on which we can build our definitions and algorithms. However, the notions presented in this section can easily be generalized to other languages. Before introducing them, we introduce a number of preliminary concepts and notations.

YAWL Specification. We will not repeat the full definition of a YAWL specification as defined in [19], we will only use selected parts. The set of net identifiers is given by $NetID$ and the process identifier is the net identifier of the root net, $ProcessID \in NetID$. Furthermore, each net has, among others, a set of conditions C , an input condition $i \in C$, an output condition $o \in C$, and a set of tasks T and there is a flow relation $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$.

We use the following auxiliary functions from [19]. The pre-set of x is defined as $\bullet x = \{y \in C \cup T \mid (y, x) \in F\}$ and the post-set of x is defined as $x \bullet = \{y \in C \cup T \mid (x, y) \in F\}$. We also introduced other auxiliary functions. The set of tasks that directly or through a place precedes a task t s is referred to as the *task pre-set* of t and is defined as $\circ t = \{x \in T \mid x \in \bullet t \vee \exists y \in C[y \in \bullet t \wedge x \in \bullet y]\}$. Similarly, the *task post-set* of t is defined as $t \circ = \{x \in T \mid x \in t \bullet \vee \exists y \in C[y \in t \bullet \wedge x \in y \bullet]\}$. Finally, to detect all the successors of a task, it is defined as $t \circ^*$ and it is the transitive closure of $t \circ$.

Following the convention in [19], we write e.g. T_n to access the tasks of net n . Moreover, for a YAWL specification y , T_y is the set of tasks that occur in any of its

nets, i.e. $T_y = \cup_{n \in \text{NetID}} T_n$, and for a set of YAWL specifications Y , T_Y is the set of tasks that occur in any of the nets of any of the specifications, i.e. $T_Y = \cup_{y \in Y} T_y$.

In our context we have only one Organizational model [19] and what is relevant for us is the set of resources, UserID , to whom work items can be assigned. Finally, we defined the set of *skippable* tasks as $\{t \in T_Y \mid \exists r \in \text{UserID}[\text{skip} \in \text{UserTaskPriv}(r, t)]\}$.

The set *StatusType* contains the various statuses that a work item may go through during its lifecycle. These are: *offered*, *allocated*, *started*, *completed*, *forceCompleted*, *cancelled*, *failed*, *deadlocked* used by the YAWL system and additionally *deoffered*, *deallocated*, *destarted*, *rollback*, *skipped* used for mitigation purposes. Many of these statuses are self-explanatory. The status *rollback* is the status of a work item which was completed but then enabled again though not *offered*. The status *skipped* is the status of a work item that was skipped, which is similar to the status *completed* but the work item was not actually performed. For convenience, we provide certain groupings of event types. In particular, $\text{Rel} \triangleq \text{StatusType} \setminus \{\text{cancelled}, \text{failed}, \text{rollback}\}$ is the set of event types that identify a work item as subject to mitigation. $\text{Active} \triangleq \{\text{offered}, \text{allocated}, \text{started}\}$ is the set of event types that mark a work item as in progress, $\text{Completed} \triangleq \{\text{completed}, \text{forceCompleted}\}$ is the set of event types that mark a work item as completed, and $\text{ActiveC} : \text{Active} \cup \text{Completed}$ is their union.

Given set *ActiveC* we define a partial order $\subseteq \preceq \text{ActiveC} \times \text{ActiveC}$ such that it preserves the partial ordering $\text{deoffered} < \text{offered} < \text{allocated} < \text{started} < \text{completed} = \text{forceCompleted}$.

Definition 1 (Log). *In the context of a set of YAWL specifications Y , with associated set of tasks T_Y and a set of root nets \mathcal{R} , a log is defined as $L = (\mathcal{E}, \mathcal{W}, \mathcal{C}, \text{Model}, \text{WI}, \text{Case}, \text{Task}, \text{EvType}, \text{Time}, \text{Res}, \text{Inp}, \text{Outp})$ where:*

- \mathcal{E} is a set of events,
- \mathcal{W} is a set of work items,
- \mathcal{C} is a set of case identifiers,
- $\text{Model} : \mathcal{C} \rightarrow \mathcal{R}$ is a function relating cases to the root nets of the associated YAWL specification,
- $\text{WI} : \mathcal{E} \rightarrow \mathcal{W}$ is a surjective function relating events to work items,
- $\text{Case} : \mathcal{E} \rightarrow \mathcal{C}$ is a surjective function relating events to cases,
- $\text{Task} : \mathcal{W} \rightarrow T_Y$ is a function relating work items to tasks,
- $\text{EvType} : \mathcal{E} \rightarrow \text{StatusType}$ is a function relating events to work item statuses,
- $\text{Time} : \mathcal{E} \rightarrow \mathcal{T}$ is an injective function relating events to timestamps, hence no two events in the log can have identical timestamps,
- $\text{Res} : \mathcal{E} \rightarrow 2^{\text{UserID}}$ is a function relating events to sets of resources, as some events may concern multiple resources (e.g. a work item being offered),
- $\text{Inp} : \mathcal{E} \times \text{Var} \rightarrow \Omega$ is a partial function relating events and variables to (input) values,
- $\text{Outp} : \mathcal{E} \times \text{Var} \rightarrow \Omega$ is a partial function relating events and variables to (output) values.

Definition 2 (Event Comparison). *Let L be a log, given $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{E}' \neq \emptyset$, we define the operators $e_1 < e_2$ iff $\text{Time}(e_1) < \text{Time}(e_2)$ and $e_1 \leq e_2$ iff $\text{Time}(e_1) \leq \text{Time}(e_2)$,*

which reflect the temporal ordering on events, and the operators $\min \mathcal{E}' = e_1$ iff $e_1 \in \mathcal{E}'$ and for all $e_2 \in \mathcal{E}'$, $e_1 \leq e_2$, which determines the earliest event of an event set, and $\max \mathcal{E}' = e_1$ iff $e_1 \in \mathcal{E}'$ and for all $e_2 \in \mathcal{E}'$, $e_2 \leq e_1$, which determines the latest event of an event set.

Useful is the possibility of identifying events belonging to the same work item.

Definition 3 (Work Item Event Grouping). Let L be a log, e an event in this log, $e \in \mathcal{E}$, and w a work item in this log, $w \in \mathcal{W}$, we define the set of events that belong to work item w as $\overline{WI}(w) \triangleq \{e \in E \mid WI(e) = w\}$. Similarly, we define the set of events that belong to the same work item of e as $\overline{WI}(e) \triangleq \overline{WI}(WI(e))$. Finally, the latest event for work item w is defined as $\omega_w \triangleq \max \overline{WI}(w)$.

As for events we are interested in being able to compare work items.

Definition 4 (Work Item Comparison). Let L be a log, with $w_1, w_2 \in \mathcal{W}$, we define $w_1 < w_2$ as $\max \overline{WI}(w_1) < \min \overline{WI}(w_2)$. This operator reflects the partial temporal order between work items, i.e. work item w_1 precedes work item w_2 if its latest event is earlier than the earliest event of w_2 .

An execution graph for a process case provides a view of its execution and is defined on the basis of a log and its corresponding process model.

Definition 5 (Execution Graph). Let L be a process log with case c , Y its YAWL specification, and $UserID$ the set of resources, we define the execution graph of c as $G(c) = (Node, NodeTask, Status, \rightsquigarrow, NodeRes, TimeNode, VarNode)$ where:

- $Node = \{w \in \mathcal{W} \mid EvType(\omega_w) \in Rel \wedge Case(w) = c\}$ is the set of nodes, where each node represents a work item that is not modifiable,
- $NodeTask = Task|_{Node}$ is the restriction of the function $Task$ to the set of nodes,
- $Status = \{(\omega_w, s) \in Node \times Rel \mid s = EvType(\omega_w)\}$ is a function relating a node with its status of execution,
- $\rightsquigarrow = \{(w_1, w_2) \in Node \times Node \mid Status(w_1) \in \{completed, skip\} \wedge NodeTask(w_1) \in \circ NodeTask(w_2) \wedge \nexists w_3 \in Node[(NodeTask(w_1) = NodeTask(w_3) \vee NodeTask(w_2) = NodeTask(w_3)) \wedge w_1 < w_3 \wedge w_3 < w_2]\}$ is the flow relation between work items. Its reflexive transitive closure is defined as \rightsquigarrow^* ,
- $NodeRes = \{((w, s), r) \in (Node \times Active) \times 2^{UserID} \mid \exists e_1 \in \overline{WI}(w)[EvType(e_1) = s \wedge r = Res(e_1) \wedge \nexists e_2 \in \overline{WI}(w)[e_1 < e_2 \wedge EvType(e_2) \leq s]]\}$ is a function that yields the resources that are involved in the latest changing w to status s ,
- $TimeNode = \{((w, s), t) \in (Node \times ActiveC) \times \mathcal{T} \mid \exists e_1 \in \overline{WI}(w)[EvType(e_1) = s \wedge t = Time(e_1) \wedge \nexists e_2 \in \overline{WI}(w)[e_1 < e_2 \wedge EvType(e_2) \leq s]]\}$ is a partial function that yields the timestamp when w latest moved to status s ,
- $VarNode = \{((w, x), v) \in (Node \times Var) \times \Omega \mid EvType(\omega_w) \notin \{skip, deoffered\} \wedge v = Inp(\max \{e_2 \in \overline{WI}(w) \mid EvType(e_2) = offered\}, x)\} \oplus \{((w, x), v) \in (Node \times Var) \times \Omega \mid EvType(\omega_w) \in Completed \wedge v = Outp(\omega_w, x)\}$ is a partial function relating nodes and variables to values.

As we explore mitigation options the execution graph should evolve along with it, and the initial execution graph becomes a dynamic data structure from which we can modify nodes. We will refer to this modified execution graph as *mitigation graph*.

The concept of *border* identifies work items that can be modified. Such work items are currently in execution, or they are completed work items for which there are no successor work items that are completed or being executed.

Definition 6 (Border). Let G be a mitigation graph. We define the border of G , \circ_G , as $\{n_1 \in \text{Node} \mid \forall n_2 \in \text{Node}[n_1 \rightsquigarrow^* n_2 \Rightarrow \text{Status}(n_2) \in \{\text{deoffered}, \text{skipped}, \text{rollback}\}]\}$.

Definition 7 (Mitigations). Let Y be a set of YAWL specifications, with associated set of tasks T_Y , a set of resources UserID , and a log L . A mitigation is represented as $M = (\mathcal{A}, \text{AcType}, \text{AcTask}, \text{AcRes}, \text{AcCase}, \succ)$ where:

- \mathcal{A} is a set of mitigation actions,
- $\text{AcType} : \mathcal{A} \rightarrow \{\text{deoffer}, \text{deallocate}, \text{destart}, \text{offer}, \text{allocate}, \text{start}, \text{rollback}, \text{skip}\}$, is a function relating actions to types of mitigation,
- $\text{AcTask} : \mathcal{A} \rightarrow T_Y$ is a function relating actions to tasks,
- $\text{AcRes} : \mathcal{A} \rightarrow \text{UserID}$ is a partial function relating actions to resources,
- $\text{AcCase} : \mathcal{A} \rightarrow \mathcal{C}$ is a function relating actions to cases,
- $\succ \subseteq \mathcal{A} \times \mathcal{A}$ is a total ordering on mitigation actions indicating the order in which they need to be performed. We refer to this total ordering as the mitigation sequence.

The insertion of a new mitigation action $a \notin \mathcal{A}$ into mitigation M , can be expressed as $\text{addMit}(M, a, et, t, r, c) \triangleq (\mathcal{A} \cup \{a\}, \text{AcType} \cup \{(a, et)\}, \text{AcTask} \cup \{(a, t)\}, \text{AcRes} \cup \{(a, r)\}, \text{AcCase} \cup \{(a, c)\}, \succ \cup \{(x, a) \mid x \in \mathcal{A}\})$.

Now we are in a position to introduce the mitigation actions. For each action we will provide a short description of its behavior; we will quantify its cost and specify the precondition(s) required for its application. All these actions are executed in the context of a mitigation M . As soon as a risk is detected we collect the log L containing all process cases. This log is used to generate an execution graph G' , that we refer to as the original execution graph. It is used as a reference for comparison with the original status of the system. The effects of mitigations actions are explored, though not yet applied, during execution of the mitigation algorithm, and hence they are performed on a clone of the original execution graph which we will refer to as G .

Throughout the remainder of this section G' is the original execution graph, G the mitigation graph in use, and $c \in \mathcal{C}$ is a case. Moreover, whenever a node is modified, we need to store the time this modification occurred. In order to capture the time, we use function $\text{curr}()$.

A mitigation is a sequence of mitigation actions. Below we describe the mitigation actions supported by the PRSA algorithm, and the effects that each action yields. Due to space issues, only some actions are fully furnished in the form of an algorithm in this paper. We refer the reader to the technical report for the algorithms of all mitigation actions [13].

Deoffer. This action deoffers a task from a resource to whom the task was offered. We can execute $\text{deOff}(c, G, M)$ as described in Algorithm 1 if there is a work item

$x \in \circlearrowleft_G$ such that x is an *offered* work item. The cost of this action was set to 1 and this action serves as a reference for the cost of the other actions. With reference to our working example, let us assume that for certain process instances an order cannot be updated (e.g. when an order's line items have already gone into production their quantity can no longer be reduced). To prevent such update, we can set a risk condition that is satisfied as soon as a work item of task "Update Shipment Payment Order" is offered to a resource in a specific instance. This risk can be annulled by deoffering this work item and then skipping the work item altogether to prevent it from being reoffered.

Deallocate. This action deallocates a task from the resource to whom the task was allocated. If there is a work item $x \in \circlearrowleft_G$ such that x is an *allocated* work item, we can execute $deAll(c, G, M)$. We set the cost of this action to 2, since considering the progress status of a work item, deallocating a work item should be more "expensive" than deoffering it. In the Payment subprocess this action could be used to mitigate the approval fraud risk. The work item of "Approve Shipment Payment Order" can be deallocated from the resource to whom this work item is allocated when the risk is detected, since this resource approved another order for the same customer in the past.

Destart. This action brings an already started work item back to the state *allocated* and allocates it to the resource who started it. We can execute $deSta(c, G, M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is a *started* work item. For this action we set the cost to 3 as destarting a work item requires more effort than deallocating a work item. The *destart* action may also be used to mitigate an approval fraud risk. For example, it may be used to "free up" a resource who has never approved an order for the current customer, reducing this way the probability of allocating the work item of "Approve Shipment Payment Order" to a resource who has approved another order for the same customer in the past.

Offer. This action offers a work item to a resource to whom the task is not currently offered, either because it is not yet part of the set of resources to whom the task is currently offered, or because the task is currently *deoffered*. Given a function D that relates tasks to the set of resources to whom their work items can be offered, we can execute $off(D, c, G, G', M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is an *offered* or *deoffered* work item, and this work item is an *offered*, *allocated* or *started* work item in the execution graph G' . This action has a cost of 1, the same as *deoffer*. Since this action can only be executed if we previously executed a *deoffer*, these two actions can be combined to "reoffer" a work item to another resource (with a total cost of 2). For example, to reduce the risk of approval fraud in the Payment subprocess, we can deoffer the work item of "Approve Shipment Payment Order" from all Senior Financial Officers that have already approved another order for the same customer in the past, and offer that work item to a Senior Financial Officer that does not satisfy this condition.

Allocate. This action reallocates a work item that was deallocated before (and still has not been allocated) to a resource to whom the task was not allocated when the deallocation took place. We can execute $all(c, G, G', M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is an offered work item, and x is originally an *allocated* or *started* work item. This action has a cost of -1 . This action can only be executed if we previously executed a *deallocate*, with the result of changing the resource involved in a work item (so the total cost is $2 - 1 = 1$). We can use the combination *deallocate* + *allocate* as an

Algorithm 1: Deoffer Task

```

function deOff(Case c, Mitigation Graph G, Mitigation M);
Output: Execution Graph G, Mitigation M
begin
  n  $\leftarrow$  Any( $\{x \in \circ_G \mid \text{Status}(x) = \text{offered}\}$ );
  if n  $\neq \perp$  then
    r  $\leftarrow$  Any(NodeRes(n, offered));
    if |NodeRes(n, offered)| > 1 then
      et  $\leftarrow$  offered;
      TimeNode  $\leftarrow$  TimeNode  $\oplus$   $\{((n, \text{offered}), \text{curr}())\}$ ;
      NodeRes  $\leftarrow$  NodeRes  $\oplus$   $\{((n, \text{offered}), \text{NodeRes}(n, \text{offered}) \setminus \{r\})\}$ ;
    else
      et  $\leftarrow$  deoffered;
      TimeNode  $\leftarrow$   $\{(n, \text{offered})\} \triangleleft$  TimeNode;
      NodeRes  $\leftarrow$  NodeRes(n, offered)  $\triangleleft$  NodeRes;
      VarNode  $\leftarrow$   $\{(n, v) \mid \text{VarNode}(n, v) \in \Omega\} \triangleleft$  TimeNode;
      Status  $\leftarrow$  Status  $\oplus$   $\{(n, et)\}$ ;
      M  $\leftarrow$  addMit(M, NewAction(), deoffer, NodeTask(n), r, c);
  return (G, M)
end

```

alternative to mitigate the risk of approval fraud. With a total cost of 1, this combination would be preferred to using *deoffer* + *offer*.

Start. This action restarts a work item that was previously restarted (and has not yet been restarted) and associates it with a different resource from the one who started the task. We can execute *sta*(*c*, *G*, *G'*, *M*) if there is a work item $x \in \circ_G$ such that *x* is an *allocated* work item, and *x* is originally a *started* work item. The cost of this action is -1 , and the reasoning is similar to that used for the *allocate* action. This mitigation action can be used to reduce the negative impact of a *deoffer* or *deallocate* previously performed on a process instance.

Rollback. This action returns a completed work item to the status of unoffered. We can execute *rollbackTask*(*c*, *G*) if there is a work item $x \in \circ_G$ such that *x* is a *completed* work item. Its operationalization is described in Algorithm 2. The rollback action restores the case to a consistent status where the execution of a given work item never happened. A compensation routine can be associated with a task, so that it is triggered when the task is rolled back. The idea of this compensation routine is to deal with elements outside the control of the workflow engine (e.g. returning the money to a client after their payment has been rolled back). The rollback action is our most powerful action and has a cost of 9, obtained by adding the absolute values of all the actions introduced until now. In our Payment subprocess, we can use this action when we execute a large number of updates on the same Payment Order.

Skip. This action marks an unoffered and skippable task as ‘to be skipped’. If there exists a task $t \in \text{skippable}$ which does not have any work item active or completed, and there not exists a mitigation action $a \in \mathcal{A}$ which skipped task *t* for case *c*, then we define *skipTask*(*c*, *G*). To limit the use of this action, since this action may produce

Algorithm 2: Rollback Task

```

function rollbackTask(Case c, Execution Graph G, Mitigation M);
Output: Execution Graph G, Mitigation M
begin
   $n_1 \leftarrow \text{Any}(\{x \in \odot_G \mid \text{Status}_G(x) \in \text{Completed}\});$ 
  if  $n_1 \neq \perp$  then
    foreach  $n_2 \in \text{Node}_G$  do
      if  $n_1 \rightsquigarrow_G^* n_2$  then  $\text{Status}_G \leftarrow \text{Status}_G \oplus \{(n_2, \text{rollback})\};$ 
       $\text{Status}_G \leftarrow \text{Status}_G \oplus \{(n_1, \text{rollback})\};$ 
       $M \leftarrow \text{addMit}(M, \text{NewAction}(), \text{rollback}, \text{NodeTask}_G(n_1), \perp, c);$ 
  return (G, M)
end

```

inconsistency in the data, we decided to assign a cost of 9. The utility of this action can be seen in two situations when we consider our running example. The first situation is the order unfulfillment. In this case, to prevent the reiterated execution of an update, we may decide to skip the “Update Shipment Payment Order” task. The second situation is the overtime process risk. In this case we may decide to skip some tasks in order to complete the process in time.

Relocate Resource. This action looks for a resource that is only involved in the execution of a work item belonging to a case for which no risk was defined. If once such a resource is found, it deallocates (and destarts if necessary) the work item associated with this resource and allocates the resource to a work item of the process case that we want to mitigate. The cost of this action is 7 since this action performs a (partial) sequence of destart and deallocate on two work items, and another allocate and a start action on one work item. Let x be an active border work item $x \in \odot_G$ in case c , r be a resource involved only in case c_2 , and c_2 be process which is not risky. If resource r only started or allocated one work item (of any active border events), then we can execute $relRes(c, G, M, SRC)$.

4 Evaluation

We implemented the PRSA algorithm as a custom service in the YAWL system.² We extended the YAWL system as it is built on a service-oriented architecture, which facilitates the addition of new services; it is open-source, which facilitates its distribution among academics and practitioners; and as the underlying YAWL language provides comprehensive supports for the workflow patterns [19].

The risk mitigation service interacts with the *risk detection service*³ that we developed previously [12], for the sake of identifying risks and computing their probabilities. It uses as input a reference to the process instance whose risks need to be mitigated, the complete YAWL specification for this instance, a log of the process (as extracted from the YAWL system), and a copy of the risk sensors associated with the process instance, as provided by the risk detection service. Modifications that a mitigation may introduce

² <http://www.yawlfoundation.org>

³ <http://www.yawlfoundation.org/prsa>

are communicated to the risk detection service, which recomputes the risk probabilities. The final solutions are returned to the user as recommendations. The one chosen by the user is then applied to the process instance under exam using the APIs provided by the YAWL engine. We implemented compensation actions associated with rolled back work items via the YAWL Worklet mechanism [19]. Accordingly, we equipped the YAWL Editor with an interface to allow users to associate a Worklet containing a compensation action to a task. When an instance of this task is rolled back, the associated Worklet is run as a separate process instance in the YAWL engine, so that from an engine perspective, the Worklet and its invoking processes are two distinct cases.

To prove the feasibility of our approach, we ran three experiments. First, we tested the required time to mitigate the same set of risks on different process models. Second, we checked the dependency of the mitigation time on different variables. Third, we checked the quality of the mitigations proposed on a specific process model.

For the first experiment, we used four real-life process models available to the research team, for which we could identify risk conditions. The sizes of these models range from 5 to 20 tasks. The first model (Process A) describes a film production process, carried out on a daily basis. This process is taken from a case study we conducted in collaboration with the Australian Film, Television and Radio School [25]. The other three models are subprocesses of an order fulfillment process inspired by the VICS industry standard for logistics [34]. The first one (Process B) deals with the ordering, the second (Process C) deals with the payment for the goods and is the process we showed in Section 2, the third model (Process D) deals with the delivery of the goods.

Next, we defined seven generic risk conditions that are applicable to all these processes, where with “generic” we mean risks that are not linked to a specific context, such as a financial frauds, but that are linked to control-flow aspects. These conditions represent possible undesirable situations that may arise in a process, and relate to different process aspects such as data, resources and control-flow elements. They are domain-independent so that we could define them on all four process models. The first condition detects a situation where two concurrent work items may not complete in a desired order. The second one is used to detect a violation of the four-eyes principle between parallel work items. The third one detects whether a time limit is exceeded when executing a loop. The fourth condition detects a possible delay with the execution of a work item. The fifth one detects the possibility that two concurrent work items that should be executed by the same resource are actually allocated to two different resources (a situation that is not possible to enforce with many workflow management systems). The sixth one detects a delay with the execution of a portion of the process while the seventh one detects a data error, specifically if the data values produced by two concurrent work items are not the same.

For each process model we generated a variant with a specific combination of the above risk conditions. This led to a total of 180 process models (not every risk identified could be applied to every process model, since some of these risk conditions require parallelism and/or loops that are not present in every process model). These process models are as follows: 19 models with 1 risk condition, 40 models with 2 risk conditions, 50 with 3 risk conditions, 41 models with 4 risk conditions,

22 process models with five risk conditions, seven process models with six risk conditions, and one process model with all seven risk conditions.

For each process model we ran ten tests and averaged the results. Each test was executed on the first state of a process instance where all the risk conditions evaluated to true. For each group of tests on the same process model we measured the time required to obtain the first solution that mitigates all risks, and the number of candidate solutions generated by the algorithm in order to obtain this solution. We performed the tests on an Intel Core I5 M560 2.67GHz processor with 4GB RAM, running Linux Ubuntu v11.10.

Table 1 shows the results of this experiment. The second, third and fourth columns show the size (as number of tasks), the number of variants and the number of risk conditions for each of the four process models. The fifth and sixth columns show the mitigation time required to find the first solution, and the number of candidate solutions explored to find such a solution. From this table we can observe that the algorithm takes at most 3 mins (179 secs) to mitigate multiple risks in a variant of Process A (this timing refers to a combination of 5 risks for this process), though the average time is much lower (19 secs across all models). It seems reasonable to assume that in most business scenarios mitigation times in the order of a few minutes are acceptable, compared to the average time required to perform a task, and thus the average duration of a process instance. For example, let us assume an average duration of 24 hours for the Payment subprocess, with a new task being executed every 30 mins. Let us also assume that we sample the risk conditions every 5 mins. This means we have up to 6 mins to mitigate all identified risks before a new task is executed which may change the risk conditions.

Table 1 also shows that the algorithm needs to explore a very large number of candidate solutions to find the first solution (2,456 solutions on average across all models). While it is not fair to compare the computation power of a machine to that of humans, this result highlights the complexity of finding a solution. It is reasonable to think that many of these candidate solutions explored by the algorithm would also need be evaluated by a human in order to find the right solution.

Table 1. Time and number of candidate solutions explored to find the first solution

Process	Size	Variants	Risks avg/max	Mitigation time [sec]			Candidates		
				min	max	avg	min	max	avg
Process A	20	127	3.53 / 7	0.003	178.891	26.415	2	20,181	3,456
Process B	5	7	1.71 / 3	0.001	0.033	0.015	3	54	32
Process C	15	31	3.05 / 5	0.001	0.117	0.030	2	256	60.93
Process D	5	15	2.13 / 4	0.004	0.929	0.170	2	553	78.2
Total	45	180	3.18 / 7	0.001	178.891	18.657	2	20,181	2,457

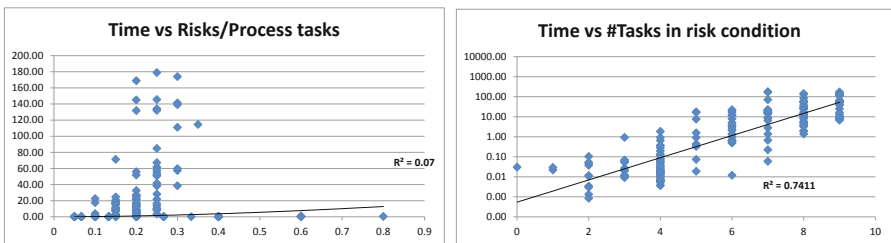


Fig. 3. Correlation between time and a) risks/tasks ratio, b) tasks in risk conditions

In the second experiment, we investigated the factors affecting the performance of the algorithm. One would think that the mitigation time is proportional to the number of risks defined in a process model, and to the model size itself. The larger the number of risks and/or the model size, the longer it should take to mitigate such risks. However the data we extrapolated from Table 1 does not confirm this hypothesis. For example, the 21 variants of Process A with 5 risks have mitigation times ranging from 3.3 to 179 secs, despite their sizes and number of risks being the same. To verify that the mitigation time is not sensitive to the number of risks, nor to the process size, we plotted the correlation between the mitigation time and the ratio risks/process size in Figure 3a (the solid line is the linear regression of the points). The low value of the coefficient of determination R^2 (0.07) confirms this intuition. We then checked the correlation between the mitigation time and the number of tasks used in risk conditions. The intuition is that the more work items of these tasks are pending in a given state of the process instance, the larger the number of possible mitigation actions. The corresponding scatter plot is shown in Figure 3b, which indeed confirms this intuition ($R^2 = 0.74$).

Finally, we checked the feasibility of the solutions proposed by the algorithm, when mitigating the domain-specific risks associated with the Payment subprocess (cf. Section 2). We recall that two of these risks (overtime process and order unfulfillment) are detected when the associated probability, obtained by analyzing historical data, exceeds a tolerance threshold, whereas the third risk (approval fraud) involves a complex risk condition. We considered the first state of an instance of the Payment subprocess when all three risks are active. This occurs after executing “Update shipment payment order” for the third time, once task “Approve shipment payment order” has been allocated to a resource who has already executed this task in the past.

To obtain a small number of solutions, we stopped the algorithm after one min of execution. In this time-frame, five solutions were retrieved. For each solution, Table 2 reports whether the solution mitigates each of the three risks, and the cost of the solution in terms of mitigation actions performed on the initial process instance. In particular, a “-” indicates a risk not mitigated, a “+” indicates a risk mitigated (with risk probability lower than the specific threshold if the condition depends on the risk probability), and a “±” indicates a risk mitigated whose condition cannot be computed for lack of information, i.e. some of the variables used in the risk condition are null. We recall that the algorithm prioritizes a solution whose risk is mitigated by computing the risk condition, than a solution whose risk is mitigated because the respective condition cannot be computed.

Table 2. Payment subprocess mitigation

Solutions [at 1 min]	1	2	3	4	5
Overtime Process	+	+	+	+	+
Approval Fraud	+	+	+	+	+
Order Unfulfillment	+	+	±	±	-
Cost	50	50	40	40	19

The five solutions identified are pairwise mutually non-dominating. Solutions 1 and 2 are dominated by solutions 3, 4 and 5 cost-wise, but dominate these solutions w.r.t. the mitigation of the order unfulfillment risk. Solution 5 dominates solutions 3 and 4 cost-wise but is dominated by these two solutions w.r.t. the mitigation of the order unfulfillment risk.

Let us briefly examine the mitigations performed by the five solutions. The first four solutions mitigate the approval fraud by deallocating the resource that was allocated

“Approve shipment payment order”, while solution 5 additionally allocates the work item to a resource who did not execute this task for the same customer in the past. All these mitigations are feasible, though the one provided by solution 5 is more robust, since there is no risk that the task gets allocated to a resource who has already executed it. The order unfulfillment risk is mitigated by solutions 1 and 2 through rolling back the work item of task “Update shipment payment order” (which leads to a deoffer of the work item of task “Approve shipment payment order” that comes afterwards). Solutions 3 and 4 do this too but also mark this task ‘to be skipped’ preventing a possible re-execution of it. This action sets to null the risk variables associated with this task that retrieve the number of executions and its estimated remaining time making the risk mitigated but not computable. Thus, while all four solutions are feasible, we would prioritise the first two since these ensure that the risk probability has actually dropped below the threshold. Finally, all solutions differ in the way they mitigate the overtime process risk. Each of them skips a different task among those not yet executed (for simplicity, all of them have the same estimated duration). Despite the fact that all these solutions are feasible, only the mitigation proposed by solution 3 is interesting since it proposes to skip tasks “Update Shipment Payment Order” and “Approve Shipment Payment Order” avoiding this way that the loop is taken again. In other words, it prevents the order to undergo further updates, and subsequent approvals.

5 Related Work

Risk mitigation is an essential step in the risk management process [29]. Several risk analysis methods such as OCTAVE [4], CRAMM [6] and CORAS [22] describe guidelines for identifying risk mitigations. For example, these guidelines provide instructions on how to conduct structured brainstorming sessions with experts in order to identify viable mitigation procedures. Although helpful, these guidelines are too generic and no support is offered on how mitigation procedures could be operationalized. Similarly, the academic literature recognizes the importance of mitigating process-related risks, though it focuses on risk-aware BPM methodologies in general, rather than on concrete algorithms for automating risk mitigation [24,20,14,30,27,8,33]. For example, the ROPE (Risk-Oriented Process Evaluation) methodology [33] is concerned with threats to the resources required for process execution. If a required resource becomes unavailable, pre-planned countermeasures and recovery procedures are *manually* enacted to handle the fault. These procedures are defined and validated via a simulator at design-time; enactment at runtime is designated to a ‘responsible person’, that is, the mitigation and recovery operations are not automated. For a comprehensive survey of these risk-aware BPM methodologies, we refer to [31].

The mitigation actions proposed in this paper share commonalities with the workflow exception patterns [26]. For example, the *reoffer* pattern at the work item level can be obtained by combining our mitigation actions *deoffer* + *offer*, which represent atomic operations in this respect. Another similarity is between our *rollback* action and the recovery patterns *rollback* and *compensate*, since our rollback can also trigger a compensation action if this is available for the task being rolled back. Given that we need to operationalize these actions, we do provide a precise characterization of all actions and their effects, whereas the work in [26] limits itself to a textual description

of these exception patterns, as they are observed from an analysis of process modeling languages and tools. That said, the main contribution of our paper is not the proposed set of mitigation actions per se (which could be replaced or modified) but rather an automated mechanism for combining these actions in an optimal way in order to mitigate a set of process-related risks as far as possible.

Risks like those we illustrated in this paper can also be encoded as *constraints* on top of a process model. Approaches exist that can check whether there exists at least an instance of a constrained process model that can be executed without violating the constraints. For example, Combi and Posenato [10] propose a general method for checking the satisfiability of temporal process constraints (like our overtime process risk) at design-time. Other approaches can also enforce these constraints at run-time, so that any process instance will satisfy all the constraints by construction. For example, Tan et al. [32] propose a model for constrained workflow execution that addresses cardinality constraints (e.g. to control how many times a task can be executed), binding of duty constraints (i.e. the ability of a resource to retain a familiar work item) and separation of duty constraints (i.e. different resources should execute different tasks). These constraints can be enforced only within a given process instance. The approach proposed by Warner and Atluri [35] overcomes this limitation by proposing a constraint specification language for resource allocation that also addresses inter-instance constraints. Compared to our work, constrained workflow execution provides a rigid approach according to which if the constraints cannot be satisfied, a process model will simply not be instantiated, or if instantiated, the instance violating the constraints will throw an exception. Commercial systems such as IBM WebSphere and AristaFlow also support constrained workflow execution, and handle these violations (e.g. violations of temporal constraints) by escalating control to a process administrator.⁴ Instead, our approach allows a process instance to be automatically *adapted* on-the-fly in order to reduce the risk of violating such constraints.

Various research frameworks have been proposed for the *dynamic adaptation* of process instances. For example, ADEPT [15] supports adding, deleting and changing the sequence of tasks at both the model and instance levels, however such changes must be achieved via manual intervention by an administrator. AgentWork [23] provides the ability to modify process instances by dropping and adding individual tasks based on events and rules. CBRFlow [36] uses case-based reasoning to support runtime adaptation by allowing users to annotate rules during process execution. CEVICHE [18] is a service-based framework that uses the AO4BPEL (Aspect-Oriented for BPEL) language [9] to provide an option for skipping or reallocating tasks to other services in an ad-hoc manner. While these approaches could be used for risk mitigation purposes, they do not provide any help for the identification of which particular mitigation actions should be used. The YAWL Worklet Service [3] provides each task of a process instance with the ability to be associated with an extensible repertoire of actions ('drop-in' processes), one of which is contextually and dynamically bound to the task at runtime. It also supports capabilities for dynamically detecting and handling runtime exceptions.

⁴ <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.612.help.tel.ui.doc/topics/tescal8.html>

However the approach is generic and not specifically designed for risk detection and mitigation. Also a new situation cannot automatically be dealt with but requires a workflow administrator to intervene.

Our work is also related to *operational support* in process mining [1]. Operational support deals with the analysis of current and historical execution data, with the aim to predict future states of a running process instance, and provide recommendations to guide the user in selecting the next activity to execute based on certain objectives. For example, the approach for cycle time prediction in [2] could be, with the opportune modifications, adapted for risk prediction. Using this approach it would be possible to estimate the probability of an overtime risk and suggest the next steps the current instance should take in order to keep this risk under control. The application of this approach unfortunately requires that the process model captures all the possible mitigation actions as normal activities, i.e. as control-flow alternatives. For instance, if a task can be skipped, there should be a path without that task that leads to the end node of the process model. This may drastically increase the complexity of the process model. Moreover, this approach would not be applicable to capture mitigation operations on resources (i.e. deallocating a resource) or on task states (e.g. suspending a task). That said, more in general, our approach could be seen as a possible provider for operation support, and could thus be integrated in process mining environments like ProM.⁵

Our work provides recommendations to users as to which mitigation actions can be applied to the specific context at hand. As such, it shares commonalities with recommendation and decision support systems. Alter [5] states that the focus of such systems should be towards improving decision making within work systems, rather than externalizing support. This view is shared by our technique, which provides an extension to existing process-aware information systems, rather than a separate standalone tool. As such, it may be considered a member of the domain known as *Group Decision Support Systems*, which facilitate task support in group environments.

In previous work [17] we explored the use of dominance-based MOSA for automatically fixing behavioral errors in process models, at design-time. Our work on risk mitigation can thus be seen as an adaptation of that idea to run-time aspects, since we aim to improve running process instances. Besides their distinct aims, the main difference between the two approaches is that for correcting behavioral errors we defined three objective functions capturing the structural and behavioral similarity of a solution to the incorrect model, whereas in risk mitigation the number and type of objective functions depends on which risks are active in a given state of a process instance.

6 Conclusion

This paper contributes a concrete technique for the automatic mitigation of process-related risks at run-time. The technique requires as input an executable process model and a set of associated risk conditions. At run-time, when one or more risk conditions evaluate to true, a process administrator can launch our technique to mitigate the identified risks and bring the process instance back to a safe state. This is achieved by generating a set of possible mitigations that change the current instance in order to bring

⁵ <http://processmining.org>

the likelihood of the identified risks below a tolerance level. These mitigation actions are not performed directly on the instance under consideration. Rather, their effects are simulated and those solutions that mitigate the most risks in a given timeframe, are proposed as recommendations to the process administrator.

The mitigation actions are determined via a dominance-based MOSA algorithm. This choice allows us to explore the solution space as widely as possible, avoiding local optima. In essence, each risk is treated as an objective function whose likelihood needs be minimized. The objective is reached as soon as the likelihood goes below the tolerance value for that particular risk. Mitigation actions affect various aspects of a process, such as task execution and resources utilization. To the best of our knowledge, this is the first time that process-related risks can be mitigated automatically.

The technique was implemented in the YAWL system and its performance evaluated with real-life process models. The tests show that on the analyzed process models a set of possible solutions can be found in a matter of seconds, or within a few minutes in the worst case, and that in all cases the associated risks are mitigated. We expect this technique to reduce the effort and time required by process administrators to understand what mitigation actions are feasible based on a particular state of the system. That said, we still need to validate the feasibility and appropriateness of the proposed mitigation actions with domain experts. We plan to do so by comparing the solutions obtained with our algorithm with those proposed by them. We also plan to improve the exploration of the solution space by prioritizing the mitigation of those risks that have the highest impact on the process objectives. In fact, currently all risks are treated alike whereas in reality this might not be the case. Finally, the algorithm could also be extended to prioritize certain mitigation actions based on how these have been ranked by the users in previously mitigated instances.

Acknowledgments We thank Peter Hughes and Wil van der Aalst for their valuable comments and suggestions. This research is funded by the ARC Discovery Project “Risk-aware Business Process Management” (DP110100091) and by the NICTA Queensland Lab.

References

1. van der Aalst, W.M.P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer (2011)
2. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. *Information Systems* 36(2), 450–475 (2011)
3. Adams, M., ter Hofstede, A.H.M., van der Aalst, W.M.P., Edmond, D.: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: Meersman, R., Tari, Z. (eds.) *OTM 2007, Part I*. LNCS, vol. 4803, pp. 95–112. Springer, Heidelberg (2007)
4. Alberts, C.J., Dorofee, A.J.: OCTAVE criteria, version 2.0. Technical Report CMU/SEI-2001-TR-016, Carnegie Mellon University (2001)
5. Alter, S.: A work system view of DSS in its fourth decade. In: *DSS*, vol. 38 (December 2004)
6. Barber, B., Davey, J.: The use of the CCTA Risk Analysis and Management Methodology CRAMM in health information systems. In: *MEDINFO*. North Holland Publishing (1992)
7. Basel Committee on Bankin Supervision. *Basel II: International Convergence of Capital Measurement and Capital Standards* (2006)

8. Betz, S., Hickl, S., Oberweis, A.: Risk-aware business process modeling and simulation using xml nets. In: IEEE CEC, pp. 349–356 (September 2011)
9. Charfi, A., Mezini, M.: AO4BPEL: An aspect-oriented extension to BPEL. In: WWW (2007)
10. Combi, C., Posenato, R.: Controllability in Temporal Conceptual Workflow Schemata. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 64–79. Springer, Heidelberg (2009)
11. International Electrotechnical Commission. IEC 61025 Fault Tree Analysis, FTA (1990)
12. Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M.: History-Aware, Real-Time Risk Detection in Business Processes. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part I. LNCS, vol. 7044, pp. 100–118. Springer, Heidelberg (2011)
13. Conforti, R., ter Hofstede, A.H.M., La Rosa, M., Adams, M.J.: Automated risk mitigation in business processes (extended version). QUT ePrints 49331 (2012)
14. Cope, E.W., Kuster, J.M., Etzweiler, D., Deleris, L.A., Ray, B.: Incorporating risk into business process models. IBM Journal of Research and Development 54(3), 4:1–4:13 (2010)
15. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. CSRD 23, 81–97 (2009)
16. 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)
17. 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)
18. Hermosillo, G., Seinturier, L., Duchien, L.: Using complex event processing for dynamic business process adaptation. In: SCC, pp. 466–473. IEEE (2010)
19. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N. (eds.): Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
20. Jallow, A.K., Majeed, B., Vergidis, K., Tiwari, A., Roy, R.: Operational risk analysis in business processes. BTTJ 25(1), 168–177 (2007)
21. Johnson, W.G.: MORT: The Management Oversight and Risk Tree. U.S. Atomic Energy Commission (1973)
22. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach. Springer (2011)
23. Muller, R., Greiner, U., Rahm, E.: AgentWork: a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering 51(2), 223–256 (2004)
24. Neiger, D., Churilov, L., zur Muehlen, M., Rosemann, M.: Integrating risks in business process models with value focused process engineering. In: ECIS. AISel (2006)
25. Ouyang, C., La Rosa, M., ter Hofstede, A.H.M., Dumas, M., Shortland, K.: Toward web-scale workflows for film production. IEEE, Internet Computing 12(5), 53–61 (2008)
26. 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)
27. Sienou, A., Lamine, E., Pingaud, H., Karduck, A.P.: Risk driven process engineering in digital ecosystems: Modelling risk. In: Proc. of IEEE DEST, pp. 647–650 (2010)
28. Smith, K.I., Everson, R.M., Fieldsend, J.E., Murphy, C., Misra, R.: Dominance-based multi-objective simulated annealing. IEEE TEC 12(3), 323–342 (2008)
29. Standards Australia and Standards New Zealand. Standard AS/NZS ISO 31000 (2009)
30. Strecker, S., Heise, D., Frank, U.: RiskM: A multi-perspective modeling method for IT risk assessment. Information Systems Frontiers, 1–17 (2010)

31. Suriadi, S., Weiß, B., Winkelmann, A., ter Hofstede, A., Wynn, M., Ouyang, C., Adams, M.J., Conforti, R., Fidge, C., La Rosa, M., Pika, A.: Current research in risk-aware business process management - overview, comparison, and gap analysis. QUT ePrints 50606 (2012)
32. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow. In: Proc. of IEEE CSFW, pp. 155–169 (June 2004)
33. Tjoa, S., Jakoubi, S., Goluch, G., Kitzler, G., Goluch, S., Quirchmayr, G.: A formal approach enabling risk-aware business process modeling and simulation. IEEE TSC 4(2) (2011)
34. Voluntary Interindustry Commerce Solutions Association. Voluntary Inter-industry Commerce Standard (VICS), <http://www.vics.org> (accessed: June 2011)
35. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proc. of SACMAT, pp. 190–199. ACM, New York (2006)
36. Weber, B., Wild, W., Feige, U.: CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 434–448. Springer, Heidelberg (2004)

Aligning Service-Oriented Architectures with Security Requirements

Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini

University of Trento, Italy

{mattia.salnitri,f.dalpiaz,paolo.giorgini}@unitn.it

Abstract. Aligning requirements and architectures is a long-standing concern in software engineering. Alignment is crucial in the area of systems evolution, wherein requirements and system architectures keep changing after system deployment. We address a specific alignment problem, namely, checking the compliance of a service-oriented architecture—representing a composite service—with security requirements. Service-oriented architectures are dynamic (services can be replaced on-the-fly), and assessing compliance with security requirements is key, since non-compliance may lead to sanctions as well as privacy violation. After motivating and describing the problem, we propose algorithms to check two specific security requirements: non-disclosure and non-repudiation. We illustrate the approach using an e-government scenario.

Keywords: SOA, alignment, evolution, security requirements.

1 Introduction

The alignment between requirements and software architectures (R/A alignment, from now on) is an age-old yet actual problem in software engineering in general, and in requirements engineering in particular [17, 22]. This problem is traditionally addressed at design-time in a top-down fashion [2, 16], by providing methodological guidelines to requirements engineers and system architects for deriving an architecture that satisfies a given requirements specification.

Though useful at design-time, these approaches are only a partial solution when considering that both requirements and architectures are subject to evolution after system deployment. Requirements evolve [11] due to changes in the stakeholders needs, in organizational policies, in norms and laws in the deployment environment, and as a result of feedback about system operation. Architectural evolution [1] can be either internal—the architecture topology changes—or external—the specification of components and interactions is altered.

Architectural evolution is driven by requirements evolution, i.e., architectures need to evolve when they do not satisfy their requirements. Semi-automated runtime compliance verification requires understanding and formalizing the conceptual relationship between requirements and architectural models. The challenge is to relate these two artifacts, that exploit different modeling abstractions, which refer to the problem space and the solution space, respectively.

In this paper, we are interested in the alignment between security requirements and service-oriented architectures:

- *security requirements* are crucial, as non compliance may violate norms, e.g., about privacy, or usage / modification of official documents. In turn, this may imply monetary compensations, dissatisfaction of customers, and decreasing reputation;
- *service-oriented architectures* represent an inherently evolving type of architecture, where composite services [3] may evolve—either due to designer intervention or through self-recomposition—when services are replaced or when the composition structure changes.

Our contribution is a semi-automated approach to support runtime alignment between composite services and security requirements. First, the analysts are expected to conceptually connect the architecture specification with the security requirements, i.e., linking concepts in the requirements (goals, actors, security needs) with concepts in the composite service (activity, participant, data flow). Second, automated algorithms check compliance at run-time, whenever changes in the architecture or in the requirements occur. Compared to existing approaches, ours minimizes human involvement after system deployment.

Organization. Sec. 2 presents the baseline languages for security requirements and composite services. Sec. 3 details the R/A alignment problem. Sec. 4 describes the conceptual link between the two models. Sec. 5 illustrates two algorithms for checking security requirements: non-disclosure and non-repudiation. Sec. 6 discusses related work and presents future directions.

2 Baseline

In Sec. 2.1 we introduce the adopted modeling language for security requirements, while in Sec. 2.2 the language to describe the architecture of service compositions.

Example 1 (eGovernment). Land selling involves not only finding a buyer, but also exchanging documents with governmental bodies. The municipality has to certify that the land is residential zoning. We suppose land selling is supported by an eGov application that sends the official contract (including the municipality's certification) to the ministry (who can object), and archives the contract. □

2.1 Security Requirements with STS-ml

Many security requirements modeling languages have been proposed so far. Some extend UML—mainly abuse cases [14] and misuse cases [19]—by adding undesirable usage scenarios of the system. In a similar spirit, Lamsweerde [24] extends goal modeling by adding an anti-model that describes the goals of attackers. Differently, SI* [9] and Secure Tropos [15] aim at modeling security needs. We choose STS-ml [5], a goal-oriented language for service-oriented settings, in which security requirements constrain the interactions between actors.

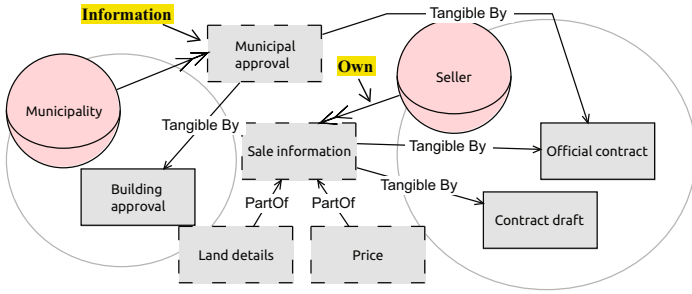


Fig. 2. STS-ml information view for the eGovernment example

actors with a middle box. Such box details the granted permissions (Use, Modify, Produce, Distribute) on documents representing specific information. Authorizations can be limited by defining a scope: one or more goals for whose fulfillment the permissions are granted. In Fig. 3, the *Seller* authorizes the *Municipality* to use *Sale information* in the scope of goal *Approval provided*.

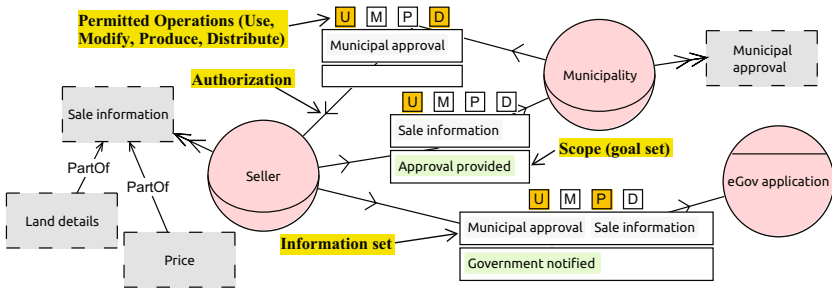


Fig. 3. STS-ml authorization view for the eGovernment example

Starting from the three views that constitute an STS-ml model, a Security Requirements Specification (SRS) is derived using the STS-tool¹. The specification details the security requirements—social commitments between couples of actors—, as well as a knowledge base, that makes the SRS self-contained.

A social commitment is a quaternary relation $C(x,y,p,q)$, where a debtor actor x commits to a creditor actor y that, if the proposition p is brought about, then the proposition q will be brought about [20]. STS-ml supports a specialized version of commitments to express security requirements; in particular, the consequent q is about the satisfaction of a security need.

Table 1 is a partial SRS for the eGovernment scenario. C_1 is a security requirement about non-repudiation: the *eGov application* commits to the *Seller* that, if goal *Government notified* is delegated, such delegation will not be repudiated. Both C_2 and C_3 concern non-disclosure. In C_3 , the *Municipality* commits to the

¹ <http://www.sts-tool.eu>

Table 1. Part of the security requirements specification for the scenario in Figures 1-3

<p>Security requirements: $C_1: C(\text{eGov application, Seller, } D=\text{delegation}(\text{Seller, eGov application, Government notified), non-rep}(D)),$ $C_2: C(\text{e-Gov application, Seller, } \top, \text{non-disc}(\text{Municipal approval}\wedge\text{Sale information}),$ $C_3: C(\text{Municipality, Seller, } \top, \text{non-disc}(\text{Sale information}), \dots$</p> <p>Knowledge base: $\text{part-of}(\text{Land details, Sale information}), \text{part-of}(\text{Price, Sale information}), \dots$ $\text{tangible-by}(\text{Sale information, Official contract}), \text{tangible-by}(\text{Sale information, Contract draft}), \dots$ $\text{owns}(\text{Seller, Sale information}), \dots$</p>

Seller that *Sale information* will not be disclosed. The knowledge base describes relations that enable the understanding of the relationships related to information. For example, it describes the information parts that *Sale information* is composed of, it details the documents that enable exchanging *Sale information*, and information owners.

2.2 BPMN with Security Extensions

We describe the architecture of services composition—i.e., the way services are interconnected—by using an extended version of the Business Process Modeling Notation (BPMN) with support for security annotations (inspired by [18]). The notation we use in this paper can be easily adopted by existing tools, provided that they include support for representing the data flow between activities and enable the definition of custom security-related activities.

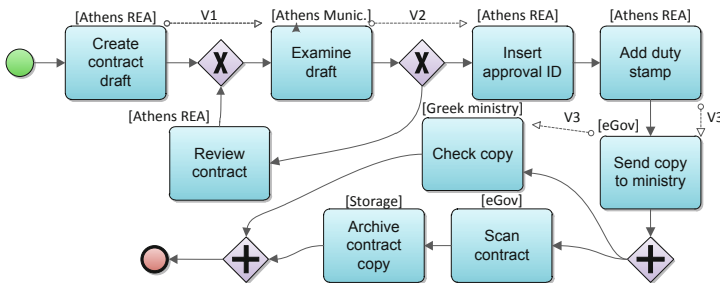


Fig. 4. Security-extended BPMN for the eGovernment example

Fig. 4 shows a security-extended BPMN model for the eGovernment scenario. Such model is a set of activities (e.g., *Create contract draft*) carried out by different performers (e.g., *Athens REA*) following the specified control flow (solid arrows, exclusive and parallel gates). For example, after the real estate company

Athens REA has completed activity *Create contract draft*, the municipality of Athens has to examine the draft. The model also represents the information flow through labeled dashed arrows, whose label indicates the variable that is transferred between two performers. So as to determine how an activity uses a variable, we consider the incoming and outgoing arrows related to that variable:

- *Create contract draft* produces $V1$, as it has an outgoing arrow labeled $V1$ but no incoming arrow;
- *Examine draft* uses $V1$, as there is only an incoming arrow labeled $V1$;
- *Send copy to ministry* may modify $V3$, as there are both incoming and outgoing arrows labeled $V3$.

3 The R/A Alignment Problem

We motivate and illustrate the evolution of requirements and architectures, and show its implications on R/A alignment. We refer to specific requirements specifications (business processes) with the labels SRS_1, \dots, SRS_n (BP_1, \dots, BP_n). After illustrating the problem in Sec. 3.1, we analyze the evolution of security requirements and service compositions in Sec. 3.2 and Sec. 3.3, respectively.

3.1 The R/A Alignment Problem Explained

Fig. 5 shows some possible evolutions of requirements and architectures, and emphasizes their impact on R/A alignment. At design-time, the service composition BP_1 meets the initial security requirements specification SRS_1 . When the composition self-reconfigures at runtime, and is replaced by another composition BP_2 , alignment with SRS_1 is lost. The requirements analyst relaxes some security requirements of secondary importance to the stakeholders. This leads to a new specification SRS_2 which guarantees R/A alignment. Later, stricter privacy laws are introduced. The analyst revises the requirements models, leading to a new specification SRS_3 . BP_2 is now non-aligned with SRS_3 . Then, the system architect defines a new composition BP_3 that is aligned with SRS_3 .

Non-compliance with security requirements has severe consequences [4]. Loss of privacy and unauthorized disclosure of data are serious threats, especially when loss of confidentiality can potentially open the doors to world-wide access to personal information through the Internet. Another consequence is data integrity: imagine what would happen if the price for a land sale were changed by an unauthorized user! Compliance with organizational processes and standards is also at risk: take the case, for instance, if binding-of-duties, separation-of-duties, and redundancy are not provided as expected. In general, security requirements violations have consequences from an economical perspective [6], lead to penalties imposed by laws, and decrease the reputation of involved organizations.

3.2 Security Requirements Evolution

The effect of the evolution of security requirements is that a new specification SRS_2 replaces the previously valid specification SRS_1 , and the two differ in at

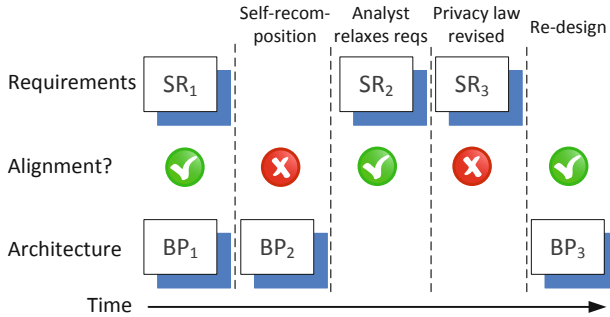


Fig. 5. R/A alignment threatened by the evolution of requirements and architecture

least one element. Since our considered security requirements specification is derived from an STS-ml model, changes in *SRS* are triggered by changes in the social, resource, and authorization views of the corresponding STS-ml model.

The effects of requirements evolution may turn a compliant architecture into a non-compliant one, or vice-versa. These effects arise due to the occurrence of specific events that cause the evolution of security requirements:

- *Revised security needs*: the stakeholders change their desires about security, or the organizational policies are revised. For example, in the eGovernment scenario, the municipality may establish that the seller cannot transfer its authorization on documents representing the municipal approval;
- *Broadened analysis scope*: the requirements analyst may realize that the conducted analysis does not cover all the important stakeholders. For example, the requirements analyst may realize that the STS-ml models do not include any actor representing the competent ministry. In turn, this will lead to further security requirements about confidentiality;
- *Evolution of norms*: the security norms in the legal environment where the service composition is deployed change. New laws imposing stricter security could be introduced, existing norms could be strengthened, weakened, and abrogated. For example, consider a new municipal law stating that the prices of land transactions shall not be notified to governmental bodies;
- *Legal context switching*: the actors participating in security requirements are not always located in the same legal context. If the actor is a role, in particular, different agents—in different locations—may adopt such role; depending on the location, a different set of laws applies. For instance, a Spanish buyer would have to comply both with Greek and Spanish laws.

3.3 Service Composition Evolution

The architecture of a secure composite service *BP*₂ evolves a previous architecture *BP*₁ if they differ in either: (i) the activities—including the security-related activities—they are composed of; (ii) the control flow between activities; (iii) the information flow between activities; or (iv) the performers assigned to activities.

A service composition may evolve either autonomously, or due to human intervention. *Manual redesign* occurs when a designer defines an alternative composition to better satisfy her needs. *Self-recomposition* occurs when, based on the monitored runtime data, a workflow engine replaces the composition.

Diverse reasons may trigger the evolution of a composition, among which:

- *No available or no trustworthy service*: there is no available service for a specific activity, or existing providers are not sufficiently trustworthy. For example, if no employee were available to add the duty stamp, a web service that uses an electronic duty stamp could be introduced;
- *Functional failure or under-performance*: the service composition does not deliver the expected outcome, or its performance is not adequate. For instance, the service composition about eGovernment could be too slow for Athens REA, who may decide to outsource the approval process chunk;
- *Security failure*: the service composition fails to correctly deliver the security features declared at the architectural level. For example, if the service responsible for sending a copy of the contract to the ministry publishes it—violating the architectural information flow—confidentiality is violated;

4 Supporting R/A Alignment: A Methodological Approach

We describe a methodological approach to support R/A alignment. It starts at design-time by establishing initial alignment, and is continuously carried out at runtime, so as to identify non-alignment and to re-establish alignment in case of evolutions. The approach is semi-automated, as it includes activities to be conducted by analysts as well as machine-executable algorithms.

Fig. 6 illustrates our approach. Initially, an analyst models and analyzes the security requirements—e.g., using STS-ml, as explained in Sec. 2.1—and derives the *Security Requirements Specification (SRS)*. As soon as a business process *BP* defining the architecture a service composition is available, the analyst has to devise a conceptual mapping between elements in *SRS* and elements in *BP*. This step (Sec. 4.1) produces a conceptual mapping *CM*, which enables the automated instantiation of the security requirements. Such activity (Sec. 4.2) interprets the *SRS*—which is expressed over concepts in STS-ml such as actors and goals—in terms of the concepts that characterize service composition at hand—such as performers and activities—and produces an *Instantiated SRS (ISRS)*. This document and the *BP* feed alignment checking, a set of automated algorithms (some described in Sec. 5) that compute the actual R/A alignment. Based on the alignment status, different paths are followed:

- *Non-alignment*: the analyst has to revise either the composition, the *SRS*, or both. While the basic case involves the analyst in a redesign activity, an interesting option is to trigger an automated service re-composition. The applied revisions require the conceptual mapping to be adjusted; then, instantiation and checking are executed on the revised artifacts;

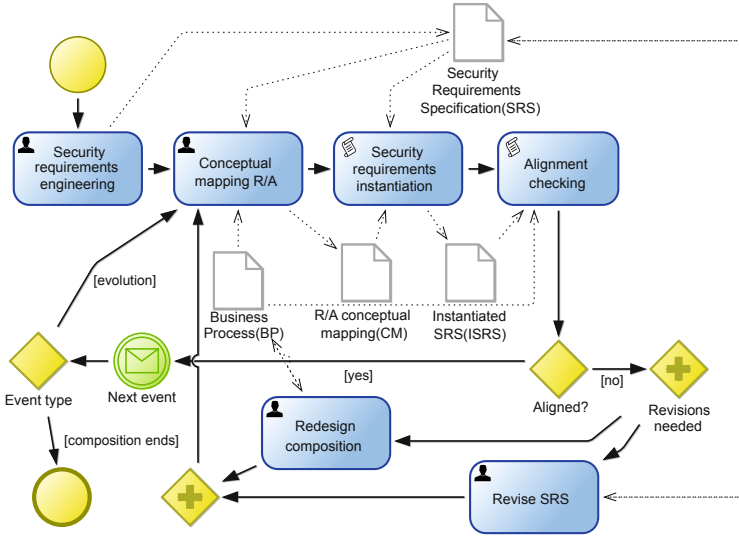


Fig. 6. Our method for supporting R/A alignment in presence of evolution

- *Alignment*: our framework waits for the next event to happen. If the event is the occurrence of an evolutionary action, the process re-starts from the conceptual mapping; if the composition terminates, the process ends.

In this paper, we detail the activities in the upper part of Fig. 6, which constitute our basic framework. We leave to future work the activities concerned with fixing non-compliance and repeating the check when an artifact evolves.

4.1 Conceptual Mapping R/A

This activity is carried out by an analyst to define a conceptual mapping CM between the SRS and the BP . The creation of CM is necessary because the architectural model is at a different level of abstraction from the requirements specification. Therefore, the skills of a human analyst to reconcile these abstraction level gap are required. CM consists of a set of relations between BP and SRS elements:

- *Participants to actors*: for each participant in BP , the analyst identifies links with the actors appearing in SRS as debtor or creditor. To do so, BP participants shall be specialized into agents or roles. Two possible relationships are possible: (i) *is-a* relates one or more BP roles (agents) to one SRS role (agent); (ii) *plays* links one or more BP agents to one or more SRS roles;
- *Activities to goals*: all activities in BP shall be linked to relevant goals in SRS (if any relevant goal exists). The relationship we support is called *relates-to*, and indicates that the activity is performed in order to achieve a goal. An activity may be related to several goals, and a goal to several activities;

- *Variables to documents*: each variable appearing in *BP* is linked to zero or one documents in *SRS* via a *represents* relationship, indicating that the variable represents a document.

Table 2. R/A conceptual mapping for the eGovernment scenario

plays (Athens REA, Seller), plays (Athens Munic., Municipality), is-a (eGov,eGov application), is-a (Storage, eGov application)
relates-to (Create contract draft, Draft prepared), relates-to (Review draft, Draft prepared), relates-to (Examine draft, Resid zone checked), relates-to (Insert approval ID, Approval provided), relates-to (Add duty stamp, Contract finalized), relates-to (Send copy to ministry, Government notified), relates-to (Scan contract, Government notified), relates-to (Archive contract copy, Government notified)
represents (V1, Contract draft), represents (V2, Building approval), represents (V3, Official contract)

Table 2 shows a possible *CM* between the *SRS* in Table 1 and the *BP* in Fig. 4. There are two *BP* participants (*eGov* and *Storage*) which are linked (by an *is-a* relation) to the agent *eGov application* in *SRS*. The *BP* participant *Greek ministry* is not linked to any actor in *SRS*. Among the mapped activities, both *Create contract draft* and *Review draft* are related to goal *Draft prepared*.

Security requirements are not directly mapped to security activities. These requirements correspond to patterns and anti-patterns which will be checked using algorithms like the ones described in Sec. 5.

4.2 Security Requirements Instantiation

The conceptual mapping *CM* enables to instantiate *SRS* on the service composition *BP*, i.e., understanding these requirements in terms of *BP* concepts (e.g., participant, activity, variable). Instantiation of security requirements comprises two main steps: (i) instantiation of debtors and creditors of security requirements, according to *CM*, into participants in *BP* (see Sec. 4.2; and (ii) instantiating the security needs, originally expressed on *SRS* actors, to *BP* participants (see Sec. 4.2). As a result, these steps return an instantiated specification *ISRS*.

Debtor and Creditor Instantiation. Each commitment in *SRS* is instantiated, based on *CM*, by checking the corresponding *BP* performers associated with the debtor and creditor of that commitment. If *CM* says that a debtor actor in an *SRS* commitment is linked to two *BP* performers, that commitment will be instantiated twice, as each performer has to create a commitment instance.

Algorithm 1 (function INSTANTIATEDEBTOR) shows how the debtor instantiation is performed, and returns an instantiated set of commitments *instCommitments*. If the debtor is a role (lines 2-3), the *CM* is searched for, in order to identify all performers who play that role. These performers are added to the set *bpPerformers_{deb}*. Then (line 4), *is-a* relationships involving the debtor

Algorithm 1 Debtor instantiation for a commitment

```

INSTANTIATEDEBTOR( $C(\text{deb}, \text{cred}, p, q)$ ,  $CM$ )
1   $instCommitments \leftarrow \emptyset$ 
2  if  $\text{TYPEOF}(\text{deb}) = \text{role}$  then
3     $bpPerformers_{deb}.ADD(CM.SEARCH(\text{plays}(*, \text{deb})))$ 
4     $bpPerformers_{deb}.ADD(CM.SEARCH(\text{is-a}(*, \text{deb})))$ 
5    for each  $\text{perf} \in bpPerformers_{deb}$  do
6       $instCommitments.ADD(c(\text{perf}, \text{cred}, p, q))$ 
7  return  $instCommitments$ 

```

are searched for, and the set $bpPerformers_{deb}$ is enriched. For each of these performers (lines 5-6), an instance of the original commitment is created with that performer as debtor; the commitment instance is added to $instCommitments$.

The returned set of commitments is processed to instantiate the creditor. The algorithm for the creditor is analogous to Algorithm 1, with the only difference that the creditor is replaced (instead of the debtor).

Example 2 (Actors instantiation). Consider commitment $C_1: C(\text{eGov application, Seller, non-discl}(\text{Sale information}))$ from Table 1, and the mapping in Table 2. The *SRS* role *eGov application* is linked to two participants: *eGov* and *Storage*. The instantiation of C_1 creates two commitments:

$C_{1.1}: C(\text{eGov, Athens REA, non-discl}(\text{Sale information}))$
 $C_{1.2}: C(\text{Storage, Athens REA, non-discl}(\text{Sale information}))$

The debtor and creditor of these commitments are now referring to *BP*. \square

Security Need Instantiation. After the commitments have been instantiated with respect to debtor and creditor, they are instantiated with respect to the security need in the consequent. The general form of a consequent is a parametric predicate: $sec\text{-need}(par_1, \dots, par_n)$. Unlike the previous instantiation activities, we cannot devise a generic algorithm for security needs. We detail specific algorithms in Sec. 5. Here, we illustrate the problem on two security need types:

- Separation of duties: $C(\text{Seller, Municipality, SoD}(\text{Approval provided, Draft prepared}))$. The instantiation algorithm will create $n \times m$ commitments, where n is the number of activities that relate to *Approval provided* and m the number of activities that relate to *Draft prepared*;
- Need-to-know: $C(\text{Seller, municipality, NtK}(\text{Municipal approval, Draft prepared}))$. Even if there are p activities related to the goal ($p > 1$), a single commitment is created. In our example, the instance will contain the set of activities $\{\text{Create contract draft, Review draft}\}$.

5 Compliance Checking

We detail two algorithms to check R/A alignment—to execute alignment checking in Fig. 6—for two security requirements types in STS-ml: non-disclosure of information (Sec. 5.1) and non-repudiation of a delegation (Sec. 5.2).

These algorithms need the conceptual mapping CM , the security requirements specification SRS (and its instantiated version $ISRS$), and the business process of a composition BP . The result is the compliance status of BP . A possible extension could return also the cause for non-compliance.

The service composition architecture BP is a BPMN model extended with security-related activities. Typically, BP would be defined by enriching a regular business process model. The security activities are special types of activities meant to guarantee security and reliability; for instance, sending an acknowledgement, encrypting some data, replicating stored data, etc. In [18], these activities are graphically represented as annotations on regular activities.

In general, a security requirement is satisfied if BP exhibits a specific structural pattern, either in terms of the sequencing of activities (the control flow), the information flow, the included security activities, and the assigned performers. Our algorithms enable verifying the presence (absence) of specific patterns (anti-patterns). Our ultimate objective is to build a repository of algorithms to allow checking all the security requirements supported by STS-ml.

5.1 Non-disclosure

In STS-ml, a non-disclosure security requirement is a commitment $C(\text{deb}, \text{cred}, \top, \text{non-disc}(\text{info}))$, where actor deb commits to actor cred that information info will not be distributed to other actors than cred or the owner of info .

Algorithm 2 Non-Disclosure instantiation

```

INSTANTIATEND( $C(\text{deb}, \text{cred}, \top, \text{non-disc}(\text{info}))$ ,  $CM$ ,  $SRS$ )
1   $\text{instCommitments} \leftarrow \emptyset$ 
2   $\text{documents} \leftarrow SRS.\text{SEARCH}(\text{tangible-by}(\text{info}, *))$ 
3  for each  $\text{doc} \in \text{documents}$ 
4  do  $\text{bpVariables} \leftarrow CM.\text{SEARCH}(\text{represents}(*, \text{doc}))$ 
5     for each  $\text{var} \in \text{bpVariables}$ 
6     do  $\text{instCommitments}.\text{ADD}(C(\text{deb}, \text{cred}, \top, \text{non-disc}(\text{var})))$ 
7  return  $\text{instCommitments}$ 

```

Algorithm 2 (INSTANTIATEND) takes in input a non-disclosure requirement, CM , and SRS (after debtor and creditor instantiation). It searches SRS for all the documents that make tangible the information (line 2). For each document (lines 3-6), CM is searched for variables representing that document (line 4). A non-disclosure commitment instance is created (line 6) for each of those variables.

The commitments returned by Algorithm 2 feed Algorithm 3, which checks whether that commitment is satisfied or not by BP . If any commitment is not satisfied, R/A alignment does not hold. Given a commitment instance $C(\text{deb}, \text{cred}, \top, \text{non-disc}(\text{var}))$ and the process BP , Algorithm 3 determines whether there is at least one activity, performed by the debtor deb , that transfers variable var to an activity executed by a performer that differs from the variable owner

Algorithm 3 Non-Disclosure Verification

```

VERIFYND( $C(\text{deb}, \text{cred}, \top, \text{non-disc}(\text{var}))$ ),  $BP$ ,  $SRS$ ,  $CM$ )
1   $\text{actByDeb} \leftarrow BP.ACTIVITIESBY(\text{deb})$ 
2   $\text{actByCred} \leftarrow BP.ACTIVITIESBY(\text{cred})$ 
3   $\text{actUsingVar} \leftarrow BP.ACTIVITIESUSING(\text{var})$ 
4   $\text{doc} \leftarrow CM.SEARCH(\text{represents}(\text{var}, *))$ 
5  if  $\text{doc} \neq \text{null}$ 
6    then  $\text{info} \leftarrow SRS.SEARCH(\text{tangible-by}(*, \text{doc}))$ 
7      for each  $i \in \text{info}$ 
8        do  $\text{own} \leftarrow SRS.SEARCH(\text{owns}(*, i))$ 
9           $\text{actByOwner.ADD}(BP.ACTIVITIESBY(\text{own}))$ 
10  $\text{actByOthers} \leftarrow \text{actUsingVar} \setminus \text{actByDeb} \setminus \text{actByCred} \setminus \text{actByOwner}$ 
11 for each  $a_i \in \text{actByDeb}$ 
12 do for each  $a_j \in \text{actByOthers}$ 
13   do if  $\text{var} \in \text{output}(a_i) \cap \text{input}(a_j)$ 
14     then return non-compliant
15 return compliant

```

or the creditor cred . In order to check this, a number of sets of activities are defined in the algorithm by querying BP (e.g., actByDeb indicates all activities performed by deb), CM (e.g., doc is the document that the variable represents, if any), and SRS (e.g., info is the set of information the document makes tangible).

Example 3 (Checking non-disclosure). Suppose an evolution of the security requirements occurs. Take BP as in Fig. 4, CM as in Table 2, and the evolved requirement $C_3:C(\text{eGov application, Seller, } \top, \text{non-disc}(\text{Sale information}))$. By executing the instantiation algorithms for debtor (Algorithm 1), creditor, and security need (Algorithm 2), we obtain the following four commitment instances:

$C_{3.1}:C(\text{eGov, Athens REA, } \top, \text{non-disc}(V1))$
 $C_{3.2}:C(\text{eGov, Athens REA, } \top, \text{non-disc}(V3))$
 $C_{3.3}:C(\text{Storage, Athens REA, } \top, \text{non-disc}(V1))$
 $C_{3.4}:C(\text{Storage, Athens REA, } \top, \text{non-disc}(V3))$

By running Algorithm 3 on the four commitments, it returns compliance for $C_{3.1}$, $C_{3.3}$, and $C_{3.4}$, while it returns non-compliance for $C_{3.2}$. Indeed, $eGov$'s activity "Send copy to ministry" transfers $V3$ to activity "Check copy" performed by the *Greek ministry*, who is neither the owner of $V3$ nor the creditor of $C_{3.2}$.

In order to re-establish R/A alignment, either BP is re-designed so that $V3$ is not transferred to the *Greek ministry*, or the non-disclosure requirement is relaxed, by allowing sale information to be re-distributed. \square

5.2 Non-Repudiation

Non-repudiation in STS-ml is defined as a commitment $C(d, c, \text{del}=\text{delegate}(c,d,g), \text{non-rep}(\text{del}))$, where actor d commits to actor c that the delegator (c) will be provided with a proof that the delegation of goal g is acknowledged by d (so that d cannot able repudiate the delegation later on).

Algorithm 4 Non-Repudiation instantiation

```

INSTANTIATENR( $C(d, c, del=delegate(c,d,g), non-rep(del))$ ,  $CM$ )
1   $instCommitments \leftarrow \emptyset$ 
2   $activities \leftarrow CM.SEARCH(relates-to(*, g))$ 
3  for each  $act \in activities$ 
4  do  $inst \leftarrow C(d, c, del=delegate(c,d,act), non-rep(del))$ 
5      $instCommitments.ADD(inst)$ 
6  return  $instCommitments$ 

```

Function INSTANTIATENR (Algorithm 4) takes in input a non-repudiation commitment and CM . CM is searched for all activities that are linked to the delegated goal g via a *relates-to* relationship. For each of these activities (lines 3-5), a commitment instance is created where the debtor d commits not to repudiate the delegation of that activity.

Algorithm 5 Non-repudiation verification

```

VERIFYNR( $actNR, perf, actCurr, found, visited$ )
1  switch TYPEOF( $actCurr$ )
2    case  $ack$  :
3      if ( $actCurr \in ACKFOR(actNR) \wedge PERF(actCurr) = perf$ )
4        then return true
5    case  $end$  :
6      return not found
7    case  $default$  :
8      if ( $actCurr = actNR$ )
9        then found  $\leftarrow true$ 
10  $next \leftarrow NEXTACTIVITIES(actCurr) \setminus visited$ 
11 if  $next = \emptyset$  then return true
12 switch TYPEOF(NEXTELEMENT( $actCurr$ ))
13   case  $gway-excl$  :
14     return  $\bigwedge_{a \in next} VERIFYNR(actNR, perf, a, found, visited \cup actCurr)$ 
15   case  $gway-incl$  :
16     return  $\bigvee_{a \in next} VERIFYNR(actNR, perf, a, found, visited \cup actCurr)$ 
17   case  $activity$  :
18      $VERIFYNR(actNR, perf, GETFIRST(next), found, visited \cup actCurr)$ 

```

Each instantiated non-repudiation commitment—which refers to a specific activity $actNR$ —is verified on BP by the recursive function VERIFYNR (Algorithm 5). The commitment is satisfied when, for each path from start to end in BP that includes that activity, there is an *acknowledge* for $actNR$ made by the executor of $actNR$. The performer of the *acknowledge* activity certifies the delegator that the delegation has taken place. A possible implementation is notifying a workflow engine about the delegation acceptance.

The parameters of VERIFYNR are the activity for which an ack is needed ($actNR$); the debtor in the non-repudiation commitment ($perf$); the currently

examined activity in BP ($actCurr$); a boolean variable indicating if, in the path the algorithm is exploring, $actNR$ was encountered ($found$); and the set of activities the algorithm has already encountered ($visited$). This last variable enables dealing with cycles in BP . Given a commitment for the non-repudiation of act , in which the debtor is deb , the algorithm is initially invoked as $VERIFYNR(act, deb, start, false, \{start\})$.

If the current activity is an ack for the searched activity executed by $perf$, it returns true (lines 2-4). If it is the end activity, the algorithm returns false if the activity was found but the ack was not found, while it returns true if the activity was not found (lines 5-6). If the current activity is the searched one, then the variable $found$ is set to true: an ack is required (lines 7-9). If there are no subsequent activities, this means the algorithm has reached the end of a cycle; since cycles are supported only via exclusive gateways, the algorithm returns true, as such value does not affect the computation of compliance (lines 10-11). The recursive calls of the algorithm depend on the type of the next encountered element (line 12). If an exclusive gate is found, the algorithm is recursively called for all subsequent activities, and the compliance results are conjuncted, as compliance is needed in all paths (lines 13-14). If a parallel gateway is encountered, all outgoing paths are followed, and the results are disjuncted, as one ack suffices (lines 15-16). In case of an activity, the algorithm examines it (lines 17-18).

Example 4 (Checking non-repudiation). Suppose the architecture of the composite service evolves. We check the evolved BP in Fig. 4 with the non-repudiation commitment $C_1:C(eGov\ application, Seller, D=delegate(Seller, eGov\ application, Government\ notified), non-rep(D))$. The instantiation process—which includes Algorithm 4—returns the following commitments:

$C_{1.1}:C(eGov, Athens\ REA, non-rep(Send\ copy\ to\ ministry))$
 $C_{1.2}:C(eGov, Athens\ REA, non-rep(Scan\ contract))$
 $C_{1.3}:C(eGov, Athens\ REA, non-rep(Archive\ contract\ copy))$
 $C_{1.4}:C(Storage, Athens\ REA, non-rep(Send\ copy\ to\ ministry))$
 $C_{1.5}:C(Storage, Athens\ REA, non-rep(Scan\ contract))$
 $C_{1.6}:C(Storage, Athens\ REA, non-rep(Archive\ contract\ copy))$

By running Algorithm 5 on all the commitments, it returns that $C_{1.3}$, $C_{1.4}$, and $C_{1.5}$ are compliant: the debtor is not the performer of the activity, thus no acknowledge is required. $C_{1.1}$, $C_{1.2}$, and $C_{1.6}$ are not compliant: the activity specified in the commitment is executed but there is no corresponding acknowledge. To align the BP with $C_{1.1}$, $C_{1.2}$ and $C_{1.6}$, either BP is modified adding the needed acknowledge activities, or security requirement C_1 is removed. A concrete solution is to add an acknowledge activity *Send copy to ministry* between *Add duty stamp* and *Send copy to ministry*.

6 Discussion

We have proposed a methodological and semi-automated approach to align service-oriented architectures—specifically, service compositions—with security

requirements specifications—in particular, social commitments in STS-ml. While our approach addresses the entire R/A alignment problem, this paper focuses on checking alignment. Such activity is key in the era of software evolution, where both requirements and software systems are subject to unpredicted changes.

Our approach includes three steps: (i) an analyst creates a conceptual mapping between requirements and architecture—e.g., the activities that relate to a certain goal; (ii) algorithms are executed to check R/A alignment; and (iii) in case of non-alignment, evolutionary actions are taken by the analysts—or the system itself—to revise either the architecture or the requirements.

We have provided algorithms to check R/A alignment for two security requirements types: non-disclosure of information and non-repudiation of delegations. We have already investigated other types (need-to-know, fall-back and true redundancy) which were not shown due to space limitations.

Our approach complements methods to derive architectures from requirements (e.g. [2, 10, 16, 23]), which can be applied at design-time to define a suitable architecture for a given set of requirements. Our approach provides a continuous on-line alignment checking, which enables coping with evolution of both requirements and architecture.

Compliance is a hot topic in information security [12]. The effects of non-compliance are well known and existing empirical studies have investigated [21] how compliance is perceived by employees in organizations.

However, only recently the compliance between requirements and business processes has led to concrete research efforts. Liu et al [13] describe how to check the compliance between a set of formally expressed regulatory requirements and business processes. They created a tool which transforms (i) business process models, expressed with Business process Execution language (BPEL), in pi-calculus; (ii) regulatory requirements, expressed with Business Property Specification Language (BPSL), in linear temporal logic. This tool verifies the business process against these compliance rules by means of model-checking technology. Our approach takes a different yet orthogonal standpoint as it considers security requirements over interactions.

Ghose and Koliadis [8] enrich BPMN with annotations, then transform models created using such language into Semantic Process Network (SPNets). This allows for defining a class of proximity relations that highlight the extent to which evolutions of an original business process deviate. Unlike us, they focus only on the structural difference between processes, and they don't take into account security requirements.

Other approaches (e.g., [7]) tackle the evolution problem from a legal perspective. They propose a systematic approach to help organizations align their business processes with (privacy) laws. Unlike ours, however, their approach is off-line and mainly design-time.

Some algorithms are implemented in the Security Requirements Compliance Module (SRCM) tool. It currently supports non-repudiation and several authorization-related requirements: non-modification, non-usage, non-production and non-disclosure. Future versions of the module will support other require-

ments such as redundancy, separation of duties, binding of duties, integrity, etc. The tool takes as input three XML files: a specific version of BP, the SRS, and the CM. SRCM returns another XML file which contains the SRS commitments grouped in three sets: satisfied, violated, or undecidable. SRCM is implemented as an OSGi bundle², so as to facilitate integration with other tools.

Our future work includes extending the framework in many ways. First, we will devise algorithms to support different types of security requirements. Second, we will provide a complete implementation of the tool to support the analyst in the mapping phase as well as to check R/A alignment in a continuous on-line fashion. Third, we will investigate how service-oriented architectures can self-evolve to guarantee R/A alignment. Fourth, we will empirically evaluate the effectiveness of our approach on industrial case studies in the Air Traffic Management domain (from Aniketos). Fifth, we will extend our approach to include verifying alignment between architecture and implementation. We plan to use techniques such as bytecode verification to determine whether the security properties in the business process are correctly implemented.

Acknowledgement. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos) and 256980 (NESSoS).

References

1. Barais, O., Le Meur, A.F., Duchien, L., Lawall, J.: Software Architecture Evolution. In: Mens, T., Demeyer, S. (eds.) *Software Evolution*. LNCS, pp. 233–262. Springer, Heidelberg (2008)
2. Bastos, L.R.D., Castro, J.F.B.: Systematic Integration Between Requirements and Architecture. In: Choren, R., Garcia, A., Lucena, C., Romanovsky, A. (eds.) *SELMAS 2004*. LNCS, vol. 3390, pp. 85–103. Springer, Heidelberg (2005)
3. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) *CAiSE 2000*. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
4. Crook, R., Ince, D., Lin, L., Nuseibeh, B.: Security Requirements Engineering: When Anti-Requirements Hit the Fan. In: *Proc. of RE 2002*, pp. 203–205. IEEE (2002)
5. Dalpiaz, F., Paja, E., Giorgini, P.: Security Requirements Engineering via Commitments. In: *Proc. of STAST 2011* (2011)
6. Garg, A., Curtis, J., Halper, H.: Quantifying the Financial Impact of IT Security Breaches. *Information Management & Computer Security* 11(2), 74–83 (2003)
7. Ghanavati, S., Amyot, D., Peyton, L.: Compliance Analysis Based on a Goal-oriented Requirement Language Evaluation Methodology. In: *Proc. of RE 2009*, pp. 133–142 (2009)
8. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOC 2007*. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)

² <http://www.osgi.org/>

9. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling Security Requirements through Ownership, Permission and Delegation. In: Proc. of RE 2005, pp. 167–176. IEEE (2005)
10. Hall, J.G., Jackson, M., Laney, R.C., Nuseibeh, B., Rapanotti, L.: Relating Software Requirements and Architectures using Problem Frames. In: Proc. of RE 2002, pp. 137–144. IEEE (2002)
11. Harker, S.D.P., Eason, K.D., Dobson, J.E.: The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering. In: Proc. of RE 1993, pp. 266–272. IEEE (1993)
12. Julisch, K.: Security Compliance: the Next Frontier in Security Research. In: Proc. of the 2008 Workshop on New Security Paradigms, pp. 71–74. ACM (2008)
13. Liu, Y., Müller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. *IBM Systems Journal* 46(2), 335–361 (2007)
14. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In: Proc. of ACSAC 1999, pp. 55–64. IEEE (1999)
15. Mouratidis, H., Giorgini, P.: Secure Tropos: A Security-Oriented Extension of the Tropos methodology. *International Journal of Software Engineering and Knowledge Engineering* 17(2), 285–309 (2007)
16. Nuseibeh, B.: Weaving together requirements and architectures. *Computer* 34(3), 115–119 (2001)
17. Nuseibeh, B., Easterbrook, S.: Requirements Engineering: a Roadmap. In: Proc. of FOSE 2000, pp. 35–46. ACM (2000)
18. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security requirements in Business Processes. *IEICE Transactions on Information and Systems* 90(4), 745–752 (2007)
19. Sindre, G., Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases. *Requirements Engineering* 10(1), 34–44 (2005)
20. Singh, M.P.: An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law* 7(1), 97–113 (1999)
21. Siponen, M., Pahlila, S., Adam Mahmood, M.: Compliance with Information Security Policies: An Empirical Investigation. *Computer* 43, 64–71 (2010)
22. van Lamsweerde, A.: Requirements Engineering in the Year 2000: A Research Perspective. In: Proc. of ICSE 2000, pp. 5–19 (2000)
23. van Lamsweerde, A.: From System Goals to Software Architecture. In: Bernardo, M., Inverardi, P. (eds.) *SFM 2003*. LNCS, vol. 2804, pp. 25–43. Springer, Heidelberg (2003)
24. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: Proc. of ICSE 2004, pp. 148–157. IEEE (2004)

Looking into the Future^{*}

Using Timed Automata to Provide a Priori Advice about Timed Declarative Process Models

Michael Westergaard and Fabrizio Maria Maggi

Eindhoven University of Technology, The Netherlands
{m.westergaard,f.m.maggi}@tue.nl

Abstract. Many processes are characterized by high variability, making traditional process modeling languages cumbersome or even impossible to be used for their description. This is especially true in cooperative environments relying heavily on human knowledge. Declarative languages, like Declare, alleviate this issue by not describing what to do step by step but by defining a set of constraints between actions that must not be violated during the process execution. Furthermore, in modern cooperative business, time is of utmost importance. Therefore, declarative process models should be able to take this dimension into consideration. Timed Declare has already previously been introduced to monitor temporal constraints at runtime, but it has until now only been possible to provide an alert when a constraint has already been violated without the possibility of foreseeing and avoiding such violations. Moreover, the existing definitions of Timed Declare do not support the static detection of time-wise inconsistencies. In this paper, we introduce an extended version of Timed Declare with a formal timed semantics for the entire language. The semantics degenerates to the untimed semantics in the expected way. We also introduce a translation to timed automata, which allows us to detect inconsistencies in models prior to execution and to early detect that a certain task is *time sensitive*. This means that either the task cannot be executed after a deadline (or before a latency), or that constraints are violated unless it is executed before (or after) a certain time. This makes it possible to use declarative process models to provide a priori guidance instead of just a posteriori detecting that an execution is invalid.

Keywords: declarative process modeling, metric temporal logic, error detection, operational support, timed automata, Declare.

1 Introduction

Organizations work today in a dynamic, complex and interconnected world. Even in the heterogeneity of the environment where they operate, they need to execute

^{*} This research is supported by the Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

their processes in a trustworthy and correct manner. A compliance model is, in general, a set of *business constraints* that allow practitioners to discriminate whether a process instance behaves as expected or not.

During the execution of a business process it is often extremely important to meet deadlines and optimize response times, especially in cooperative environments where contracts among multiple parties need to be adhered to. To this aim, a compliance model can also include temporal constraints to guarantee the correct execution of a process in terms of *latencies* (related to events that cannot occur *before* a certain time, or must occur *after* a certain time) and *deadlines* (related to events that cannot occur *after* a certain time, or must occur *before* a certain time).

Nevertheless, the declarative nature of business constraints makes it difficult to use procedural languages to describe compliance models. First, the integration of diverse and heterogeneous constraints would quickly make models extremely complex and tricky. Second, business constraints often target uncontrollable aspects, such as activities carried out by internal autonomous actors (e.g., a doctor in a health-care process) or even by external independent entities (e.g., a web service in a service choreography). Representing this variability through a procedural model would require the explicit specification in the same model of multiple alternatives. Again, this would make models completely unreadable.

For this reason, in this paper, we represent business constraints using *Declare* [9,10,1,13]. Declare is a declarative language that combines a formal semantics grounded in Linear Temporal Logic (LTL) with a graphical representation for users. Differently from procedural models, a Declare model describes a process as a list of constraints that must be satisfied during the process execution. A Declare model is an “open world” where everything is allowed unless it is explicitly forbidden. In this way, Declare is very suitable for describing compliance models.

Nevertheless, the standard LTL semantics of Declare is not sufficient to represent metric temporal constraints, i.e., constraints that specify latencies and deadlines on the execution of the activities of a business process. Therefore, in order to monitor such a kind of constraints in Declare, it is necessary to represent them through a more expressive formal semantics. For example, in [8,5], the authors use the Event Calculus (EC). However, this approach allows users to identify a violation only *after* it has occurred and it is not possible to prevent violations from taking place. Moreover, by using the EC, it is not possible to detect violations that cannot be ascribed to an individual constraint but are determined by the interplay of two or more constraints.

To address these issues, the approach presented in this paper uses timed automata [2] instead of the EC to evaluate the compliance of a process instance w.r.t. a (timed) Declare model at runtime. In particular, to express metric temporal constraints in Declare, we extend the original LTL semantics of Declare with MTL (Metric Temporal Logic) [7,3], a real-time extension of LTL with quantitative temporal operators. MTL reasons over infinite traces. In contrast, traces in a business process are supposed to finish sooner or later. Therefore,

we use a variant of MTL for finite traces first introduced in [11]. This semantics produces as output a boolean value representing whether the current (finite) trace complies with the monitored property or not. In addition, we extend MTL for finite traces with the four valued semantics RV-MTL (Runtime Verification MTL), in order to respect the fact that it is not always possible to produce at runtime a definitive answer about compliance.

We monitor RV-MTL rules through timed automata. However, we show with some counterexamples that RV-MTL is undecidable, i.e., it is not possible to translate every RV-MTL rule to timed automata. For this reason, we restrict our perspective to the set of rules that we use to formally represent the semantics of Timed Declare. For this (limited) set of rules, we present automata to monitor them at runtime and check models a priori.

While evaluating the compliance of a running process instance w.r.t. a Declare model, users are allowed, using timed automata, to “look into the future” from two different perspectives. First of all, using timed automata, it is possible to generate a (red) alert to warn users that a constraint (or a combination of constraints) is going to be violated. In this way, they are advised to undertake specific actions within a specific lapse of time *before* the violation has occurred so that the violation can be avoided. We can also generate alerts with a lower severity (yellow or orange), i.e., alerts to warn users that a specific activity can currently be executed but, in the future, it will be (temporarily or permanently) forbidden. Secondly, our approach allows *early detection* of violations. In fact, using timed automata, it is possible to detect *non-local violations* when still none of the individual constraints in the compliance model has been violated. A non-local violation is a violation that cannot be ascribed to an individual constraint but is determined by the interplay of two or more constraints and indicates that (at least) one of them will be violated in the future.

The paper is structured as follows. Section 2 introduces some background notions about automata, timed automata and MTL. In this section, we also present RV-MTL. In Sect. 3, we present the semantics of Timed Declare and automata to check individual constraints. In Sect. 5, we outline our prototype implementation of Timed Declare, and in Sect. 6, we sum up our conclusions and provide directions for future work.

2 Background

In this section, we introduce some background material. We first present standard Declare using a running example. Then, we introduce timed automata and MTL (Metric Temporal Logic), the temporal logic we use in this paper. We also present a runtime version of MTL, which extends MTL to a four-valued logic for handling ongoing traces.

2.1 Declare and Running Example

Declare is a workflow language and tool [13] for modeling workflows using a declarative approach. Instead of specifying what has to be done, constraints

between tasks are specified. In Fig. 1, we see a simple Declare model (ignore the intervals for now) of an ordering process in a web shop. Here we have five tasks, specified using rectangles (e.g., Order) and five constraints, specified either using arrows or as the house annotation above Discount. The constraints specify when certain tasks are allowed or required. For example, we have a *precedence* constraint from Order to Pay, indicating that we can only pay after placing an order (we do not have to pay after placing an order, as we can cancel it though this is not explicitly modeled). We have a *succession* constraint from Pay to Delivery, indicating that if an order has been paid, it must be delivered, and it can only be delivered after successful payment. Furthermore, we can only get a discount if we order something (as specified by the *precedence* constraint from Order to Discount), but if we get a discount, we have to sign up for subsequent advertisement, as indicated by the *response* constraint from Discount to Advertisement. The house above Discount restricts how many times this task can occur; in this case the restriction is that it must occur zero or more times, which does not mean anything for an untimed version of the model, but which shall become useful later.

2.2 Timed Automata

A timed automaton augments standard finite automata with a set of clocks. Clocks all run at the same rate and are typically denoted by c . While we cannot control the progression of clocks, we can observe and reset them. We can also perform actions depending on *clock constraints*, which compare the value of a clock with any integer:

Definition 1 (Clock Constraints). Given a single clock c , the set of **clock constraints** over c are

$$\mathcal{B}(c) = \{c \sim n \mid n \in \mathbf{N}, \sim \in \{\leq, <, =, >, \geq\}\}.$$

A timed automaton extends standard finite automata by adding *invariants* as clock constraints to states and adding *guards* as clock constraints to transitions:

Definition 2 (timed Automaton). A *timed automaton* is a septuple:

$$\mathcal{TA} = (S, AP, C, \delta, I, s_I, A)$$

where S is a finite set of **states** (also called **locations**), AP is a finite set of **labels**, C is a set of clocks, $\delta \subseteq S \times 2^{\mathcal{B}(C)} \times AP \uplus \{\tau\} \times 2^{\{C\}} \times S$ is the **transition**

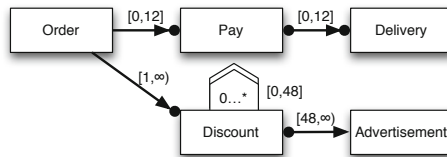


Fig. 1. Running example: Declare model consisting of four constraints

relation where each transition has a set of clock constraints as **guards** and a label, $I : S \rightarrow \mathcal{B}(C)$ assigns **invariants** to states, s_I is the **initial state** and $A \subseteq S$ is the set of **accepting states**.

The intuition is that time progresses at a constant and uncontrollable rate. We are only allowed to stay in a state as long as the state invariant holds and can only follow a transition if the guard constraint is true at the current time. The distinguished transition τ represents time passing and corresponds to an invisible transition, i.e., we can follow such a transition without consuming events as long as the guard constraint is true. When following a transition, we reset all clocks in the fourth component of the transition.

Often, we represent timed automata as directed graphs where nodes correspond to states and arcs to transitions. An example of a timed automaton is shown in Fig. 2. We have 4 labeled states and 8 transitions. We indicate the initial state using an unrooted arrow (s_0) and accepting states using a double outline (s_0 and s_2). Invariants are shown in brackets below states (e.g., $[xA \leq a]$ below s_1). Next to transitions, we show their labels, guards, and clocks to reset. For example, the transition from s_2 to s_0 has label τ (indicated by $:\tau$), has guard $yA > a$, and resets no clocks. The transition from s_0 to s_1 resets both xA and yA . We can have any number of guards and clock resets, including none (e.g., the transition from s_0 to s_3). The dashed state (s_3) indicates that no accepting state is reachable from there.

We interpret timed automata over *timed sequences*, i.e., strings over $\mathbf{R}_+ \times AP$ where $\mathbf{R}_+ = \{x \in \mathbf{R} \mid x \geq 0\}$, denoted by $\sigma = (t_0, p_0)(t_1, p_1) \cdots (t_{k-1}, p_{k-1})$. We require that for $i < j$ we have $t_i \leq t_j$. The intuition is the same as for regular automata; we follow states from the initial state. However, we also introduce steps happening automatically due to time progressing.

Formally, we consider triples $(t, s, V) \in \mathbf{R}_+ \times S \times C_+^{\mathbf{R}}$, where the first entry is the absolute timestamp, the second is the state and the third includes the values of the clocks. We allow 2 kinds of transitions:

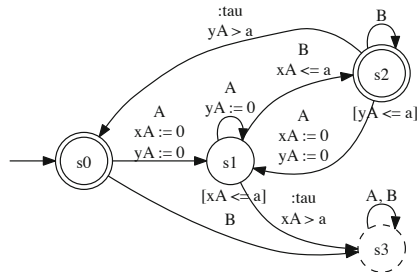


Fig. 2. Timed automaton for the $succession_{[0,a]}(A, B)$ constraint

$$\begin{aligned}
(t, s, V) &\rightarrow^d (t + d, s, V + d) && \text{if } \forall t' \in [V, V + d], I(s)(t') = \text{true, and} \\
(t, s, V) &\rightarrow^a (t, s', V') && \text{if } (s, \gamma, a, C, s') \in \delta, \gamma(V) = \text{true, and} \\
&&& I(s')(V') = \text{true with } V'(c) = \begin{cases} 0 & \text{if } c \in C, \text{ and} \\ V(c) & \text{otherwise.} \end{cases}
\end{aligned}$$

Note that the second case also includes invisible steps. If either of the two cases hold, we write $(t, s, V) \rightarrow (t', s', V')$. A *trace* $(t_0, s_0, v_0) \rightarrow (t_1, s_1, v_1) \rightarrow \dots \rightarrow (t_{k-1}, s_{k-1}, v_{k-1})$ is *accepting* if $(t_0, s_0, V_0) = (0, s_I, \mathbf{0})$ and $s_{k-1} \in A$. A timed sequence $\sigma = (t_0, p_0)(t_1, p_1) \dots (t_{k-1}, p_{k-1})$ is accepting if there exists an accepting trace $T = (t'_0, s_0, V_0) \rightarrow (t'_1, s_1, V_1) \rightarrow \dots \rightarrow (t'_{n-1}, s_{n-1}, V_{n-1})$ such that $\sigma = \text{project}(T)$ where *project* projects the trace onto the first two components and ignores τ steps.

2.3 Metric Temporal Logic

Metric Temporal Logic (MTL) is a logic talking about timed sequences of states. The idea is that we have a set of *atomic propositions*, denoted by $AP = \{p_0, p_1, \dots, p_{n-1}\}$. For our variant of metric temporal logic, we look at timed sequences of *events* and assume that events fall in the set of atomic propositions. We deal with a fragment of MTL where all traces are finite. Therefore, we use the MTL semantics for dealing with finite timed sequences presented in [11].

To express MTL formulas, we use the syntax:

Definition 3 (MTL Syntax). *Formulas of MTL contain **atomic propositions** and are closed under negation, conjunction, disjunction, timed next operator, timed until operator, timed previous/yesterday operator and timed since operator, i.e., a **formula** ψ belongs to MTL if*

$$\psi ::= p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \mathbf{X}_I\psi \mid \psi_1 \mathbf{U}_I\psi_2 \mid \mathbf{Y}_I\psi \mid \psi_1 \mathbf{S}_I\psi_2$$

where $p \in AP$, $\psi, \psi_1, \psi_2 \in \text{MTL}$, and $I \subseteq \mathbf{R}_+$ is an interval.

Let $\sigma = (t_0, p_0)(t_1, p_1) \dots (t_{k-1}, p_{k-1})$ be a finite timed sequence of states and let ψ be an MTL formula. We write $\sigma, i \models \psi$ to indicate that ψ holds at position i in σ . The semantics of $\sigma, i \models p$, $\sigma, i \models \neg\psi$, $\sigma, i \models \psi_1 \wedge \psi_2$ and $\sigma, i \models \psi_1 \vee \psi_2$ are as normally in propositional logic: p is true at position i in σ if $p = p_i$, $\neg\psi$ is true if ψ is not, $\psi_1 \wedge \psi_2$ is true if both ψ_1 and ψ_2 are and $\psi_1 \vee \psi_2$ if either is. We say that $\sigma, i \models \mathbf{X}_I\psi$, if (t_i, p_i) has a successor state $(i < k - 1)$ and $\sigma, i + 1 \models \psi$ with $t_i + a \leq t_{i+1} \leq t_i + b$. Moreover, $\sigma, i \models \mathbf{Y}_I\psi$, if (t_i, p_i) has a predecessor state $(i > 0)$ and $\sigma, i - 1 \models \psi$ with $t_i - b \leq t_{i-1} \leq t_i - a$. We say that $\sigma, i \models \psi_1 \mathbf{U}_I\psi_2$, if $\sigma, j \models \psi_2$ for some $j \geq i$ with $t_i + a \leq t_j \leq t_i + b$ and $\sigma, l \models \psi_1$ for all $i \leq l < j$. Finally, $\sigma, i \models \psi_1 \mathbf{S}_I\psi_2$, if $\sigma, j \models \psi_2$ for some $j \leq i$ with $t_i - b \leq t_j \leq t_i - a$ and $\sigma, l \models \psi_1$ for all $j < l \leq i$. This semantics coincides with FLTL (LTL for finite traces) where only $I = \mathbf{R}_+$ is allowed.

We add syntactic sugar for the normal connectives, such as $\psi_1 \rightarrow \psi_2 \equiv (\neg\psi_1) \vee \psi_2$ and $\psi_1 \leftrightarrow \psi_2 \equiv (\psi_1 \rightarrow \psi_2) \wedge (\psi_2 \rightarrow \psi_1)$. We also add temporal syntactic sugar, $\mathbf{F}_I\psi \equiv \text{true}\mathbf{U}_I\psi$ (timed future operator), $\mathbf{G}_I\psi \equiv \neg(\mathbf{F}_I(\neg\psi))$ (timed globally operator), $\mathbf{O}_I\psi \equiv \text{true}\mathbf{S}_I\psi$ (timed once operator) and $\mathbf{H}_I\psi \equiv \neg(\mathbf{O}_I(\neg\psi))$ (timed historically operator). The intuition behind the future operator and the once operator is that ψ has to happen in the specified interval of time from now (in the future or in the past). The intuition behind the globally operator and the historically operator is that ψ has to hold for the entire interval (in the future or in the past).

2.4 RV-MTL: A Metric Temporal Logic for Runtime Verification

When focusing on runtime verification of MTL properties, reasoning is carried out on partial, ongoing traces, which describe a portion of the system's execution. Therefore, here, we extend MTL (for finite traces) with a four-valued semantics called *Runtime Verification Metric Temporal Logic* (RV-MTL). Differently from the original MTL semantics, which gives to the user only a boolean feedback (specifying whether a trace is compliant or not w.r.t. a given property), RV-MTL provides more sophisticated diagnostics.

Let $\sigma = (t_0, p_0)(t_1, p_1) \cdots (t_{k-1}, p_{k-1})$ be a finite timed sequence of states and let ψ be an MTL formula. The semantics of $[\sigma \models \psi]_{RV}$ is defined as follows:

- $[\sigma \models \psi]_{RV} = \top$ if for each possible continuation w of σ : $\sigma w \models \psi$ (in this case ψ is *permanently satisfied* by σ);
- $[\sigma \models \psi]_{RV} = \perp$ if for each possible continuation w of σ : $\sigma w \not\models \psi$ (in this case ψ is *permanently violated* by σ);
- $[\sigma \models \psi]_{RV} = \top^p$ if $\sigma \models \psi$ but there is a possible continuation w of σ such that $\sigma w \not\models \psi$ (in this case ψ is *possibly satisfied* by σ);
- $[\sigma \models \psi]_{RV} = \perp^p$ if $\sigma \not\models \psi$ but there is a possible continuation w of σ such that $\sigma w \models \psi$ (in this case ψ is *possibly violated* by σ).

3 Timed Declare

In this section we introduce a timed version of Declare. The version is similar to the one in [8], but we allow time on more constraints and instead give a semantics which collapse to the standard LTL semantics when removing time. We also introduce new constraints that are useful when dealing with time.

Returning to the running example in Fig. 1, we see that the constraints all have intervals next to them. This represents *when* things have to occur. For example, we indicate that payment has to be performed within 12 time units after the initial order (for example, because orders without payment are purged after 12 time units), and shipment has to take place within 12 time units after payment. Processing a discount cannot occur earlier than 1 time unit after the order. Sending out advertisements is only performed 48 time units after the discount. Finally, we have a new constraint, *exclusive allowance*, which states that the discount can only be applied in the first 48 time units of the process.

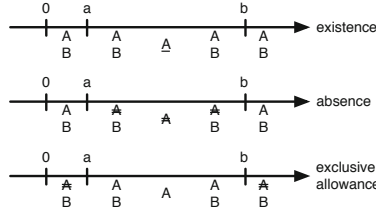


Fig. 3. Existence, Absence, and Exclusive Allowance

In Fig. 3, we give a graphical representation of the semantics for the timed existence and absence, and a new constraint which only makes sense in the timed version, exclusive allowance. The *timed existence* indicates that, starting from the beginning of a process instance, *A* must occur at least once (indicated by underline) at some point $t \in I$ where I is some interval from a to b (either of which may be included and b may also be ∞). *A* is allowed outside this interval (as is any other event, indicated by *B* in the figure). The *timed absence* specifies that *A* must not occur in the interval I (indicated by a double strikeout). *A* is allowed outside this interval. Where existence forces something to happen, it may also be useful to just allow something to happen in a specific interval, i.e., consider the conjunction of absence in the intervals before and after. This yields the *exclusive allowance*, which specifies that *A* is only allowed inside the interval I (e.g., $exclusive\ allowance_{[2,7]}(A) \equiv \wedge absence_{[0,2]}(A) \wedge absence_{(7,\infty]}(A)$). In our example in Fig. 1, we use exclusive allowance to only allow for a discount within the first 48 time units (though here it is equivalent to absence after this interval).

Figure 4 shows a graphical representation of the semantics for the *timed responded existence*. This constraint indicates that, if *A* occurs at time t_1 , *B* must occur at some point $t_0 \in [t_1 - b, t_1 - a]$ or $t_2 \in [t_1 + a, t_1 + b]$ (assuming $I = [a, b]$; if the interval is semi-open the intervals for t_0 and t_2 need to be updated accordingly). In the interval $[t_0, t_2]$ another *A* or another *B* can occur and, also, any event different from *A* and *B* (indicated by *C* in the figure). For the sake of readability, in this representation, we do not specify the behavior outside the interval $[t_0, t_2]$ where any event can occur. This semantics must be valid for each *A* in a process instance. The *timed co-existence* (which is not shown) is the conjunction of the timed responded existence with parameters (*A*, *B*) and the timed responded existence with parameters (*B*, *A*).

Figure 5 shows the semantics for the timed response, the timed alternate response and the timed chain response. The *timed response* indicates that, if *A*

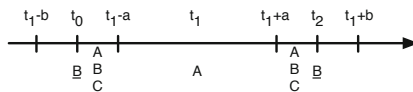


Fig. 4. Responded Existence

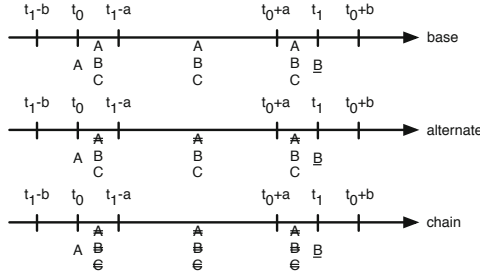


Fig. 5. Response, Alternate Response, Chain Response

occurs at time t_0 , B must occur at some point $t_1 \in I$. Any event can occur inside this interval. In all representations in Fig. 5 (and also in the ones in Fig. 6) we do not specify the behavior outside the interval $[t_0, t_1]$, because outside this interval any event can occur. The *timed alternate response* specifies that if A occurs at time t_0 , B must occur at some point $t_1 \in I$. A is not allowed in the interval $[t_0, t_1]$. Any event different from A is allowed. The *timed chain response* indicates that, if A occurs at time t_0 , B must occur next at some point $t_1 \in [t_0 + a, t_0 + b]$. Nothing is allowed between A and B . Each of these constraints must hold for each A .

In Fig. 6, we give a graphical representation of the semantics for the timed precedence, the timed alternate precedence and the timed chain precedence. The *timed precedence* indicates that, if B occurs at time t_1 , A must occur at some point $t_0 \in I$. Any event can occur between A and B . The *timed alternate precedence* specifies that, if B occurs at time t_1 , an A must occur at some point $t_0 \in [t_1 - b, t_1 - a]$. B is not allowed in the interval $[t_0, t_1]$. Any event different from B is allowed. The *timed chain precedence* indicates that, if B occurs at time t_1 , A must occur immediately before at some point $t_0 \in [t_1 - b, t_1 - a]$. Other events are not allowed in between.

The *timed succession*, the *timed alternate succession* and the *timed chain succession* (which are not shown in the table for the sake of readability) can be

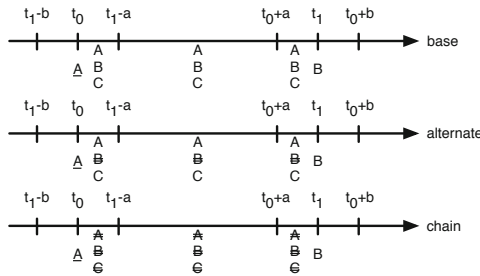


Fig. 6. Precedence, Alternate Precedence, Chain Precedence

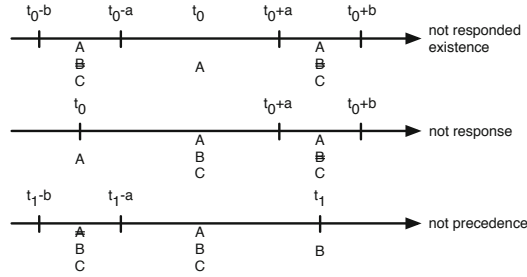


Fig. 7. Not Co-existence, Not Succession and Not Chain Succession

defined as the conjunction of the appropriate timed precedence and the timed response.

Figure 7 gives a graphical representation of the semantics for negations of constraints. The constraint *timed not responded existence* specifies that, whenever A occurs, B is forbidden in the specified interval before and after. Again, the *timed not co-existence* is not shown, but remains the conjunction of the timed not responded existence with parameters (A, B) and the timed not responded existence with parameters (B, A) . The *timed not response* indicates that, whenever A occurs, B is forbidden in the time interval $t \in I$. The *timed not precedence* indicates that, whenever B occurs, A is forbidden in the time interval $t \in [t_B - b, t_B - a]$. The *timed not succession* is the conjunction of these last two constraints. The chain versions of the precedence and response constraints (not shown) allow B/A to occur inside the interval if any action occurs between them.

In Table 1, we summarize the timed semantics for each constraint. The semantics for the untimed constraints is the same as in [9], except we allow for the use of past operators, which makes specifying some constraints simpler. The timed versions are in most cases the same as for the untimed version except we add time. The (negated) responded existence is a bit more complicated to make the timing correct, but it is easy to see that this formula is equivalent to the corresponding untimed formula if time is removed.

3.1 Timed Automata for Declare

MTL is undecidable. It is not always possible to translate an MTL formula to a timed automaton. We show this with a counterexample. Consider, for instance, the timed semantics for the response constraint $\mathbf{G}(A \rightarrow \mathbf{F}_I B)$. We cannot express this semantics with a fixed number of clocks. Intuitively, we need to start a new timer for each A to make sure we that can see which (if any) of the A s are satisfied by a given B . We have chosen to use MTL to specify Timed Declare anyway as we need the power to express the full semantics of Timed Declare. Other timed logics similar to LTL cannot express [4] the semantics for, e.g., timed responded existence and response.

Table 1. Semantics for some Declare constraints

Constraint	Untimed semantics	Timed semantics
existence	$\mathbf{F}A$	$\mathbf{F}_I A$
absence	$\neg \mathbf{F}A$	$\neg \mathbf{F}_I A$
exclusive allowance	–	$\neg \mathbf{F}_{[0,a]} A \wedge \neg \mathbf{F}_{[b,\infty]} A$
responded existence	$\mathbf{F}A \rightarrow \mathbf{F}B$	$\mathbf{G}(A \rightarrow (\mathbf{O}_I B \vee \mathbf{F}_I B))$
response	$\mathbf{G}(A \rightarrow \mathbf{F}B)$	$\mathbf{G}(A \rightarrow \mathbf{F}_I B)$
alternate response	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U} B))$	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U}_I B))$
chain response	$\mathbf{G}(A \rightarrow \mathbf{X}B)$	$\mathbf{G}(A \rightarrow \mathbf{X}_I B)$
precedence	$\mathbf{G}(B \rightarrow \mathbf{O}A)$	$\mathbf{G}(B \rightarrow \mathbf{O}_I A)$
alternate precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B \mathbf{S} A))$	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B \mathbf{S}_I A))$
chain precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}A)$	$\mathbf{G}(B \rightarrow \mathbf{Y}_I A)$
not responded existence	$\mathbf{F}A \rightarrow \neg \mathbf{F}B$	$\mathbf{G}(A \rightarrow (\neg \mathbf{O}_I B \wedge \mathbf{F}_I B))$
not response	$\mathbf{G}(A \rightarrow \neg(\mathbf{F}B))$	$\mathbf{G}(A \rightarrow \neg(\mathbf{F}_I B))$
not precedence	$\mathbf{G}(B \rightarrow \neg(\mathbf{O}A))$	$\mathbf{G}(B \rightarrow \neg(\mathbf{O}_I A))$
not chain response	$\mathbf{G}(A \rightarrow \neg(\mathbf{X}B))$	$\mathbf{G}(A \rightarrow \neg(\mathbf{X}_I B))$
not chain precedence	$\mathbf{G}(B \rightarrow \neg(\mathbf{Y}A))$	$\mathbf{G}(B \rightarrow \neg(\mathbf{Y}_I A))$

The uncomputability is only present if we wish to translate a formula to an automaton for static analysis. For on-line monitoring, we can easily instantiate a timer every time we activate a constraint. We can do this in terms of automata or as in [8] in terms of the Event Calculus. The desire to provide a meaningful timed semantics for Declare, even though we lose some analytical power is what prompts us to go with the undecidable logic.

As not every MTL formula can be translated to an automaton and not every Timed Declare constraint can be represented using a timed automaton, we need to restrict what we allow if we are to do analysis. If we restrict all intervals to either include 0 or go to ∞ it turns out that all constraints can be represented as an automaton. The reason is that it now becomes enough to remember the first and last time we saw each event for each constraint. Therefore, for an event, A , we introduce two clocks xA and yA ; xA keeps track of the first outstanding A and yA keeps track of the last. Note that $\text{succession}_{[0,b]}(A, B) \wedge \text{succession}_{[a,\infty]}(A, B) \neq \text{succession}_{[a,b]}(A, B)$, so this does not contradict that we cannot construct the automaton for the right side of the expression. The left side of the expression can be satisfied using two A s or two B s, each in one interval but not in the other, but the right side does not admit this.

The automaton in Fig. 2 checks the $\text{succession}_{[0,a]}(A, B)$ constraint. From the initial state s_0 , we can take an A to s_1 and a B to s_3 . State s_3 is an inescapable non-accepting state and can be thought of as a failure (indicated by a dashed outline). This makes sense: if we see a B before and A , we have violated the constraint. For all other events, we just remain in s_0 . When we see an A from s_0 , we reset the clocks xA and yA , indicating when we first and last saw an A (namely now). We can stay in s_1 as long as $xA \leq a$, i.e., until it is long enough ago we saw an A that executing a B is mandatory; if we do not progress before that, we are forced to follow the τ transition to s_3 . Intuitively s_1 means “we have seen A s that are not followed by B s”, s_2 means “we have seen A s, all obligations are satisfied, and the last A is close enough that we may still execute B s”, and

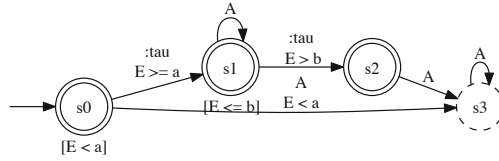


Fig. 8. Timed automaton for the *exclusive allowance*_[a,b](A) constraint

s0 means “we have no outstanding As and no A is close enough that we may execute Bs”. Thus, we reset when we last saw an A in s1 and transition to s2 if we see a B. We may only stay in s2 as long as the last A is close enough (the invariant on s2, $yA \leq a$); if we see an A we have a new outstanding A and move to s1 resetting both clocks. We allow for executing Bs, but when the invariant no longer is satisfied, we transition to s0 and start from scratch.

We employ a similar technique for all precedence, response, succession, and response constraints. The existence, absence, and exclusive allowance can trivially be checked in their full generality using a single clock for each constraint (see, e.g., Fig. 8 where the automaton for exclusive allowance is shown and uses only clock E). Precedence and response constraints can be checked using a single clock (we only need to keep track of either the first or last occurrence of A depending on whether the interval includes 0 or ∞ ; see Figs. 9 and 10 for examples). We can also represent the alternate response and the three chain constraints in their full generality. The intuition is that we can no longer execute more As between A and B (cf., Figs. 5 and 6).

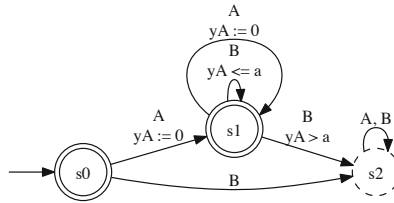


Fig. 9. Timed automaton for the *precedence*_[0,a](A, B) constraint

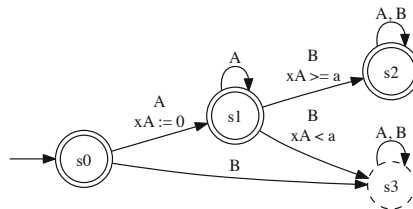


Fig. 10. Timed automaton for the *precedence*_{[a,∞)}(A, B) constraint

4 Analysis

In this section we show how to use automaton-based Timed Declare for analysis purposes to provide alerts when tasks are time-sensitive and to implement an a priori check of whether it is possible to meet deadlines.

4.1 Colored Alerts to Provide a Priori Advice

To monitor a Timed Declare model, we translate the model into a timed automaton. We simply instantiate our timed automata for each constraint. It is possible to compute the product of timed automata efficiently [2] so we do that in a way similar to how we construct colored automata for untimed Declare models. Such automata contains information about acceptance for each constraint in isolation and also about the acceptance of the conjunction. In the untimed version, this allows us to discover an inevitable violation even though no violation has yet taken place. The goal is to extend this to temporal properties as well.

Using a timed automaton we allow users to have relevant feedback during the process execution. During the execution of the process, this automaton can be used to give advice to the users about the action to undertake to obey the latencies and the deadlines specified by the compliance model, thus preventing possible violations from taking place. This advice is given through alerts that can be associated to different colors: *yellow* for alerts with low severity, *orange* for alerts with medium severity and *red* for alerts with high severity.

A *red alert* is generated when the automaton is in a consistent state, but letting time pass will unavoidable lead to violating one of the constraints. For instance, in the automaton in Fig. 2, modeling the succession constraint (in our example think of the succession from *Order* to *Pay*), if A (*Pay*) is executed (moving to state $s1$), a red alert is generated for the execution of B (*Delivery*) within 12 time units (as $a=12$). If *Delivery* is not executed, the constraint is violated (delivery has not taken place on time). Such an alert is generated for a state with an invariant where the only τ transition leads to a failure (dashed) state.

An *orange alert* is generated when the automaton is in a state where an activity can currently be executed but, after a certain number of time units, it cannot be executed anymore (and never in the future). For instance, in the automaton in Fig. 8 modeling the exclusive allowance constrains (in our model think of the *Discount* which is only available for the first 48 time units), when monitoring starts, we immediately transition to $s1$ (as $a=0$). Then, an orange alert is generated for A (*Discount*, because the customer missed out on the limited-time discount). The alert is generated because in state $s1$ it is possible to execute A , but $s1$ has an invariant and a τ transition to $s2$ from which executing A will always lead to a failure state. Alternatively, we generate such an alert if we are in a state where an action is guarded and can no longer be executed in any successor state and the clocks in the constraints cannot be reset.

A *yellow alert* is generated when the automaton is in a state where an activity can currently be executed but, after a certain number of time units, it cannot be

executed anymore. However, there is still the possibility to execute the activity somewhere in the future. For instance, in the automaton in Fig. 9 modeling the precedence constraint (in our example, we have a precedence of this type from Order to Pay). If A (Order) is executed, a yellow alert is generated to execute B (Pay) within 12 time units (as $a=12$). This is because in state $s1$ we can execute Pay but if we let time pass, we can no longer satisfy the guard and there is no τ transition to a state where we can. We also generate such an alert if we are in a state with an invariant and a τ transition to a state where the event is not enabled.

4.2 Constraint Interaction

Constraint interaction can be checked by computing the strongly connected components (SCCs) of the automaton, marking failure states (dashed states in the examples), and for each state compute which events lead to non-failure states, called the *enabled events*. We compute the union of these sets for each strongly connected component, and propagate them backwards in the SCC graph. We call these the *possible events*. Now, as we traverse the automaton, we can identify each of the three kinds of alerts and notice that after payment, delivering is very important (red alert) to avoid violating a constraint, applying for the discount is of medium importance (orange alert) as it becomes unavailable after some time, and after ordering, payment is important to avoid erasure of the order.

When we just start the process, we may not realize that we are on the clock. In our example, we actually have to hurry with our order, because if we do not order within 47 time units, we cannot wait one time unit more and apply for the discount before it becomes unavailable at time 48. We thus wish for an orange alert for Order in the initial state even though Order itself does not become permanently unavailable. We cannot see this from automata from individual constraints, but the product automaton is needed. In Fig. 11, we see the product of the automata for $precedence_{[1,\infty)}(\text{Order}, \text{Discount})$ and $exclusive\ allowance_{[0,48]}(\text{Discount})$. We have hidden the failure state for legibility; anytime Order or Discount is not explicitly possible, they lead to the failure

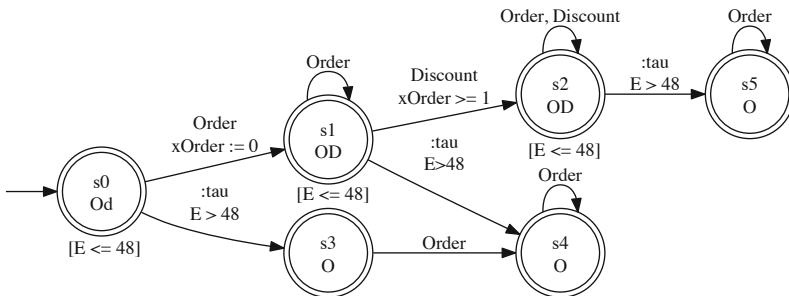


Fig. 11. Timed automaton for the conjunction of the $exclusive\ allowance_{[0,48]}(\text{Order})$ and $precedence_{[1,\infty)}(\text{Order}, \text{Discount})$ constraints

state. We have also annotated the states with the enabled and possible events. An O means that **Order** is enabled, a D means that **Discount** is enabled, and d means that **Discount** is possible. We see that in the initial state, **Discount** is possible, but the state has an invariant and a τ transition to a state where **Discount** no longer is possible. We can see that while **Discount** is possible, it is not enabled, so producing an orange alert would be of little use. Instead, we produce an orange alert for **Order** as we can see it leads us to another strongly connected component where **Discount** still is possible and even enabled in this case. The orange alert now moves to **Discount** where it rightly belongs.

4.3 Detection of Inconsistencies

If we modify the model in Fig. 1, adding a *not succession* constraint from **Order** to **Deliver** with a time limit of 36 time units, we model that we may not deliver goods within 36 time units from the order (e.g., due to local tax laws). We can see that **Pay** has to be executed no more than 12 time units from **Order** and **Deliver** no more than 12 time units from **Pay**, forcing delivery within 24 time units of **Order**. This of course conflicts with the new constraint, and we get an automaton accepting the empty language. We can detect this and point out the conflict between the 3 constraints, and let the user alleviate it by removing one or loosening one of the temporal constraints.

5 Implementation

In this section, we briefly describe our prototype implementation of Timed Declare in UPPAAL [12], a tool for analysis of timed automata.

UPPAAL makes it possible to design a model by defining process templates. We have designed a process template for each of the (most commonly used) constraints in Timed Declare. One such an example is shown in Fig. 12. In this example, we have a template in the field to the left for each constraint. The model shown is the UPPAAL implementation of the automaton from Fig. 2 testing the $succession_{[0,a]}(A, B)$ constraint. UPPAAL implements automata slightly differently from what we want. Most importantly, it does not have a notion of accepting states nor of synchronization. This is possible to get around, though.

To get around lack of synchronization, we use *broadcast channels*, which make it possible for a single sender to synchronize with multiple recipients. We then have a single driver (see Fig. 13) acting as sender and all the templates act as receivers and hence progress as the driver dictates. The driver (Fig. 13 (left)) has two states, an initial state and a termination state. The driver is a real flower-model, which allows for any (here of four) actions looping in the initial state. Each time, we transmit on the corresponding broadcast channel (e.g., a!). At some point, the driver decides to terminate, and communicates on **done**. If we look back at the implementation of Fig. 2 in Fig. 12, we see that many events receive on channels (e.g., A?). This will be synchronized with other automata listening to the same channel. Furthermore, channels cannot advance without receiving, so they just stay put for events that do not affect them.

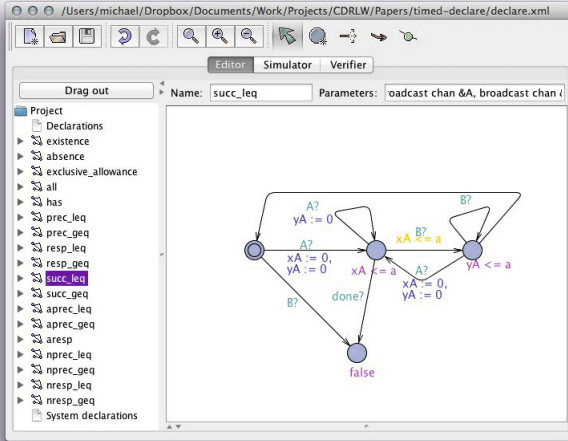


Fig. 12. Our prototype in UPPAAL

We handle non-accepting states by indicating that they forbid communicating on done. In our example, we see this construction from the middle state in Fig. 12. It has a transition receiving on done but leading to a state with the invariant false underneath it. This means that it is never a legal move to go there, so if the automaton in Fig. 12 is in the middle state, the driver is prevented from sending on done and hence terminating.

Listing 1. System declarations for the model in Fig. 1

```

1 broadcast chan done, Order, Pay, Delivery, Discount, Advertisement;
3 c1 = prec_leq(Order, Pay, 12);
4 c2 = succ_leq(Pay, Delivery, 12);
5 c3 = prec_geq(Order, Discount, 1);
6 c4 = exclusive_absence(Discount, 0, 48);
7 c5 = response_geq(Discount, Advertisement, 48);
8 a = all(Order, Pay, Delivery, Discount, Advertisement);
10 system a, c1, c2, c3, c4, c5;

```

Finally, we need to tie our symbolic names (A, B, and a) in Fig. 12 to actual tasks. We notice near the top right of Fig. 12 that we have a field for parameters. We here state that A and B are broadcast channel input parameters, and we have declared a as an integer input parameter. We then tie the entire system together in the System declarations, (the description is as simple as Listing 1). We first set up a channel for each task (l. 1), then we instantiate the individual constraints (ll. 3–7) and the driver (l. 8), and finally we start the system (l. 10). Now the formal names are tied to actual names and we can run the model in the simulator



Fig. 13. Our driver (left) and test (right)

in UPPAAL. We can also perform analysis. We can check if the model is non-empty by checking if the driver (Fig. 13 (left)) can reach the finished state. We can also instantiate the test in Fig. 13 (right) for each task and, when the found state is reachable, indicating that it is possible to execute that event.

6 Conclusion

In this paper, we have introduced a timed version of Declare. Our version is similar to the one in [8], but allows the use of time for more Declare constraints. We give a semantics in terms of MTL, a timed version of LTL, and we represent these semantics through timed automata. We show how we can use these automata to not only identify when a constraint is violated like in [8], but even to provide a priori warnings that time constraints may be violated in the future or that certain actions may become unavailable if not executed swiftly. We can also detect that deadlines are impossible to meet prior to execution.

In this paper, we have considered tasks without duration taking place with time spans between them. We are very interested in looking into giving tasks duration. This can be done either by considering the start and completion of a task as separate events or by looking at tasks as signals instead of events. When we do so, it is obvious to start looking at the resource perspective as well, as it may be that a model cannot be executed by a single person (for example if two 14 time unit tasks have to be executed within a 24 time unit period). For these cases we can compute interesting statistics like how fast can a model be executed given infinite resources, how fast can it be executed (if at all) using a given amount resources, and how many resources are necessary to execute a model. We believe that this can be extended to also provide plans for individual resources, and we believe we can extend this to do planning for running multiple instances of multiple models. This is very similar to providing operational support (except where operational support tries to answer similar questions on-the-fly, we try to answer them before the fact).

Here, we have used timed automata because they make it possible and easy to do sophisticated analysis. It would also be very interesting to investigate how moving to more advanced automata admitting creating fresh clocks skews the balance between expressiveness and analysis.

We would also like to integrate the presented analysis facilities in Declare [13], preferably in a backwards compatible way. One way to do that is to integrate UPPAAL's command line tool, which may definitely be good for analysis, but less optimal for on-the-fly execution, as UPPAAL computes the product on-the-fly while checking properties. We can, therefore, not precompute the enabled and possible events, which is necessary to be able to provide orange and yellow alerts. Another possibility is to use UPPAAL's DBM library, which implements difference-bound matrices [6] (a very efficient data-structure to implement timed automata), and to leverage the automaton library already available in Declare.

References

1. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development* 23, 99–113 (2009)
2. Alur, R., Dill, D.: A Theory of Timed Automata. *Theoretical Computer Science* 126(2), 183–235 (1994)
3. Alur, R., Henzinger, T.: Real-time logics: complexity and expressiveness. In: *Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990*, pp. 390–401 (June 1990)
4. Bauer, A., Leucker, M., Schallhart, C.: Comparing ltl semantics for runtime verification. *Logic and Computation*, 651–674 (2010)
5. Chesani, F., Mello, P., Montali, M., Torroni, P.: Verification of Choreographies During Execution Using the Reactive Event Calculus. In: Bruni, R., Wolf, K. (eds.) *WS-FM 2008*. LNCS, vol. 5387, pp. 55–72. Springer, Heidelberg (2009)
6. David, D.: Timing Assumptions and Verification of Finite-state Concurrent Systems. In: Sifakis, J. (ed.) *CAV 1989*. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
7. Koymans, R.: Specifying real-time properties with metric temporal logic. *Real-Time Systems* 2, 255–299 (1990), <http://dx.doi.org/10.1007/BF01995674>, 10.1007/BF01995674
8. Montali, M.: Specification and Verification of Declarative Open Interaction Models. *LNBIP*, vol. 56, pp. 1–383. Springer, Heidelberg (2010)
9. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
10. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: *IEEE International EDOC Conference 2007*, pp. 287–300 (2007)
11. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. *Electron. Notes Theor. Comput. Sci.* 113, 145–162 (2005), <http://dx.doi.org/10.1016/j.entcs.2004.01.029>
12. UppAal webpage, <http://www.uppaal.org>
13. Westergaard, M., Maggi, F.: Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In: Ludwig, H., Reijers, H. (eds.) *Business Process Management Demonstration Track (BPM Demos 2011)*. *CEUR Workshop Proceedings*, vol. 820. CEUR-WS.org (2011)

Planlets: Automatically Recovering Dynamic Processes in YAWL

Andrea Marrella, Alessandro Russo, and Massimo Mecella

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Sapienza Università di Roma, Rome, Italy
{marrella, arusso, mecella}@dis.uniroma1.it

Abstract. Process Management Systems (PMSs) are currently more and more used as a supporting tool to coordinate the enactment of processes. YAWL, one of the best-known PMSs coming from academia, allows to define stable and well-understood processes and provides support for the handling of expected exceptions, which can be anticipated at design time. But in some real world scenarios, the environment may change in unexpected ways so as to prevent a process from being successfully carried out. In order to cope with these anomalous situations, a PMS should automatically recover the process at run-time, by considering the context of the specific case under execution. In this paper, we propose the approach of PLANLETS, self-contained YAWL specifications with recovery features, based on modeling of pre- and post-conditions of tasks and the use of planning techniques. We show the feasibility of the proposed approach by discussing its deployment on top of YAWL.

Keywords: Process Management Systems, YAWL, recovery, planning.

1 Introduction

In the last years, the increasing demand in solutions for *dynamic processes* and the need to provide support for flexible and adaptive process management has emerged as a leading research topic in the BPM domain [15,20] and has led to reconsider the trade-off between flexibility and support provided by existing Process Management Systems (PMSs). Research efforts in this field try to enhance the ability of processes and their support environments to modify their behavior in order to deal with contextual changes and exceptions that may occur in the operating environment during process enactment and execution. On the one hand, existing PMSs like YAWL [17] provide the support for the handling of expected exceptions. The process schemas are designed in order to cope with potential exceptions, i.e., for each kind of exception that is envisioned to occur, a specific contingency process (a.k.a. exception handler or compensation flow) is defined. On the other hand, adaptive PMSs like ADEPT2 [19] support the handling of unanticipated exceptions, by enabling different kinds of ad-hoc deviations from the pre-modeled process instance at run-time, according to the structural process change patterns defined in [18].

However, in a dynamic process the sequence of tasks heavily depends on the specifics of the context (e.g., which resources are available and what particular options exist at that time), and it is often unpredictable the way it unfolds. The use of processes for supporting the work in highly dynamic contexts like healthcare and emergency management has become a reality, thanks also to the growing use of mobile devices in everyday life, which offer a simple way for picking up and executing tasks. To deal with exceptions and uncertainty introduced by such contexts, the need for flexible and easy adaptable processes has been recognized as critical [10]. However, traditional approaches that try to anticipate how the work will happen by solving each problem at design time, as well as approaches that allow to manually change the process structure at run time, are often ineffective or not applicable in rapidly evolving contexts. The design-time specification of all possible compensation actions requires an extensive manual effort for the process designer, that has to anticipate all potential problems and ways to overcome them in advance, in an attempt to deal with the unpredictable nature of dynamic processes. Moreover, the designer often lacks the needed knowledge to model all the possible contingencies, or this knowledge can become obsolete as process instances are executed and evolve, by making useless his/her initial effort.

This paper, based on our previous work [11,12,1], introduces the notion of PLANLETS, as self-contained YAWL nets where tasks are annotated with pre-conditions, desired effects and post-conditions. The main characteristic of a PLANLET is in the ability to recover itself - if an exception arises - automatically, without explicitly defining any recovery policy at design-time. This feature is critical for processes executed in dynamic environments where requirements and context can change rapidly and unpredictably. An external planner is in charge of synthesizing the needed recovery procedure on-the-fly, by contextually selecting the compensation tasks from a specific repository linked to the PLANLET under execution.

The rest of the paper is organized as follows. Section 2 presents and discusses related works. Section 3 introduces our running example that helps to clarify the scope of the approach. Section 4 presents the general approach and shows how it can be concretely built on top of the YAWL architecture, whereas Section 5 discusses task annotations and the use of planning techniques for automatically recovering dynamic processes. Section 6 reports on experimental evaluation results and Section 7 concludes the paper by discussing limitations and future developments of the approach.

2 Related Works

Recently, techniques from the field of artificial intelligence (AI) have been applied to process management. In [5], the authors present a concept for dynamic and automated workflow re-planning that allows recovering from task failures. To handle the situation of a partially executed workflow, a multi-step procedure is proposed that includes the termination of failed activities, the sound suspension of the workflow, the generation of a new complete process definition and the

adequate process resumption. In [9], the authors take a much broader view of the problem of adaptive workflow systems, and show that there is a strong mapping between the requirements of such systems and the capabilities offered by AI techniques. In particular, the work describes how planning can be interleaved with process execution and plan refinement, and investigates plan patching and plan repair as means to enhance flexibility and responsiveness. A new life cycle for workflow management based on the continuous interplay between learning and planning is proposed in [3]. The approach is based on learning business activities as planning operators and feeding them to a planner that generates the process model. The main result is that it is possible to produce fully accurate process models even though the activities (i.e., the operators) may not be accurately described. The approach presented in [13] highlights the improvements that a legacy workflow application can gain by incorporating planning techniques into its day-to-day operation. The use of contingency planning to deal with uncertainty (instead of replanning) increases system flexibility, but it does suffer from a number of problems. Specifically, contingency planning is often highly time-consuming and does not guarantee a correct execution under all possible circumstances. Planning techniques are also used in [4] to define a self-healing approach for handling exceptions in service-based processes and repairing faulty activities with a model-based approach. During the process execution, when an exception occurs, a new repair plan is generated by taking into account constraints posed by the process structure and by applying or deleting actions taken from a given generic repair plan, defined manually at design time.

If compared with the above works, the PLANLET approach provides some interesting features in dealing with exceptions: (i) it modifies only those parts of the process that need to be changed/adapted by keeping other parts stable; (ii) it synthesizes the recovery procedure at run-time, without the need to define any recovery policy at design-time.

3 Running Example

As an application scenario, we consider an emergency management process defined for train derailments and inspired by a real process used by the main Italian Railway Company. The corresponding YAWL process, introduced in [12], is shown in Fig. 1.a. The process starts when the railway traffic control center receives an accident notification from the train driver and collects some information about the derailment, including the GPS location and the number of coaches and passengers. In Fig. 2.a a possible map of the area is depicted as a 4x4 grid of locations. For the sake of simplicity, we supposed that the train is composed by a locomotive (located in $loc(3,3)$) and two coaches (located in $loc(3,2)$ and $loc(3,1)$ respectively). Then, it may be required to cut off the power in the area and to interrupt the railway traffic near the derailment scene. In parallel, after having collected additional information about the train (e.g., security equipment) and emergency services available in the area, a response team can be sent to the derailment scene. Such a team is composed by four first responders (in the rest of the paper, we refer to them also as *actors*) and two robots,

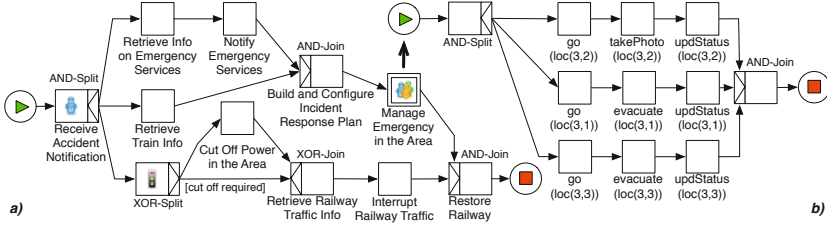


Fig. 1. The YAWL process defined for a train derailment scenario (a), in which the composite task “Manage Emergency in the Area” is a PLANLET (b)

initially located in $loc(0,0)$. We assume that the actors are equipped with mobile devices (for picking up and executing tasks) and provide specific skills. For example, actor $a1$ is able to take pictures and to extinguish fire, whereas $a2$ and $a3$ are in charge of evacuating people from train coaches. The connection between mobile devices is supported by a network provided by a fixed antenna (whose range is limited to the dotted squares in Fig. 2.a), and the robots $rb1$ and $rb2$ can act as wireless routers for extending the network range in the area. A robot provides a connection limited to the locations adjacent (in any direction) to its position. Each robot can move in the area, but it is constrained to be always connected to the main network. This is guaranteed if the intersection between the squares covered by the main network and the squares covered by the robot connection is not empty. A robot connected to the main network can act as a “bridge”, allowing the other robot to be connected through it to the main network. Robots have a battery that discharges a fixed quantity after each movement. The actor $a4$, in charge of checking the correct working of the antenna, can change the battery of a robot if empty. Collected information is used for defining and configuring at run-time an incident response plan, defined by a contextually and dynamically selected set of activities to be executed on the field by first responders. Such activities are abstracted into the composite task¹ “Manage Emergency in the Area” (cf. Fig. 1.b). The subnet is composed by three parallel branches with tasks that instruct first responders to act for evacuating people from train coaches, to take pictures and to assess the gravity of the accident. Despite the simple structure of the incident response plan, the high dynamism of the operating environment can lead to a wide range of exceptions. In general, for dynamic processes there is not a clear, anticipated correlation between a change in the context and corresponding process changes. Suppose, for example, that the task $go(loc(3,3))$ is assigned to actor $a1$ (cf. Fig. 1.b), which reaches instead the location $loc(0,3)$. This means that $a1$ is now located in a different position than the desired one, and s/he is out of the network range. Since all the actors/robots need to be continually inter-connected to execute the process, the PMS has to find a recovery procedure that first instructs the robots

¹ A composite task is a container for another YAWL sub-net, with its own set of elements.

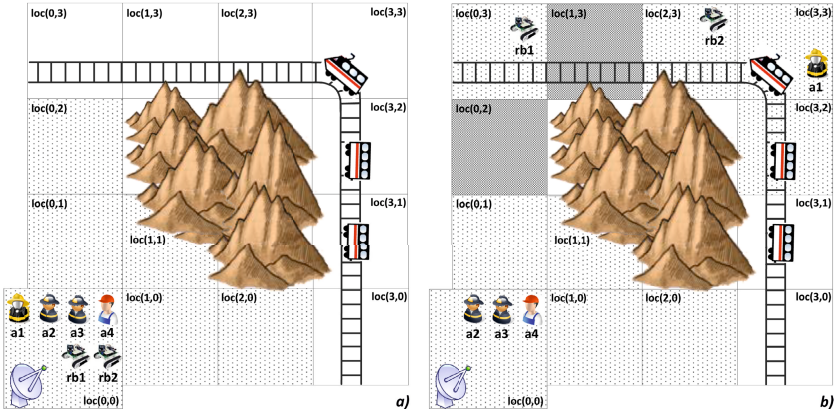


Fig. 2. Area (and context) of the intervention

to move in specific positions for maintaining the network connection, and then re-assign the task $go(loc(3, 3))$ to $a1$. It is unrealistic to assume that the process designer can pre-define all possible compensation activities for dealing with this exception (apparently simple), since the process may be different every time it runs and the recovery procedure strictly depends on the actual contextual information (the positions of operators/robots, the range of the main network, the battery level of each robot, etc.). For the same reason, it is also difficult to manually define an ad-hoc recovery procedure at run-time, as the correctness of the process execution is highly constrained by the values (or combination of values) of contextual data.

4 The General Approach and Architecture

4.1 Introducing Planlets

Most of current PMSs are not sufficiently able to deal with dynamic processes, as they do not automatically adapt process instance executions in order to align them to the changes to the environment. In this paper we propose a solution that builds on top of YAWL [17], and consists of annotating at design-time a YAWL specification with additional information which allows process instances to be automatically recovered. In particular, we assume the tasks of a YAWL process specification to be annotated with *pre-conditions*, *desired effects* and *post-conditions*. Failures arise either when associated pre-conditions for a task are not satisfied at the time the task is to be started, or when post-conditions do not hold after the execution of the task. Effects represent the changes that a successful task execution imposes on the *state* of the world, reflecting the current value of the contextual properties that constraint the process under execution. Hence, the process designer just states *what* conditions have to be satisfied, without having to anticipate *how* these can be fulfilled. In order to formalize the concept, we introduce the definition of PLANLET:

Definition 1 (Planlet). Let YN be a YAWL net, T be the tasks defined in YN , and V be the set of variables defined in YN . Let $Expr(V)$ be the set of expressions over the variables in V . A PLANLET is a tuple $(YN, Pre, Post, Eff)$ where (i) $Pre : T \rightarrow Expr(V)$ returns an expression representing the pre-conditions of tasks in T ; (ii) $Post : T \rightarrow Expr(V)$ returns an expression representing the post-conditions of tasks in T ; (iii) $Eff : T \rightarrow Expr(V)$ returns an expression representing the effects of tasks T .

The role of pre/post-conditions and effects for a YAWL task is twofold: (i) pre-conditions and post-conditions enable run-time process execution monitoring and exception detection: they are checked respectively before and after task executions, and the violation of a pre-condition or post-condition results in an exception to be handled; (ii) along with the input/output parameters consumed/produced by the task, pre-conditions and effects provide a complete specification of the task: this allows the task to be represented as an action in a planning domain description and used for solving a planning problem built to handle an exception.

At design-time, the annotated tasks are stored in a repository linked to the PLANLET specification, which may contain also other annotated tasks deriving from previous executions on the same contextual domain. At run-time, while instances of the YAWL specification are carried on, tasks become enabled. Every time a task $t \in T$ becomes enabled, expression $Pre(t)$ is evaluated; similarly, upon the completion of t , expression $Post(t)$ is evaluated. If an evaluation returns false upon enablement or completion of a task, the system is in an *invalid state* and, hence, the YAWL specification instance needs to be adapted to come back into the “right track”. In order to do that, the case execution is suspended, and a recovery procedure is automatically synthesized. To provide more details, let us assume that the current PLANLET is $\delta_0 = (\delta_1; \delta_2)$ in which δ_1 is the part of the PLANLET already executed and δ_2 is the part of the PLANLET which remains to be executed when an exception is identified. The adapted PLANLET is $\delta'_0 = (\delta_1; \delta_h; \delta_2)$. However, whenever a PLANLET needs to be adapted, every running task is interrupted, since the “repair” sequence of tasks $\delta_h = [t_1, \dots, t_n]$ is placed before them. Thus, active branches can only resume their execution after the repair sequence has been executed. This last requirement is fundamental to avoid the risk of introducing data inconsistencies during a repair.

The automatic synthesis of the recovery procedure δ_h is enacted on-the-fly by an external planner. A planner solves the problem to find a sequence of actions that move a system state from the initial one to a target goal, using a predefined set of admissible actions. Each action is associated the set of pre-conditions $Pre(t)$ in order for that step to be chosen, as well as the effects $Eff(t)$ obtained as result of the action’s execution. Along with defining the set of admissible actions, it is also crucial to define how the state is represented, since pre-conditions and effects of actions are given in term of the chosen state representation. The actions’ set and the state definition are often referred to as *planning domain*. The standard representation language of planners to define actions and state is the Planning Domain Definition Language (PDDL) [2]. In the context of adaptation

of instances of YAWL specifications, each task specification is associated with a different action in the planning domain; the task’s pre-conditions and effects are translated in PDDL and associated to the corresponding action. In addition to the so-created planning domain, when an exception arises, the invalid state and the pre-condition (or post-condition) violated is given in input to a planner, which can try to build a plan. If the plan exists, the planner is eventually going to return it. In this case, the plan is converted into a sequence of YAWL tasks which are assigned to qualifying participants. When the converted plan is carried out, the original suspended process is restored for execution.

Let us consider the example introduced in Section 3. The composite activity “Manage Emergency in the Area” may be modeled as a self-contained PLANLET specification (cf. Fig. 1.b), linked to a repository containing a set of emergency management (annotated) tasks, that range from the simple activity of taking pictures to the more complex extinguishment of a fire. An explicit representation of contextual information (the connection of each actor to the network, the map of the area, the battery charge level of each robot etc.) is needed for preserving the correct PLANLET execution. The same exception shown in Section 3 (the actor a1 is not more connected to the network and s/he is in a position different than the desired one) results in a post-condition failure, and now may be easily caught and solved. The planner builds a planning problem by taking as initial state the invalid state of the PLANLET, and as goal a state where all actors/robots are inter-connected to the network and a1 is in the desired location $loc(3,3)$. The recovery plan is automatically synthesized by contextually selecting tasks from the repository linked to the PLANLET. Suppose, for example, that the two robots $rb1$ and $rb2$ have an empty battery. In such a case, the planner devises on-the-fly a possible solution, composed by a sequence of 5 tasks² [$chargeBattery(a4,rb1)$, $move(rb1,loc(1,3))$, $go(a1,loc(3,3))$, $chargeBattery(a4,rb2)$, $move(rb2,loc(3,3))$] that change the state of the world as shown in Fig. 2.b.

4.2 Incorporating Planlets into YAWL

The architectural extension and integration we designed takes advantage of YAWL’s exception detection capabilities and leverages the flexibility of the exlet-based handling techniques.

Exception Handling in YAWL. The exception handling capabilities provided by YAWL³ build on the conceptual framework presented in [14]. In order to understand how exceptions are detected and handled in YAWL we refer to the architecture in Fig. 3 (for now, do not consider the *Planning Service* and the *PLANLET Repositories*⁴). For each exception that can be anticipated, it is

² The recovery plan is synthesized by taking care of the skills of process participants, and their availability for task assignment and execution. Hence, each task composing the plan is already associated to the participant that will execute it.

³ In this paper we refer to the final release of YAWL 2.1.

⁴ With the exclusion of the *Planning Service* and of the *PLANLET Repositories*, the picture refers to the architecture defined in [17].

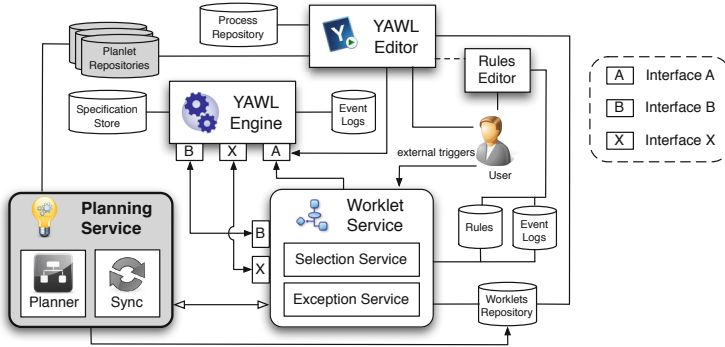


Fig. 3. The YAWL architecture extended with the *Planning Service*

possible to define an exception handling process, named *exlet*, which includes a number of exception handling primitives (for removing, suspending, continuing, etc. a work item/case) and one or more compensatory processes in the form of *worklets* (i.e., self-contained YAWL specifications executed as compensatory processes [17]). Exlets are linked to specifications by defining *rules* (through the *Rules Editor* graphical tool), in the shape of Ripple Down Rules specified as *if condition then conclusion*, where the *condition* defines the exception triggering condition and the *conclusion* defines the exlet. At run-time, exceptions are detected and managed by the *Exception Service* [17]. The service determines whether an exception has occurred and, if so, it executes the corresponding exlet. If the exlet includes a compensation worklet, the service retrieves it from the repository, loads it into the engine and executes it as a new separate case, possibly in parallel with the parent case if it was not suspended by the exlet.

Enabling Planlets in YAWL. From an architectural perspective, as shown in Fig. 3, planning capabilities are provided by a *Planning Service* that implements the planning logic and algorithm. In order to define the role of the *Planning Service* and clarify how it interacts with existing YAWL architectural components and services, we follow the process and exception handling life-cycle, from process design, enactment and monitoring to exception detection, handling and (possibly) resolution. At design time, the process designer builds one or more PLANLET *Repositories* (or modifies the existing ones), by inserting/deleting annotated tasks and by (possibly) modifying the contextual domain linked to each repository. Tasks involved in a PLANLET specification are selected from a specific PLANLET repository, since they are thought to be enacted in a specific contextual domain. Before executing a PLANLET, the process designer instantiates the initial values for the properties of the contextual domain. As shown in Section 5, tasks pre- and post-conditions are automatically translated in YAWL pre- and post-constraints. In order to delegate the exception handling to the *Planning Service*, we introduce the possibility of mapping a *compensation activity* to the *Planning Service*. By defining this mapping instead of explicitly selecting a compensation worklet, the process designer configures the *Exception Service* so that

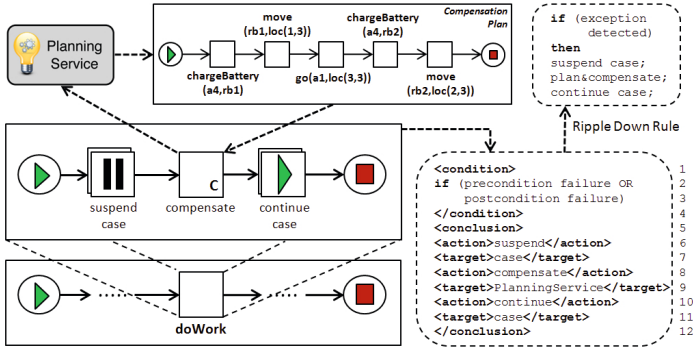


Fig. 4. *Planning Service* activation hierarchy for exception handling

the generation of the compensation worklet is delegated to the *Planning Service*. Fig. 4 shows an excerpt of the rule file defined for detecting and handling a workitem-level pre-execution (or post-execution) constraint violation. Lines 1-4 define the exception triggering condition (a pre- or a post-condition failure), while lines 5-12 define the exception handling exlet (which consists of suspending the current case, performing some compensation activities and then resuming the suspended case). In our extended version, the mapping of a compensation task to the *Planning Service* is identified by a `<target>` element containing the `PlanningService` value (line 9), in order to enact planning capabilities. If we consider our running example, the compensation plan devised in Fig. 4 corresponds to the one needed for re-establishing the network connection between actors/robots and for instructing actor *a1* to move in the desired location.

Planning Service Activation. When the *Exception Service* activates the *Planning Service*, it provides as input all case data associated with the running case, along with the detected violation over pre- and post-conditions. Based on this information, and on the specifications of available tasks, stored in the repository linked to the PLANLET under execution, the *Synchronization* component of the *Planning Service* is able to build the planning domain and to define a planning problem, and submit them to the *Planner* module in charge of synthesizing a recovery plan. If the *Planner* is able to successfully synthesize a compensation plan, it stores it as an executable specification (i.e., a worklet) in the *Worklets Repository* and notifies the *Exception Service*. The *Exception Service* is then able to enact the execution of the compensation worklet as if it was manually selected at design time, by loading the specification into the engine and launching it as a separate case. When the execution completes, output data produced by the worklet are mapped back to the parent case and subsequent actions in the exlet are executed. Following the exlet defined in Fig. 4, as the compensation worklet synthesized by the *Planner* is supposed to recover from the constraint violation, the suspended case can then be resumed and executed. If no valid plan can be found by the *Planner*, a notification alert is sent to an administrator, who is charge of handling the unsolved exception, e.g., manually building a compensation process or just canceling the process case.

5 Annotating YAWL Specifications in Planlets

A main step of our approach in YAWL consists of enriching the process model with a specification of process tasks, in terms of pre-conditions, desired effects and post-conditions, and with an explicit representation of the contextual domain needed for the correct process enactment.

In YAWL, each atomic task t can be linked to a decomposition. Decompositions can have a number of input and output *parameters*, each identified by a *name* and characterized by a *type* dictating valid values it may store, and define the so-called YAWL Service that will be responsible for task execution. As process data are represented through net-level variables, inbound and outbound mappings define how data is transferred from net variables to task variables and vice-versa [17]. We propose to extend task specifications at the decomposition level, with the possibility of defining pre-conditions, post-conditions and effects as logical formulae and expressions over task parameters.

Defining and Representing Finite Domain Types. The definition of a PLANLET requires the specification of the data types that characterize the information manipulated by process instances and define the domains over which predicates and functions are interpreted. In order to have a compact and finite representation of a process state, given by the values assumed by process variables at a given point in the execution, all data types must correspond to *finite* domains over which variables of that type can range; this requirement is imposed by the planning-based approach we propose. Examples of such domains are finite integer intervals or sets of strings, and other enumerated domains. As YAWL applies strong data typing and all data types are defined using XML Schemas, this can be easily achieved by defining data types as XML Schemas and using restrictions (e.g., via the `enumeration` constraint) to limit the content of an XML element to a set of acceptable values. In our example, we need to define data types for representing actors, robots⁵ and locations in the area (e.g., data type $Loc = \{loc00, loc10, \dots, loc33\}$), whose possible values are constant symbols that univocally identify objects in the domain of interest.

Defining and Representing Predicates and Functions. Predicates can be used to express properties of domain objects and relations over objects. A predicate consists of a predicate *symbol* P and a set of *typed parameters* or *arguments*⁶. Argument types (taken from the set of data types previously defined) represent the finite domains over which predicates are interpreted. In our example, we may need predicates for expressing the presence of a fire in a location or whether a location is covered by the network signal provided by the main an-

⁵ Although emergency operators and robots can be considered as *resources* or *services* able to execute tasks and can be represented in the organizational model provided by YAWL, we also need to explicitly represent them in the process because we need to define predicates and functions over these domains.

⁶ Predicates with no arguments, i.e., with arity 0, are allowed and can be considered as propositions; they are directly represented as boolean variables.

tenna, or relations, such as the adjacency between locations, i.e., $Fire(loc : Loc)$, $Covered(loc : Loc)$, $Adjacent(loc1 : Loc, loc2 : Loc)$.

In addition to basic predicates, we allow the designer to define *derived* predicates. They are declared as basic predicates, with the additional specification of a well-formed formula φ that determines the truth value for the predicate. In our domain, we may need to express that an actor is connected to the network if s/he is in a covered location or if s/he is in a location adjacent to a location where a robot is located (and is thus connected through the robot); assuming we have defined the data types $Robot = \{rb1, rb2\}$ and $Actor = \{a1, a2, a3, a4\}$, we have: $Connected(act : Actor) \{ \text{EXISTS}(l1 : Loc, l2 : Loc, rbt : Robot) ((at(act) = l1) \text{ AND } (Covered(l1) \text{ OR } (atRobot(rbt) = l2 \text{ AND } Adjacent(l1, l2)))) \}$

Numeric and object functions allow to represent and handle numeric values and domain objects as functions of other objects. Function declarations consist of a function *symbol* f , a set of *typed parameters*⁷, and a *return type*. Numeric functions have as return type an integer or a real number, whereas object functions have a return type taken from the set of data types defined in the net specification. The arguments of functions range over *finite* domains, and for object functions the same requirement holds for result types. In our example, we need to keep track of the battery level of the robots. This can be represented through the numeric function $batteryLevel(robot : Robot) : Integer$.

Similarly, we can represent the position of actors and robots by defining the following functional predicates that map actors and robots to their location: $at(actor : Actor) : Loc$ and $atRobot(robot : Robot) : Loc$.

State Variables Representation. The use of predicates and functions requires that at run-time we represent the corresponding logical interpretations, as state variables that hold (a) the truth value of the defined predicates over domain objects, and (b) the values of the defined functions with respect to different argument assignments. The interpretations are used to evaluate pre- and post-conditions, and are modified as a result of task executions. As a consequence of the declaration of a predicate or function, two new data types are automatically generated and added to the XML data types definitions for the net:

- T1. a complex data type that is able to represent the name of the predicate or function and
 - for predicates, all argument assignments for which the predicate holds⁸ (i.e., the current interpretation P^I for the predicate);
 - for functions, all argument assignments for which the function is defined, along with the corresponding value⁹ (i.e., the current interpretation f^I for the function);

⁷ Numeric functions with no arguments are allowed, and can be considered as state variables rather than constants, as their value may change during process executions; they are represented as integer or float/double variables.

⁸ Basically, a set containing all object tuples for which the predicate is true.

⁹ Basically, a map where object tuples are mapped to objects.

T2. a complex data type that is able to represent a predicate or function instance, in terms of the name of the predicate or function, the set of arguments and their assignment, and the truth value or numeric/object value of the predicate or function with respect to the specific assignment; different parameters of this type can be defined for process tasks, to be used for representing the effects that they can have on the predicate or function interpretation.

For each predicate and function, a single net-level state variable of type T1 is defined and it can be initialized so as to contain all values for the objects for which the predicate is true or the function is defined in the initial state. Derived predicates are not explicitly represented through net-level state variables, as their interpretation can be always derived from the corresponding formula, and they can not appear in task effects (but task effects can act on the basic predicates and functions that appear in the formula, thus indirectly modifying the truth value for the derived predicate).

Initial Interpretation for a Process Instance. It is given by an assignment of values to the state variables that represent truth values for predicates (initial facts) and initialization values for functions.

Pre-conditions, Post-conditions and Effects. They are defined at design time as logical annotations associated with tasks in a PLANLET. We assume a first-order predicate logic with numeric and object functions, with the restriction that free variables are not allowed and thus all variable symbols must be task parameter names or occur in the scope of a quantifier. The language is clearly inspired from PDDL, although we prefer an infix notation for the operators. Task pre/post-conditions and effects are represented in task specifications via the `<precondition>`, `<effect>` and `<postcondition>` markup elements. In our example, consider the task labeled as `go`, which requires that an actor moves from a location to another in the area. It defines two input parameters *from* and *to* of type *Loc*, representing the starting and arrival locations, and an input parameter *actor* of type *Actor* representing the emergency operator that executes the task. An instance of this task can be executed only if s/he is currently at the starting location and is connected to the network. As an effect of task execution, the actor moves from the starting to the arrival location, but we need, as post-condition, to verify whether the arrival location has been reached and the actor is still connected to the network. We can thus define the following annotations:

```
<precondition>at(actor) == from AND Connected(actor)</precondition>
<effect>at(actor) = to</effect>
<postcondition>at(actor) == to AND Connected(actor)</postcondition>
```

The designer can distinguish between: (i) *direct effects*, i.e., effects that always take place after an execution, and therefore the corresponding changes on the state variables are automatically performed when the task completes (e.g., if an effect of the form $BatteryLevel(robot) += 5$ is marked as automatic, after task execution the value for $BatteryLevel(robot)$ is directly increased by 5); and (ii) *supposed effects*, i.e., effects that define changes that are assumed to be performed only when the task is considered as an action in a planning domain.

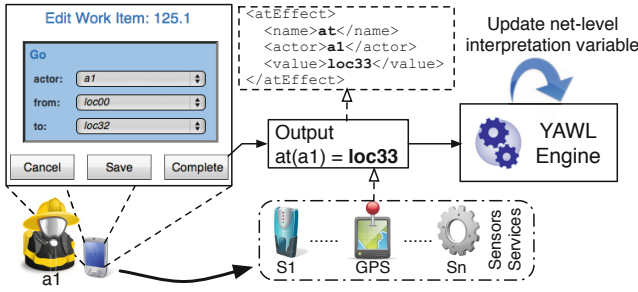


Fig. 5. A task effect represented as a variable assignment

Supposed effects can be interpreted as the effects that a task is supposed to have, but the actual produced changes are defined at run-time as a result of the concrete execution, such as the actual truth value of a predicate or the actual value for a direct assignment. In our example, $at(actor) = to$ is a supposed effect, as the actual value for $at(actor)$ is produced as a task output and may be different from the desired one (i.e., the value of the to variable prescribed in the effect). If the designer needs to verify if a task execution has produced the intended effect, s/he has to define a corresponding post-condition (i.e., the $at(actor) == to$).

Direct effects can be directly represented by generating an outbound mapping with an XQuery expression that adds/removes a tuple to/from the state variable representing predicate's interpretation (for positive/negative predicates), or updates the value for a tuple in the state variable representing function's interpretation (for assignment effects). In supposed effects, the actual values are produced by workitem executions, and all predicates and functions that appear in the effect expression have to be represented as task variables, so as to allow to specify (according to task's execution logic) the truth value for predicates or the value for functions. To this end, we represent each predicate and function that appears in the supposed effects as task parameters of type T2, where the predicate/function name is given and fixed, the values for the argument variables (i.e., the grounding) are defined by the inbound mappings for task parameters and the predicate/function actual value will be defined as a result of task execution. For these variables, outbound mappings are then generated, including XQuery expressions to update net-level state variables as for direct effects. Fig. 5 shows an example of how a variable can be used to represent an effect and how the actual value for $at(a1)$ can be produced as output; we show that the output value for $at(a1)$ is produced by a sensor (i.e., a GPS device) supporting the worklist handler. The produced value, in the example 'loc33', is then used to update the net variable representing the at interpretation to reflect that $at(a1) \mapsto loc33$.

State Model and Exceptions. In a PLANLET, a process state S is given by the token marking m_S (as defined in [17] for YAWL nets) and the logical

interpretation \mathcal{I}_S that assigns truth values to predicates and values to functions. The initial state over which a process instance is executed is given by the initial marking and an assignment of values to the state variables that represent the initial interpretation for predicates and functions. When a task t becomes enabled in a state S (as determined by m_S), its execution can start only if the task precondition formula φ_{pre} is true in \mathcal{I}_S , i.e., $\mathcal{I}_S \models \varphi_{pre}$. A task execution changes the interpretation according to actual task effects (which for a successful execution are given by the corresponding effects expression $expr_{eff}$) and leads to a new state S' where $m_{S'}$ is the produced marking and $\mathcal{I}_{S'}$ is the new interpretation. A completed task is considered as successfully executed if its postcondition formula φ_{post} is true in $\mathcal{I}_{S'}$, i.e., $\mathcal{I}_{S'} \models \varphi_{post}$. At run time all task executions are thus preceded and followed by the verification of whether $\mathcal{I} \models \varphi_{pre}$ and $\mathcal{I} \models \varphi_{post}$ ¹⁰. In this model, an exception occurs in a given state with an interpretation \mathcal{I} if a task is enabled but $\mathcal{I} \not\models \varphi_{pre}$ or if a task has completed but $\mathcal{I} \not\models \varphi_{post}$.

From Pre-/Post-conditions to Pre-/Post-execution Constraints. As part of its exception handling mechanism, YAWL supports the definition of workitem-level pre- and post-execution constraints, as rules with conditions that (i) are checked when the workitem becomes enabled and when it is completed, and (ii) if violated, they trigger an exception and the execution of an exception handling process (i.e., a YAWL exlet [14]). Conditions are defined over case variables as strings of operands and arithmetic, comparison and logical operators; conditional expressions may also take the form of boolean XQuery expressions [17]. In our approach, we leverage on this built-in feature and map the evaluation of pre- and post-conditions to the evaluation of pre and post-execution constraints, by automatically translating φ_{pre} and φ_{post} formulae for each task into YAWL conditional expressions. While arithmetic, comparison and logical operators in our annotation language directly map to the operators supported by YAWL, predicates and functions can be resolved by appropriate XQuery expressions¹¹.

Representing Planlet Annotations in PDDL. In order to exploit our planning-based recovery mechanism, every task/annotation/property associated to a PLANLET needs to be translated in PDDL. A PDDL definition consists of two parts: the domain and the problem definition. The planning domain is built starting by the definition of basic/derived predicates, object/numeric functions and data types as shown in the previous sections, and by making explicit the *actions* associated to each annotated task stored in the repository linked to the PLANLET under execution, together with the associated pre-conditions, effects and input parameters. Basically, the planning domain describes how predicates and functions may vary after an action execution, and reflects the contextual properties constraining the execution of tasks stored in a specific PLANLET

¹⁰ As φ_{pre} and φ_{post} are *closed* formulae, their truth values can be considered as the answers to the corresponding boolean queries, given the interpretation \mathcal{I} .

¹¹ We recall that no free variables are allowed and all formulae are closed.

repository. Our annotation syntax allows to represent planning domains and problems with the complexity of those describable in PDDL version 2.2¹² [2]. In the following, we discuss how our annotations are translated into a PDDL file representing the planning domain:

- the *name* and the *domain* of a data type corresponds to an *object type* in the planning domain;
- basic and derived predicates have a straightforward representation as *relational predicates* (templates for logical facts) and *derived predicates* (to model the dependency of given facts from other facts);
- numeric functions correspond to PDDL *numeric fluents*, and are used for modeling non-boolean resources (e.g., the battery level of a robot);
- object functions do not have a direct representation in PDDLv2.2, but may be replaced as relational predicates. Since an object function $f : Object^n \rightarrow Object$ map tuples of objects with domain types D^n to objects with co-domain type U , it may be coded in the planning domain as a relational predicate P of type (D^n, U) ;
- a given YAWL task, together with the associated pre-conditions and effects and input parameters, is translated in a PDDL *action schema*. An action schema describes how the relational predicates and/or numeric fluents may vary after the action execution.

When an exception arises, on a same planning domain a new planning problem is built at run-time, through the description of an initial state (that corresponds to the invalid state of the process s) and the description of the desired goal (a safe state s' , derived from the violated pre- or post-condition).

- for each data type defined in the planning domain, all the possible object instances of that particular data type are explicitly instantiated as *constant symbols* in the planning problem (e.g., the fact that $a1, a2, a3, a4$ are *Actors*, $rb1$ and $rb2$ are *Robots*, $loc00, \dots, loc33$ are *Locations*);
- a representation of the *initial state* of the planning environment is needed. Basically, the initial state of the planning problem corresponds to an invalid state (i.e., a state that needs to be fixed after a pre- or post-condition violation during the process execution). It is composed by a conjunction of relational predicates, derived predicates (e.g., the information about which actors/robots are currently connected to the network) and by the current value of each numeric fluent (e.g., the battery charge level for each robot);
- the *goal state* of the planning problem is a logical expression over facts. In our approach, the goal state is built in order to reflect a safe state to be reached after the execution of a recovery procedure. Suppose that t is the task whose pre-conditions $Pre(t)$ (or post-conditions $Post(t)$) are not verified. The safe

¹² PDDLv2.2 enables the representation of realistic planning domains, with actions and goals involving numerical expressions, operators with universally quantified effects or existentially quantified preconditions, operators with disjunctive or implicative preconditions, derived predicates and plan metrics. However, currently, our formalism does not allow to represent conditional and universally quantified effects.

Table 1. Time performances of LPG-td for adaptation problems of growing complexity

Length of the recovery proc.	Problem instances	Avg. time needed for a sub-optimal sol. (sec)	Avg. length of a sub-optimal sol.	Avg. time needed for a quality sol. (sec)
1	29	6,769	3	7,768
2	36	7,213	3	16,865
3	32	7,846	4	24,123
4	25	8,128	5	37,017
5	21	8,598	8	39,484
6	17	8,736	9	52,421
7	13	9,188	13	73,526
8	12	9,953	14	81,414

state s' corresponding to the goal state is generated starting from the invalid state s , by substituting the wrong facts that led to the exception with the content of the pre-conditions (or post-conditions) violated.

6 Experiments

In order to investigate the feasibility of the PLANLET approach, we performed some testing to learn the time amount needed for synthesizing a recovery plan for different adaptation problems. We made our tests by using the LPG-td planner¹³ [7]. Such a planner is based on a stochastic local search in the space of particular “action graphs” derived from the planning problem specification. The basic search scheme of LPG-td is inspired to Walksat [16], an efficient procedure for solving SAT-problems. More details on the search algorithm and heuristics devised for this planner can be found at [7,6]. We chose LPG-td as (i) it treats the full range of PDDL2.2 [2] (that is characterized for enabling the representation of realistic planning domains) and (ii) even if it is primarily thought as a satisficing planner, it is able to compute also quality plans under a pre-specified metric. In fact, LPG-td has been developed in two versions: a version tailored to computation speed, named LPG-td.speed, which produces *sub-optimal plans*, and a version tailored for *plan quality*, named LPG-td.quality. LPG-td.speed generates sub-optimal solutions that do not prove any guarantee other than the correctness of the solution. LPG-td.quality differs from LPG-td.speed basically for the fact that it does not stop when the first plan is found but continues until a stopping criterion is met. In our experiments, the optimization criteria was fixed as the minimum number of actions needed for the planner to reach the goal. It is important to underline that satisficing planning is easy (polynomial), while optimal planning is hard (NP-complete) [8]. The experimental setup was performed with the test case shown in our running example. We stored in the PLANLET repository 20 different emergency management tasks, annotated with 28 relational predicates, 2 derived predicates and 4 numeric fluents, in order to make the planner search space very challenging. Then, we provided 185 different planning problems of different complexity, by manipulating ad-hoc the

¹³ LPG-td was awarded at the 4th International Planning Competition (IPC 2004, <http://ipc.icaps-conference.org/>) as the “top performer in plan quality”.

values of the initial state and the goal in order to devise adaptation problems of growing complexity¹⁴. As shown in Table 1, the column labeled as “Length of the recovery procedure” indicates the smallest number of actions needed for devising a plan of a specific length. Our purpose was to measure (in seconds) the computation time needed for finding a sub-optimal solution and a quality solution for problems that require a recovery procedure of growing complexity. The column labeled as “Average length of a sub-optimal solution” indicates the average number of actions that compose a sub-optimal solution for a problem of a given complexity. A sub-optimal solution is found in less time than a quality one, but generally it includes more tasks than the ones strictly needed. This means that when the complexity of the recovery procedure grows, the quality of a sub-optimal solution decreases. For example, as shown in table 1, on 21 different planning problems requiring a recovery procedure of length 5, the LPG-td planner is able to find, on average, a sub-optimal plan in 8,598 seconds (with 3 more tasks, on average) and a quality plan (which consists exactly of the 5 tasks needed for the recovery) in 39,484 seconds, without the need of any domain expert intervention. Consequently, the approach is feasible for medium-sized dynamic processes used in practice¹⁵.

7 Conclusions

In this paper, we have introduced the concept of PLANLETS, self-contained YAWL specifications featuring automatic adaptation for dynamic processes, based on modeling of pre- and post-conditions of tasks and the use of planning techniques. In contrast to most existing approaches, PLANLET covers on automatic adaptation for processes at runtime that do not need any human interaction. We have shown the feasibility of the approach by discussing its deployment on top of YAWL and by showing some experimental tests based on a real process scenario. Such tests have provided useful insights on the cases in which an automatic approach is convenient wrt. more traditional exception handlers defined at design-time. The assumptions of classical planning (determinism in the action effects, model completeness, etc.) we used for modeling dynamic processes has a twofold consequence. On the one hand, we can exploit the good performance of classical planners (e.g., LPG-td) to solve real-world problems with a realistic complexity; on the other hand, classical planning imposes some restrictions for addressing more expressive problems, including incomplete information, preferences and multiple task effects. Future works will include an extension of our approach dealing with the above aspects, with the purpose to maintain the planning process very responsive.

¹⁴ Some test instances, together with the inputs for the planner, are available at the URL: http://www.dis.uniroma1.it/marrella/public/Planlets.CoopIS2012_TestCases.zip.

¹⁵ We did our tests by using an Intel U7300 CPU 1.30GHz Dual Core, 4GB RAM machine.

Acknowledgements. This work has been partly supported by the SAPIENZA projects TESTMED and SUPER, by the Italian national PIA Calabria project COSM Factory and by the EU projects Greener Buildings and SmartVortex. The authors thank Arthur H.M. ter Hofstede and Massimiliano de Leoni for useful insights and discussions.

References

1. de Leoni, M., Mecella, M., De Giacomo, G.: Highly Dynamic Adaptation in Process Management Systems Through Execution Monitoring. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 182–197. Springer, Heidelberg (2007)
2. Edelkamp, S., Hoffmann, J.: PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Tech. rep., Albert-Ludwigs-Universitat Freiburg, Institut fur Informatik (2004)
3. Ferreira, H., Ferreira, D.: An integrated life cycle for workflow management based on learning and planning. *Int. J. Coop. Inf. Syst.* 15, 485–505 (2006)
4. Friedrich, G., Fugini, M., Mussi, E., Pernici, B., Tagni, G.: Exception handling for repair in service-based processes. *IEEE Trans. on Soft. Eng.* 36, 198–215 (2010)
5. Gajewski, M., Meyer, H., Momotko, M., Schuschel, H., Weske, M.: Dynamic failure recovery of generated workflows. In: DEXA 2005 (2005)
6. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in Lpg. *J. Art. Int. Res.* 20(1), 239–290 (2003)
7. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: Lpg-td: a fully automated planner for PDDL2.2 domains. In: ICAPS 2004 (2004)
8. Helmert, M.: Complexity results for standard benchmark domains in planning. *Art. Int.* 143, 219–262 (2003)
9. Jarvis, P., Moore, J., Stader, J., Macintosh, A., du Mont, A.C., Chung, P.: Exploiting AI technologies to realise adaptive workflow systems. In: AAAI Workshop on Agent-Based Systems in the Business Context (1999)
10. Lenz, R., Reichert, M.: IT support for healthcare processes. Premises, challenges, perspectives. *Data Knowl. Eng.* 61, 39–58 (2007)
11. Marrella, A., Mecella, M., Russo, A.: Featuring automatic adaptivity through workflow enactment and planning. In: CollaborateCom 2011 (2011)
12. Marrella, A., Mecella, M., Russo, A., ter Hofstede, A.H.M., Sardiña, S.: Making YAWL and SmartPM interoperate: Managing highly dynamic processes by exploiting automatic adaptation features. In: BPM, Demos (2011)
13. R-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating planning and scheduling in workflow domains. *Exp. Syst. with Applications* 33(2) (2007)
14. 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)
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 (2008)
16. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: AAAI 1994 (1994)
17. 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 (2009)

18. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data Knowl. Eng.* 66, 438–466 (2008)
19. Weber, B., Wild, W., Lauer, M., Reichert, M.: Improving exception handling by discovering change dependencies in adaptive process management systems. In: *BPI 2006* (2006)
20. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: *HICSS 2001* (2001)

Discovering Context-Aware Models for Predicting Business Process Performances

Francesco Folino, Massimo Guarascio, and Luigi Pontieri

Institute for High Performance Computing and Networking (ICAR)
National Research Council of Italy (CNR)
Via Pietro Bucci 41C, I87036 Rende (CS), Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Discovering predictive models for run-time support is an emerging topic in Process Mining research, which can effectively help optimize business process enactments. However, making accurate estimates is not easy especially when considering fine-grain performance measures (e.g., processing times) on a complex and flexible business process, where performance patterns change over time, depending on both case properties and context factors (e.g., seasonality, workload). We try to face such a situation by using an ad-hoc predictive clustering approach, where different context-related execution scenarios are discovered and modeled accurately via distinct state-aware performance predictors. A readable predictive model is obtained eventually, which can make performance forecasts for any new running process case, by using the predictor of the cluster it is estimated to belong to. The approach was implemented in a system prototype, and validated on a real-life context. Test results confirmed the scalability of the approach, and its efficacy in predicting processing times and associated SLA violations.

1 Introduction

Process mining techniques [11] are widely reckoned as a precious tool for the analysis of business processes, owing to their capability to extract useful information out of historical process logs, possibly providing the analyst with a high-level process model. An emerging research stream (see, e.g., [6,13]) concerns the induction of state-aware models for predicting some relevant performance metrics, defined on process instances. For example, in [13], an annotated finite-state model is induced from a given log, where the states correspond to abstract representation of log traces. Conversely, a non-parametric regression model is used in [6] to build the prediction for a new (possibly partial) trace upon its similarity to a set of historical ones, while evaluating traces' similarity based on the comparison of their respective abstract views. The interest towards such novel mining tools stems from the observation that performance forecasts can be exploited to improve process enactments, through, e.g., task/resource recommendations [9] or risk notification [5]. However, accurate forecasts are not easy to make for fine-grain measures (like, e.g., processing times), especially when the analyzed

process shows complex and flexible dynamics, and its execution schemes and performances change over time, depending on the context. In fact, the need to recognize and model the influence of context factors on process behavior is a hot issue in BPM community (see, e.g., [14]), which calls for properly extending traditional approaches to process modeling (and, hopefully, to process mining). In general, a way to increase process model precision is to partition the log by ad-hoc clustering methods [10,7,8], and to find a (more precise) model for each cluster, while regarding this latter as evidence for a peculiar execution scenario of the process. To the best of our knowledge, however, all previous clustering-oriented process mining approaches only focused on control-flow aspects, with no connections with the discovery of performance predictors.

In this paper we right attempt to overcome the above limitations by proposing an ad-hoc predictive clustering approach, capable to detect different context-related execution scenarios (or *process variants*), and to equip each of them with a tailored performance-prediction model. Our ultimate goal is to find a novel kind of predictive model, where performance forecasts for any (unfinished) process instance, are made in two steps: the instance is first assigned to a reference scenario (i.e., cluster), whose performance model is then used to eventually make the forecast. Technically, we extend and integrate a method for inducing predictive performance models [13] and a logics-oriented approach to predictive clustering [3], where the discovered model, named Predictive Clustering Tree (PCT), takes the form of a decision-tree. Specifically, the discovery of such scenarios (i.e., clusters) is carried out by partitioning the log traces based on their associated context features, which may include both internal properties of a case (e.g. the amount of goods requested in an order management process) and external factors that characterize the situation where it takes place (e.g., workload, resource availability, and seasonality indicators). Notably, the complex structure of (performance-annotated) process logs makes a trivial application of PCT learning methods likely ineffective and/or computationally expensive. We hence devise a method for encoding each log trace in a propositional form, featuring both its context properties and some associated performance measurements.

Organization. The rest of the paper is structured as follows. Section 2 introduces some notation and basic concepts. The specific problem faced in the paper and the proposed solution approach are described in Section 3. Section 4 discusses an implementation of the approach, and its usage in a real-life setting (as well as the quality metrics used for the evaluation). After discussing experiment results in Section 5, we finally draw a few concluding remarks in Section 6.

2 Formal Framework

Following a standard approach in the literature, we assume that for each process instance (a.k.a “case”) a *trace* is recorded, encoding the sequence of *events* happened during the relative enactment. Different data parameters (e.g., the amount of goods asked in a order-handling process) can be kept for any process instance, while each event is associated with a process task and a timestamp –

we here disregard other event properties, such as, e.g., task parameters or executors. We also assume that a additional features can be associated with each trace that characterize the context where it takes place, and capture environmental factors (which may well influence performances).

Let us first denote by \mathcal{T} and E the (fixed) reference universes of all (possibly partial) traces and associated events that may appear in a log. Moreover, let $\hat{\mu} : \mathcal{T} \rightarrow \mathcal{M}$ the unknown function assigning a performance value to each trace — w.r.t. to a given reference performance metrics and an associated space \mathcal{M} of values. Note that $\hat{\mu}$ abstractly indicates the final target of our search, in that we aim at eventually predicting the values of the metrics on any novel enactment. We also assume that two kinds of context properties are defined for a process instance: (i) (“intrinsic”) *case attributes* A_1, \dots, A_q , with associated domains D^{A_1}, \dots, D^{A_q} , resp., and (ii) (“extrinsic”) *environmental features* B_1, \dots, B_r , with domains D^{B_1}, \dots, D^{B_r} , resp. — this latter kind of data are meant to capture the state of the BPM system in the moment when the instance starts. Finally, for any sequence s , let $len(s)$ denote its length, and $s[i]$ the element in position i , for $i = 1 \dots len(s)$. Finally, $s[i]$ is its prefix of s of length i , for $i = 1 \dots len(s)$, and $s[0] = \langle \rangle$ (the empty sequence). Some further concepts and notation are formally introduced next to conveniently refer to log contents.

Definition 1 (Trace). A *trace* $\tau \in \mathcal{T}$ is a triple $\langle v, \bar{a}, s \rangle$ such that id is a unique identifier, $\bar{a} \in D^{A_1} \times \dots \times D^{A_q}$ are its data, and s is a sequence of events. For simplicity, let us also denote $v = id(\tau)$, $\bar{a} = data(\tau)$, $s = seq(\tau)$, $len(\tau) = len(s)$, and $\tau[i] = s[i]$. Moreover, $env(\tau) \in D^{B_1} \times \dots \times D^{B_r}$ are the environment features associated with any trace τ , and $context(\tau) \in D^{A_1} \times \dots \times D^{A_q} \times D^{B_1} \times \dots \times D^{B_r}$ is the juxtaposition of vectors $data(\tau)$ and $env(\tau)$. Finally, $\tau[i] = \langle v^i, \bar{a}^i, s^i \rangle$ is a *prefix* of τ , for $i = 0 \dots len(\tau)$, such that v^i is a new identifier, $\bar{a}^i = \bar{a}$, $s^i = s[i]$, $env(\tau[i]) = env(\tau)$, and $context(\tau[i]) = context(\tau)$. \square

Definition 2 (Log). A *log* L (over \mathcal{T}) is a finite subset of \mathcal{T} . Moreover, the *prefix set* of L , denoted by $\mathcal{P}(L)$, is the set of all prefix traces that can be extracted from L , i.e., $\mathcal{P}(L) = \{\tau[i] \mid \tau \in L \text{ and } 0 \leq i \leq len(\tau)\}$. For any log L , we will always assume that $\hat{\mu}(\tau)$ is known for any prefix trace $\tau \in \mathcal{P}(L)$. \square

Note that any prefix $\tau[i]$ in Def. 1 is a partial unfolding of τ sharing its context data, while the last statement in Def. 2 can be handled by defining an auxiliary function encoding $\hat{\mu}$ on the prefixes of past log traces — e.g., the (real) remaining time of any prefix of such a trace τ is $\hat{\mu}_{RT}(\tau[i]) = time(\tau[len(\tau)]) - time(\tau[i])$.

2.1 State-Aware Performance Prediction

A *Performance Prediction (Process) Model (PPM, for short)*, is for us a model that can predict the performance value of any future process enactment, represented as a partial trace. Such a model, indeed, can be regarded as a function $\mu : \mathcal{T} \rightarrow \mathcal{M}$ that tries to estimate $\hat{\mu}$ all over the reference universe of traces. Learning a PPM is then a special induction problem, where the training set is represented as a log L , such that the value $\hat{\mu}(\tau)$ of the target measure is known

for each (sub-)trace $\tau \in \mathcal{P}(L)$. Different solutions were proposed to this problem [13,6], which share the idea of capturing the dependence of performance values on traces (i.e., case histories) by regarding these latter at suitable abstraction levels.

Definition 3 (Trace Abstraction Functions). Let $h \in (N) \cup \{\infty\}$ be a threshold on past history. A *trace abstraction function* $abs_h^{mode} : \mathcal{T} \rightarrow \mathcal{R}$ is a function mapping each trace $\tau \in \mathcal{T}$ to an element $abs_h^{mode}(\tau)$ in a space \mathcal{R} of abstract representations. For any $\tau \in \mathcal{T}$, while denoting $n = len(\tau)$ and $j = n - h + 1$ if $n > h$ and $j = 1$ otherwise, it is: **(i)** $abs_h^{list}(\tau) = \langle task(\tau[j]), \dots, task(\tau[n]) \rangle$; **(ii)** $abs_h^{bag}(\tau) = [(t, p) \mid t \in abs_h^{set}(\tau) \text{ and } p = |\{\tau[k] \mid j \leq k \leq n, task(\tau[k]) = t\}|]$, and **(iii)** $abs_h^{set}(\tau) = \{task(\tau[j]), \dots, task(\tau[n])\}$. \square

Each $\alpha \in \mathcal{R}$ is a high level representation for some traces, capturing some hidden state of the process analyzed. In particular, the three concrete abstraction functions defined above maps traces to sequences, sets and multisets, respectively, of task identifiers, and specialize the functions presented in [13] – we here only consider to abstract each trace event into its associated task, while disregarding other event properties (e.g., executors). This restriction could be easily removed from our approach – even though, often, using multiple properties for generalizing may lead to a combinatorial explosion of the abstract representations produced (and to overfitting patterns). In [13], a Finite State Machine (FSM) model is derived, such that a one-to-one mapping exists between its states and the representations produced by some abstraction function abs , while each transition is labelled with an event property (namely, a task label in our case). For example, let us assume that abs_{∞}^{list} is used, and that a, b and c refer to three process tasks. Then, the resulting FSM model will feature a transition labelled with c from state $\langle a, b \rangle$ to state $\langle a, b, c \rangle$, if there is some trace τ in the input log such that $abs_{\infty}^{list}(\tau(i)) = \langle a, b \rangle$ and $abs_{\infty}^{list}(\tau(i + 1)) = \langle a, b, c \rangle$. In order to make this model capable to make predictions (w.r.t. a measure μ), it is turned into an *Annotated Finite State Machine (A-FSM)*, by equipping each node s with a bag gathering all the values that $\hat{\mu}$ takes at the end of any trace prefix $\tau \in \mathcal{P}(L)$ such that $abs(\tau)$ coincides with the abstraction of s . These measurements help estimate the target measure for any new process instance reaching s , e.g. by simply storing an aggregate statistics (e.g., the average) over them. In principle, our clustering-based scenario discovery scheme could be combined with other state-aware prediction techniques, for it is parametric to the kind of model that is eventually learnt for each scenario. However, in this paper we only consider using A-FSM models, and their associated learning method, to this purpose.

2.2 Predictive Clustering

The core idea of *Predictive Clustering* approaches [2] is that, once discovered an appropriate clustering model, a prediction for a new instance can be based only on the cluster where it is deemed to belong, according to some suitable assignment function. The underlying belief is that the higher similarity between instances of the same cluster will help derive a more accurate predictor – w.r.t. one induced from the whole dataset.

To this end, two kinds of features are considered for any element z in a given space $Z = X \times Y$ of instances: *descriptive* features, denoted by $descr(z) \in X$, and *target* features, denoted by $targ(z) \in Y$ – which are those to be predicted.

Then, a *predictive clustering model (PCM)*, for a given training set $L \subseteq Z$, is a function $m : X \rightarrow Y$ of the form $m(x) = p(c(x), x)$, where $c : X \rightarrow \mathbb{N}$ is a partitioning function and $p : \mathbb{N} \times X \rightarrow Y$ is a prediction function.

An important class of such models are *Predictive Clustering Trees (PCTs)* [2,3], where the cluster assignment function is encoded by a *decision tree*, which can be learnt by recursively partitioning the training set. At each step, a split test is greedily chosen, over one descriptive feature, which (locally) minimizes:

$$loss_d(m, L) \sum_{C_i} |C_i \in c(L)| / |T| \times \sum_{z \in C_i} d(targ(z), p(z))^2 \quad (1)$$

where C_i ranges over the current partition of L , and d is a distance measure d over Z . – When working with numeric targets, a good trade-off between scalability and accuracy is typically achieved by simply instantiating d with the classical Euclidean distance over target features only. In this case, $targ(avg(C_i))$ over the target subspace can be also used as the local (constant) predictor of cluster C_i , with $avg(C_i) = |C_i|^{-1} \times \sum_{z \in C_i} z$ – i.e., the cluster’s average/centroid.

A variety of PCT learning methods exists in the literature, which differ in the type/number of target features (e.g., decision trees, regression trees, multi-target regression models, clustering trees), or in the underlying representation of data instances – namely, relational (e.g., system TILDE [2]) and propositional (e.g., system CLUS [1]). In our setting, we focus on the discovery of a multi-target regression PCT out of propositional data, mainly owing to scalability reasons.

The core assumption under our work is that process performances really depend on context factors. Hence, to predict the performances of any (partial) trace τ , we regard its associated context data $context(\tau)$ as descriptive attributes.

We can now state the specific kind of performance model we want to discover.

Definition 4 (Context-Aware Performance Prediction Model (CA-PPM)).

Let L be a log on trace universe \mathcal{T} , with context features $context(\mathcal{T})$, and $\hat{\mu} : \mathcal{T} \rightarrow \mathcal{M}$, be a performance measure, known for all $\tau \in \mathcal{P}(L)$. Then, a *context-aware performance prediction model (CA-PPM)* for L is a pair $M = \langle c, \langle \mu_1, \dots, \mu_k \rangle \rangle$, encoding a predictive clustering model g_M for $\hat{\mu}$, such that: **(i)** $c : context(\mathcal{T}) \rightarrow \mathbb{N}$, **(ii)** $\mu_i : \mathcal{T} \rightarrow \mathcal{M}$, for $i \in c(context(\mathcal{T}))$, and **(iii)** $g_M(\tau) = \mu_j(\tau)$ with $j = c(context(\tau))$. \square

Notice that the dependence of the target measure on context features relies on the separate modeling of different context-dependent execution scenarios (i.e., clusters), while the performance predictions are eventually based on a cluster assignment function c , which estimates the membership of (possibly novel) process instances to these scenarios. This model is a special kind of *PPM* model, relying on a predictive clustering one. As such, it can be instantiated by combining a predictive clustering tree (PCT) and multiple (performance-)annotated FSM (A-FSM) models, as building blocks for implementing the functions c and each μ_i , respectively, as discussed next.

3 Problem Statement and Solution Approach

In principle, seeking an explicit encoding for the hidden performance measure $\hat{\mu}$, based on a given log L , can be stated as the search for a CA-PPM (cf. Def. 4) minimizing some loss measure, like that in Eq. 1, possibly evaluated on an different sample $L' \subseteq \mathcal{T}$ than the one used as training set. However, to avoid incurring in prohibitive computation times, we rather follow a heuristics approach, where the problem is turned into a combination of two simpler ones, as defined below.

Definition 5 (Problem CAPP). *Given a log L over \mathcal{T} , and a performance measure $\hat{\mu}$ only defined on $\mathcal{P}(L)$; Solve the following subproblems, sequentially: [CAPP-S1]: find a function c (locally) minimizing the loss over a concise representation of the given traces and associated measurements, irrespectively of the cluster-wise prediction function q ; and [CAPP-S2]: find a function q based on the partition $c(L)$ produced by c (keeping it fixed to as found before). \square*

Such a simplifying rephrasing of the problem frees us from the burden of simultaneously searching over both any possible partitioning c and all of its associated prediction functions q . Moreover, we want to reuse existing tools for the induction of PCTs and of A-FSM models. This clearly requires to properly define the structure of the training data used to learn a PCT model, since a naïve application of PCT induction algorithms to log contents might lead to unsatisfactory achievements in terms of both scalability and prediction accuracy.

To this end, we propose the adoption of a propositional view of the log, where each (fully unfolded) trace in L acts as an individual training example. We hence dismiss the natural idea of learning the clustering model based on all partial traces in $\mathcal{P}(L)$ (and on their associated performance measurements), for two reasons. First, if working explicitly with all partial traces, the number of training samples will grow substantially, especially in the case where log traces were generated by a process featuring complex and flexible control logics (i.e., many tasks and a high degree of non-determinism). More importantly, since performance values tend to change notably along the course of a process instance – this is right the rationale behind state-aware prediction approaches like [6,13] – the learner may get confused when trying to separate groups of instances with similar target measurements. Think, e.g., to the case of the remaining processing time measure, which progressively decreases as a process enactment goes forward.

On the other hand, using full historical traces as clustering instances, we must decide what are their associated targets, which the PCT learning algorithm has to approximate at best. In fact, each trace τ corresponds to a sequence of target values $(\hat{\mu}(\tau(1)), \dots, \hat{\mu}(\tau))$, and we do not want to use sequences as cluster prototypes, in order to keep the evaluation of candidate split tests fast enough.

As a heuristics solution, each trace is mapped into a vector space, where the dimensions correspond to relevant states of the (hidden) process model. Such target features are computed by way of the trace abstraction functions in Def. 3, which attempt to transform, indeed, each trace into an abstract representation of its enactment state, based on its past history.

Input: A log L over a trace universe \mathcal{T} , with data attributes $A = A_1, \dots, A_q$, and environment features $B = B_1, \dots, B_r$, a target measure $\hat{\mu}$ known over $\mathcal{P}(L)$,
 a trace abstraction function abs , and a relevance threshold $\sigma \in [0, 1]$.

Output: A CA-PPM model for L (fully encoding $\hat{\mu}$ all over \mathcal{T}).

Method: Perform the following steps:

- 1 Associate a vector $context(\tau)$ with each $\tau \in L$, by computing features $env(\tau)$
- 2 Compute a set $PA_\sigma(L, abs)$ of pivot state abstractions (cf. Def. 6)
- 3 Let $PA_\sigma(L, abs) = \{\alpha_1, \dots, \alpha_s\}$
- 4 Build a *performance sketch* \mathcal{S} for L using context vectors and $PA_\sigma(L, abs)$
 // $\mathcal{S} = \{ (id(\tau), context(\tau), \langle val(\tau, \alpha_1), \dots, val(\tau, \alpha_s) \rangle) \mid \tau \in L \}$ – cf. Eq.2
- 5 Learn a PCT T with classification (resp., prediction) function c (resp., q) using $context(\tau)$ (resp., $val(\tau, \alpha_i), i=1..s$) as descript. (resp., target) features, $\forall \tau \in L$
- 6 Let $L[1], \dots, L[k]$ denote the discovered clusters – with $\{1, \dots, k\} = c(\mathcal{S})$
- 7 **for each** $L[i]$ **do**
- 8 Induce an FSM model f from $L[i]$, using abs as abstraction function
- 9 Derive an A-FSM f^+ model from f
- 10 Define prediction function $\mu_i : \mathcal{T} \rightarrow \mathcal{M}$ (for cluster i) based on f^+
- 11 **end**
- 12 **return** $\langle c, \{ \mu_1, \dots, \mu_k \} \rangle$

Fig. 1. Algorithm CA-PPM Discovery

Specifically, given an abstraction function $abs : \mathcal{T} \rightarrow \mathcal{R}$, a “candidate” target feature can be defined for each abstract (state) representation $\alpha \in \mathcal{R}$, such that the value $val(\tau, \alpha)$ of this feature for any trace τ is computed as follows:

$$val(\tau, \alpha) = \begin{cases} \text{NULL}, & \text{if } abs(\tau[i]) \neq \alpha \forall i \in \{0, \dots, len(\tau)\}; \\ agg(\langle \hat{\mu}(\tau[i_1]), \dots, \hat{\mu}(\tau[i_s]) \rangle), & \text{otherwise.} \end{cases} \quad (2)$$

where $\{i_1, \dots, i_s\} = \{j \in \mathbb{Z} \mid 0 \leq j \leq len(\tau) \text{ and } abs(\tau[i]) = \alpha\}$, and $i_j < i_k$ for any $0 \leq j < k \leq s$, while agg is a function aggregating a sequence of measure values into a single one (e.g., the average, median, first, last in the sequence). Note that, for all the tests in Section 5, we always selected the last sequence element.

AS the number of state abstractions may be high, some suitable strategy is needed to select an optimal subset of them, as to prevent the PCT learner from getting lost in a high-dimensional and sparse target space (yet taking long computation times). To this end, we devise an ad-hoc, greedy, selection strategy, to identify a restricted set of “pivot” state abstractions, which looks to be the (locally) best ones in discriminating different performance profiles. The selection criterion used to this purpose relies on a fixed scoring function $\phi : \mathcal{R} \times 2^{\mathcal{T}} \rightarrow [0, 1]$ (which will be discussed in details later on), which assigns each state abstraction $\alpha \in \mathcal{R}$ to a score $\phi(\alpha, L)$, quantifying the confidence in α making a profitable target feature w.r.t. the search of a predictive clustering for L . More precisely:

Definition 6 (Pivot State Abstraction). Let L be a log, $abs : \mathcal{T} \rightarrow \mathcal{R}$ be a trace abstraction function, and $\sigma \in [0, 1]$ be a relevance threshold. Then, any

$a \in \mathcal{R}$ is a *pivot state abstraction* for L and σ w.r.t. abs , if $\phi(\alpha, L) \geq \sigma$. Moreover, $PA_\sigma(L, abs)$ is the set of all pivot state abstractions for L and σ w.r.t. abs . \square

Provided with a set of pivot state abstractions $PA_\sigma(L, abs) = \{\alpha_{j1}, \dots, \alpha_{ju}\}$, subproblem **CAPP-S1** can be eventually faced by solving a standard (multi-regression) PCT induction on a dataset where: (i) each trace τ in the log corresponds to a distinct instance, (ii) the vector $context(\tau)$ encodes the descriptive features of τ and (iii) $val(\tau, \alpha_{j1}), \dots, val(\tau, \alpha_{ju})$ are the target features of τ . This dataset, called in the following a *performance sketch* of L (w.r.t. abs and σ), offers a propositional view over the log, enabling for a fast and effective calculation of a predictive clustering model.

A detailed description of the different steps of our approach is given in the **CA-PPM Discovery** algorithm, shown in Fig. 1. The meaning of its steps is quite straightforward, as it coincide to the computation process discussed so far. However, it is worth remarking that the induction of an FSM model for each discovered cluster (**step 8**), and its subsequent annotation with performance measurements (**step 9**) are carried out by taking advantage of the techniques presented in [13]. Notably, the performance measurements associated with each state in the model are eventually aggregated into a single constant estimator (namely, the average over them all), in the implementation of $\mu[i]$ (**step 10**). Moreover, whenever a new trace τ generates an unseen sequence of states, as a simple workaround, the function can be extended in a way that its next estimate for τ will be based on the last valid one made for it. Finally, the selection of pivot state features performed in **step 2** hinges on the following scoring function:

$$\phi(\alpha, L) = \sqrt[3]{\phi_{var}(\alpha, L) \times \phi_{corr}(\alpha, L) \times \phi_{supp}(\alpha, L)} \quad (3)$$

where $\phi_{var}(\alpha, L)$, $\phi_{corr}(\alpha, L)$, and $\phi_{supp}(\alpha, L)$ are all functions ranging on $[0, 1]$.

Basically, function $\phi_{var}(\alpha, L)$ depends on the variability of the values produced by α on all input traces (i.e., $\{val(\alpha, \tau) | \tau \in L\}$) and gives preference to higher-variability features – the more the variability of trace measures the higher the score. Function $\phi_{corr}(\alpha, L)$ measures instead the maximal correlation between the value taken by the feature over each trace and the corresponding value of each descriptive (context) feature – the higher the correlation the higher the score. Finally, $\phi_{supp}(\alpha, L)$ simply is $2 \times \min(0.5, |\{\tau \in L \mid val(\tau, \alpha) > 0\}|)$ – low support state abstractions hardly help find significant groups of traces, indeed. In a sense, the overall scoring function is biased towards features guaranteeing a good compromise between support, correlation with descriptive features (which are the ones guiding the partitioning of log traces) and performance values’ variability (in order to find clusters showing quite different performance models).

Before leaving the section, let us observe that the peculiar feature selection subproblem faced here is beyond the scope of the attribute selection capabilities of the heuristics search method embedded in predictive clustering algorithms, due to the fact that our candidate features correspond to target variables, and not to predictor ones. This is also the reason why we cannot trivially reuse feature-selection (i.e., attribute-selection) techniques available in the literature.

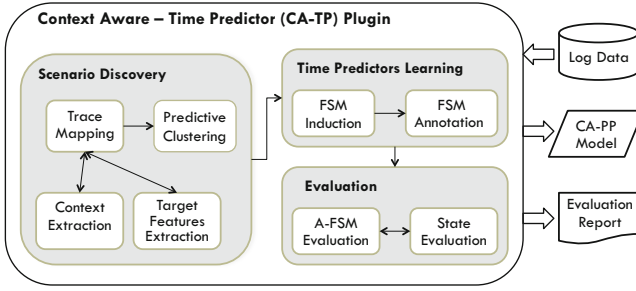


Fig. 2. CA-TP plug-in architecture

4 Case Study: Time Prediction on a Logistics Process

After illustrating the prototype system, in Section 4.1, in the remainder of this section, we discuss the experiments carried out on a real log data and the obtained results. In particular, in Section 4.2, we first illustrate the application scenario, by discussing the kind of data involved in it. Then, in Section 4.3, we introduce the setting adopted to evaluate the quality of discovered models. Finally, in Section 5, the results of tests performed on this scenario are evaluated.

4.1 The Prototype System: Plugin CA-TP

As a specialized version of algorithm *CA-PPM Discovery*, we implemented a prototype system, named *CA-TP* (i.e., *Context-Aware Time Prediction*), which can discover a *CA-PPM* for predicting the remaining processing time measure, in order to assess the validity of the approach on practical situations. The prototype system has been developed as plug-in for ProM framework [12], a popular Process Mining framework. The logical architecture of the system is sketched in Figure 2, where arrows between blocks stand for information flows. The whole mining process is driven by the control logic of the the plug-in, while the other modules basically replicate the main computation phases of the algorithm. By *Log Data* we here denote a collection of process logs represented in the MXML [12] format.

The *Scenario Discovery* module is responsible for identifying behaviorally homogeneous groups of traces in terms of both context data and remaining times. In particular, the discovery of different trace clusters is carried out by the *Predictive Clustering* submodule which groups traces sharing both similar descriptive and target values. This latter module leverages the *CLUS* system [1], a predictive clustering framework for inducing PCT models out of propositional data. Such a model is found by trying to optimize the multi-target regression models (w.r.t. a given set of target attributes) of clusters obtained by partitioning the space of descriptive attributes. In this regard, the *Trace Mapping* submodule acts as a “translator” which converts all log traces into propositional tuples, according to the (ARFF) format used in *CLUS*. As explained above, this mapping relies on the explicit representation of both context data and target attributes, derived

from the original (MXML) log. In particular, the *Context Extraction* module extract extrinsic (environmental) context features, including workload indicators and aggregated time dimensions, and add them to the descriptive attributes of each trace. Notice that this module takes advantage of auxiliary data structures to efficiently search all log data that help capture the local context of any trace τ . In particular, two indexes (based on search trees) over log traces are used, which allow to quickly find all the traces that started or finished, respectively, in a given time range. In fact, these indexes are meant to retrieve all the events occurred during the enactment of τ , to reconstruct its context. Complementarily, the *Target Features Extraction* submodule provides the *Trace Mapping* one with an quasi-optimal set of trace abstractions (obtained by combining trace activities in lists/sets/bags, possibly bounded in their size by a parameter h), which will be eventually used as target features for the predictive clustering step.

Log traces, labeled with cluster IDs, are delivered to the *Time Predictors Learning* module, which, leveraging the approach in [13], derives a collection of *A-FSM* models. More specifically, the submodule *FSM Induction* is used to build a transition model for each cluster, whereas the *FSM Annotation* annotates them with time information. As a final result, a *CA-PPM* model is eventually built, which integrates multiple *A-FSM* models for scenario-specific time predictions, with a set of logical rules (corresponding to the leaves of a PCT model) for discriminating among the discovered scenarios. For inspection purposes and further analysis, the whole model is then stored in an ad-hoc repository.

Module *Evaluator* helps the user assess the quality of time predictions on the test set, by leveraging two submodules: *A-FSM Evaluation* and *State Evaluation*, which compute a series of standard error metrics for an entire *A-FSM* model and for its individual states, respectively. The measures of all predictive models are gathered and eventually combined into global measures (described in Section 4.3), and arranged in a easily-readable report.

4.2 Application Scenario

Our approach has been validated on a real-life scenario, pertaining the handling of containers in a maritime terminal. There, a series of logistic activities are registered for each container passing through the harbor. Massive volumes of data are hence generated continually, which can profitably be exploited to analyze and improve the enactment of logistics processes. In particular, we consider only containers which both arrive and depart by sea, and focused on the different kinds of moves they undergo over the “yard”, i.e., the main area used in the harbor for storage purposes. This area is logically partitioned into a finite number of tri-dimensional slots, which are the units of storage space used for containers, and are organized in a fixed number of sectors

The lifecycle of any container can be roughly summarized as follows. The container is unloaded from a ship and temporarily placed near to the dock, until it is carried to some suitable yard slot for being stocked. Symmetrically, at boarding time, the container is first placed in a yard area close to the dock, and then loaded on a cargo. Different kinds of vehicles can be used for moving a

container, including, e.g., cranes, straddle-carriers (a vehicle capable of picking and carrying a container, by possibly lifting it up), and multi-trailers (a train-like vehicle that can transport many containers). This basic life cycle may be extended with additional transfers, classified as “house-keeping”, which are meant to make the container approach its final embark point or to leave room for other containers. More precisely, the following basic operations may be registered for any container: (i) *MOV*, when it is moved from a yard position to another by a straddle carrier; (ii) *DRB*, when it is moved from a yard position to another by a multi-trailer; (iii) *DRG*, when a multi-trailer moves to get it; (iv) *LOAD*, when it is charged on a multi-trailer; (v) *DIS*, when it is discharged off a multi-trailer; (vi) *SHF*, when it is moved upward or downward, possibly to switch its position with another container; (vii) *OUT*, when a dock crane embarks it on a ship.

In our experimentation, we focused on a subset of 5336 containers, namely the ones that completed their entire life cycle in the hub along the first four months of year 2006, and which were exchanged with four given ports around the Mediterranean sea. To translate these data into a process-oriented form, we viewed the transit of any container through the hub as a single enactment case of a (unknown) logistic process, where each log event refers to one of the basic operations above (i.e., *MOV*, *DRB*, *DRG*, *LOAD*, *DIS*, *SHF*, *OUT*) described above. Each of these operations hence acts as one activity of the reference logistics process.

Context Data. Several data attributes are available for each container (i.e., each process instance), which include, in particular, its origin and final destination ports, its previous and next calls, diverse characteristics of the ship that unloaded it, its physical features (e.g., size, weight), and a series of categorical attributes concerning its contents (e.g., the presence of dangerous or perishable goods). In addition to these internal properties of containers, some additional environmental features are associated with each container, which are meant to capture the context surrounding its arrival to the port. In particular, in our experimentation, we only considered two very basic environmental features: (i) a rough *workload* indicator, simply coinciding with the number of containers still in the port at time t_c , and (ii) a series of low-granularity time dimensions derived from the arrival time (namely, the hour, day of the week and month). Clearly, various additional context variables could be defined, in general, for a process instance (concerning, e.g., resource availability or refined workload indicators), possibly depending on the specific application domain. However, we leave this issue to future work. On the other hand, despite the narrow scope and simplistic nature of these feature, the benefits of using them to detect performance prediction scenarios were neat in our experimentation, as discussed later on.

4.3 Performance Measures and Evaluation Setting

With regard to the scenario above, we want to assess the quality of our approach in predicting the (remaining) time needed to completely process a container (i.e., until the *OUT* activity is performed on it). Knowing in advance such a metrics is of great value for harbor managers, in order to optimize the allocation of resources,

and to possibly prevent, for instance, incurring in violations of SLA (service level agreement) terms. In fact, certain typical SLAs establish that process enactments must not last more than a *Maximum Dwell Time (MDT)*; otherwise pecuniary penalties will be charged to the trans-shipment company. By the way, besides MDT, another important parameter for the scenario on hand is the *average dwell-time (ADT)*, i.e., the average sojourn time for containers in the terminal, which will be also used next for normalizing time measures.

Among the variety of metrics available in the literature, in order to assess the prediction accuracy of our models we resort (like in [13]) to the classic *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. In order to reduce the estimation bias, errors are measured according to a (10 fold) cross-validation procedure.

Formally, let us assume that $\tau \in \mathcal{P}(L')$ be a (possibly partial) trace in current test fold L' (amounting to 10% of L 's trace), and that $\hat{\mu}_{RT}(\tau)$ (resp., $\mu_{RT}(\tau)$) denote the actual (resp., predicted) remaining time for τ . Then the individual prediction errors associated with all the prefixes (i.e., partial enactments) of τ 's are measured as follows: (i) **mae** = $(1/|\mathcal{P}(L')|) \times \sum_{\tau \in \mathcal{P}(L')} |\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau)|$; (ii) **rmse** = $(\sum_{\tau \in \mathcal{P}(L')} (\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau))^2 / |\mathcal{P}(L')|)^{1/2}$; and (iii) **mape** = $(1/|\mathcal{P}(L')|) \times \sum_{\tau \in \mathcal{P}(L')} (|\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau)| / \hat{\mu}_{RT}(\tau))$.

In addition to the average prediction errors above (providing actual loss measures), we will also evaluate the capability of a CA-PPM to support the prediction of “overtime faults”, regarded as a specific form of SLA violations. To this end, let us denote by τ_c a trace encoding the full history of a container c , and $\tau_c(i)$ be its projection till some given step i . Then, an overtime fault for $\tau_c(i)$ is predicted based on the likelihood $\ell_{fault}(\tau_c(i))$ that the total time $\mu_{RT}(\tau_c(i))$, which will be eventually spent to fully handle c , does not exceed *MDT*. Precisely, letting $eTime(\tau_c(i))$ denote the time already elapsed for c from its arrival at the system, this likelihood is computed as follows:

$$\ell_{fault}(\tau_c) = \begin{cases} 1 - \frac{MDT}{eTime(\tau_c(i)) + \mu_{RT}(\tau_c(i))}, & \text{if } eTime(\tau_c(i)) + \mu_{RT}(\tau_c(i)) > MDT \\ 0, & \text{if } eTime(\tau_c(i)) + \mu_{RT}(\tau_c(i)) \leq MDT \end{cases}$$

For a suitably chosen risk tolerance threshold γ_{risk} , an alert is eventually triggered, while looking at the partial enactment $\tau_c(i)$, whenever $\ell_{fault}(\tau_c(i)) > \gamma_{risk}$, to notify the high risk of an incoming overtime fault – the greater the threshold, the lower sensitivity to the detection of potential overtime faults. Then, interpreting fault prediction as a classification problem with two given classes, i.e., **true** vs. **false** overtime faults, we can measure the prediction accuracy by computing the rates *FN* of False Negatives (i.e., overtime faults that were not deemed as such) and *FP* of False Positive (i.e., normal cases signaled as risky), as well as classical measures of *Precision* (i.e., $P = TP / (TP + FP)$), *Recall* (i.e., $R = TP / (TP + FN)$), with *TP* denoting the number of true positives, i.e., correctly predicted overtime faults. Incidentally, $\tau_c(i)$ is a **true** positive if $time(\tau(len(\tau))) > MDT$, and **true** negative otherwise.

5 Experiment Results

A series of tests were performed to assess the effectiveness and the efficiency of our approach in discovering a CA-PPM for remaining time prediction, based on the log described in the previous section. To this end, we tested our approach with various configurations of its parameters. In the following, we will report results obtained for different configurations of the two parameters associated with the abstraction function abs_h^{mode} : the horizon limit h , and the abstraction $mode \in \{list, bag\}$ – results with set-based abstractions are not shown here, due to their minor relevance, as discussed afterwards. Conversely, a fixed configuration is shown for threshold σ (namely, $\sigma = 0.4$), which was chosen pragmatically based on a series of specific tests, omitted here for space reasons.

All error results shown next were averaged over 10 trials, whereas their respective variance are not reported for the sake of brevity. Notice, however, that standard deviations were always lower than 5% of the average for all the metrics.

Qualitative Results. Before illustrating quantitative results in detail, let us show an example of one CA-PPM (when $abs_h = abs_4^{bag}$ and $\sigma = 0.4$) induced from the above log, in order to enable for a rough evaluation of the descriptive features of the model – even though its main goal is to offer operational support by means of performance predictions. In particular, the Figure 3 (a) reports, as a portion of the clustering function, the decision rule corresponding to one of the clusters found (namely, cluster 37), which actually corresponds to one of the leaves of the PCT model discovered with CLUS. This rule allows for easily interpreting the semantics of the cluster in terms of both container properties (namely, the origin port `PrevHarbor`, the destination port `NextHarbor`, the navigation line that is going to take it away `NavLine_OUT`, the navigation line bringing it in the current port `NavLine_IN`), and environmental context data (namely, the basic workload indicator `Workload`, based on instance counts, and aggregated time dimensions `ArrivalDay` or `ArrivalHour`). Despite its simplicity, the rule helps characterize a very peculiar, and yet relatively frequent scenario (the cluster gathers, indeed, 43 out of the 5336 traces) for the handling of containers.

As a matter of fact, the A-FSM model found for the same cluster (shown in Figure 3 (b)) witnesses that for this peculiar configuration of context factors (i.e.,

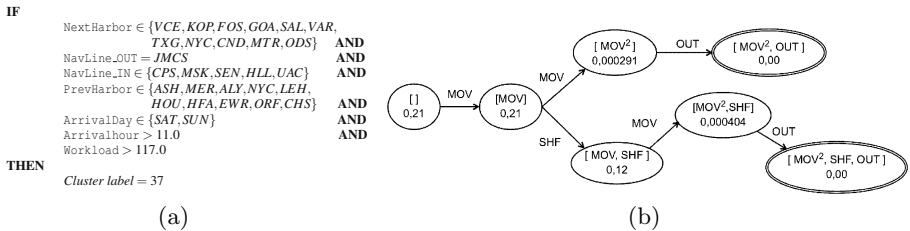


Fig. 3. Excerpt from a CA-PPM model for the harbor log, showing the (a) decision rule and (b) A-FMS model of one of the clusters found – (a) and (b) are a sort of (data-driven) descriptions for a context variant and its associated process variant, resp.

context variant), the containers tend to undergo a very small, and quite specific, paths over logistics operations. By the way, each node in the *A-FSM* is labelled with the bag of (the 4 more recent) operations leading to it – e.g., the node tagged with $[\text{MOV}^2, \text{OUT}]$ encodes all the traces in the cluster that undergo two MOVs before leaving the yard (operation **OUT**). Along with labels, each node also reports a constant prediction for the remaining time (normalized w.r.t. ADT). Edge labels codify, instead, which operations can trigger the corresponding node transition. For the sake of clarity, if a container is in the state labelled as $[\text{MOVE}, \text{SHF}]$ and a further MOV operation occurs, then the next state will be the one associated with $[\text{MOV}^2, \text{SHF}]$. Notably, this simple A-FMS model gave a neatly positive contribution to the accuracy of the global CA-PPM model – very low errors (namely, $rmse = 0.138$, $mae = 0.080$, and $mape = 0.302$) were produced, indeed, on the test traces that were assigned to it.

Time Prediction Effectiveness. Table 1 summarizes the errors made in predicting remaining times (normalized by the average dwell time ADT) for the case of $rmse$ and mae , using both our CA-TP plug-in and the prediction method proposed in [13] (here denoted by *FSM*, and also employed as a base learner in our approach). The tests were performed using different trace abstraction functions abs_h , and keeping fixed threshold $\sigma = 0.4$. For the sake of comparison, Table 2 also reports the percentage of error reduction ($\Delta\%$) obtained by CA-TP w.r.t. *FSM*. Moreover, the results of CA-TP are further differentiated according to which kinds of descriptive features were used. Specifically, CA-TP⁻ refers to the case where a CA-PPM is built only considering static container properties (e.g., dimensions, origin/destination ports). Conversely, CA-TP⁺ indicates the case where log traces are also associated with extrinsic context features (namely, workload indicators and seasonality dimensions), in addition to their primitive data attributes. These figures clearly show that our clustering-based method performs always better than the baseline, no matter of the parameter setting.

By a closer look, two factors appear to affect more the performances: the usage of derived context features and the value of history horizon h . In particular, the advantage of using environment-driven features is neat, despite they were very rough and partial, seeing as the average error reduction (computed over all error metrics) of CA-TP⁺ is close to 37%, whereas CA-TP⁻ “just” gets a 24% improvement. As to h , it is easily seen that, although the benefits of using our solution gets appreciable as soon as $h > 1$, the best performances are reached for $h = 4$, when all kinds of errors shrink more than 65% w.r.t. the baseline (see Table 2). Stretching the horizon beyond 8 seem to bring no further advantages (apart minor improvements for the $mape$ error with abs_8^{BAG}). This result is not surprising, seeing as accuracy achievements might even fall when using high values of h , due to the excessive level of detail on trace histories (and to the consequent high risk of overfitting).

The effect of the abstraction mode looks less marked, as very similar (good) results are found in both cases. Actually, whatever h and the kind of context features, less than 1% error reduction is obtained (on all metrics) when adopting bag abstractions, w.r.t. the case where lists were used. Finally, we notice that

Table 1. Average prediction errors (computed via a 10-fold cross-validation), for CA-TP and the baseline method (FSM), and different abstraction functions abs_h^{type} ($\sigma = 0.4$)

Parameters (abs_h^{mode})		FSM [13]			CA-TP ⁻			CA-TP ⁺		
mode	h	rmse	mae	mape	rmse	mae	mape	rmse	mae	mape
LIST	1	0.655	0.444	2.985	0.649	0.436	2.964	0.647	0.436	2.811
	2	0.465	0.211	0.516	0.335	0.102	0.376	0.335	0.095	0.355
	4	0.465	0.204	0.418	0.342	0.102	0.246	0.160	0.058	0.114
	8	0.465	0.204	0.407	0.349	0.102	0.175	0.164	0.058	0.107
	16	0.465	0.204	0.407	0.349	0.102	0.175	0.164	0.058	0.107
	Total		0.503	0.253	0.947	0.409	0.169	0.787	0.298	0.141
BAG	1	0.655	0.444	2.985	0.649	0.436	2.964	0.647	0.436	2.811
	2	0.473	0.218	0.560	0.342	0.109	0.404	0.342	0.102	0.375
	4	0.465	0.211	0.420	0.335	0.095	0.248	0.156	0.058	0.118
	8	0.465	0.211	0.420	0.342	0.095	0.170	0.156	0.058	0.107
	16	0.465	0.211	0.420	0.342	0.095	0.170	0.156	0.058	0.107
	Total		0.505	0.259	0.961	0.406	0.166	0.791	0.296	0.143
Grand Total		0.504	0.256	0.954	0.407	0.167	0.789	0.297	0.142	0.701

Table 2. Error reductions (%) – derived from Table 1 – achieved by CA-TP w.r.t. FSM

Parameters (abs_h^{mode})		CA-TP ⁻ ($\Delta\%$)			CA-TP ⁺ ($\Delta\%$)		
mode	h	rmse	mae	mape	rmse	mae	mape
LIST	1	-0.8%	-1.6%	-0.7%	-1.2%	-1.6%	-5.8%
	2	-28.1%	-51.7%	-27.2%	-28.1%	-55.2%	-31.3%
	4	-26.6%	-50.0%	-41.1%	-65.6%	-71.4%	-72.8%
	8	-25.0%	-50.0%	-57.0%	-64.8%	-71.4%	-73.8%
	16	-25.0%	-50.0%	-57.0%	-64.8%	-71.4%	-73.8%
	Total		-18.8%	-33.3%	-16.8%	-40.8%	-44.3%
BAG	1	-0.8%	-1.6%	-0.7%	-1.2%	-1.6%	-5.8%
	2	-27.7%	-50.0%	-27.8%	-27.7%	-53.8%	-33.0%
	4	-28.1%	-55.2%	-41.0%	-66.4%	-72.4%	-72.0%
	8	-26.6%	-55.2%	-59.6%	-66.4%	-72.4%	-74.5%
	16	-26.6%	-55.2%	-59.6%	-66.4%	-72.4%	-74.5%
	Total		-19.6%	-36.0%	-17.7%	-41.4%	-44.9%
Grand Total		-19.2%	-34.7%	-17.3%	-41.1%	-44.6%	-26.5%

poorer performances were obtained, in general, when using all methods with set-oriented trace abstraction functions (i.e., abs_h^{set}). However, since our approaches confirmed, even in such a case, similar degrees of improvement over the baseline, as those in Table 2, these results are not reported here for lack of space.

Fault Prediction Effectiveness. In general, the quality of overtime fault estimation is measured w.r.t. a given maximum dwell-time MDT, set in predefined agreements on service quality between the shipping lines and the terminal handler. In our tests we fixed $MDT = 2 \times ADT$ (namely, $MDT=11.46$ days).

Figure 4 sheds light on the ability our approach discriminate “over-time” from “in-time” containers. To this purpose, we report both Precision and Recall scores for different values of the risk threshold γ_{risk} , when a fixed, good-working, configuration of the underlying trace abstraction criterion (namely, $abs_h=abs_4^{BAG}$) is used for both our approach and the baseline one [13] (FSM), and $\sigma=0.4$ in our feature selection procedure. Notice that we only consider here the case where our tool (referred to as CA-TP⁺ in the figure) is provided with all kinds of (both intrinsic and extrinsic) context features available in the application scenario. As expected, recall tends to worsen when increasing γ_{risk} , while an opposite trend

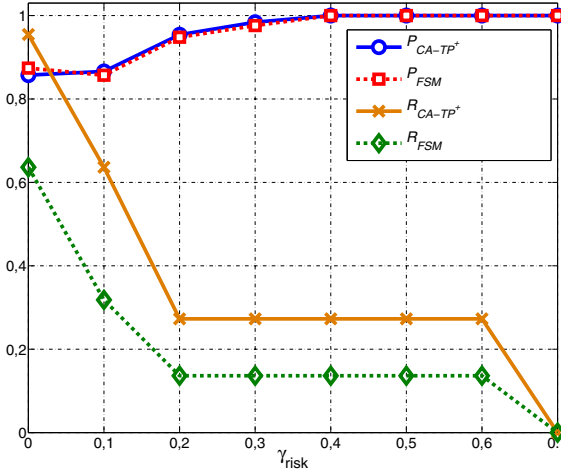


Fig. 4. Accuracy scores for the prediction of overtime faults by $CA-TP^+$ and by the baseline methods, when varying γ_{risk} , while fixing $\sigma = 0.4$, $h=4$, and abs_h^{bag}

is perceived for precision results. Interestingly, when using lower values of γ_{risk} (i.e., a more aggressive warning policy) the capability of our approach to recognize real overtime cases is compelling w.r.t. the baseline predictor – in particular, an astonishing recall of 0.95 (vs. 0.64) is reached with $\gamma_{risk}=0$. In general, recall scores are usually more important than precision ones in our scenario, since containers “stuck” in the yard implies high monetary costs, and if effectively recognizing them, suitable counter-measures could be undertaken – possibly resorting to the usage of additional (storage/processing) resources, which are not used in normal conditions for economical reasons. Clearly, such remedial policies as well come with a cost, even though it is typically far lower than SLA-violation penalties. Anyway, seeing as our method gets quite good precision scores over a wide range of γ_{risk} ’s values, it is reasonable to expect that a suitable trade-off can be reached, according to actual application requirements. More specifically, notice that the precision scores of the two methods are very similar for any value of γ_{risk} (in particular, our method never work significantly worse than the baseline one), and both flatten on 1 with $\gamma_{risk} = 0.4$.

Scalability Analysis. Table 3 shows the average computation times (in seconds) taken by $CA-TP^+$ and by the method in [13] for building a prediction model, as well as the number of clusters found in the first case (for completeness) – obviously, the second method does not perform any clustering of the log. Again, different abstraction methods abs_h^{mode} and a fixed value of σ were considered in the tests, which were all performed on a dedicated computer, equipped with an Intel dual-core processor and a 2GB (DDR2 1033 MHz) RAM, and running Windows XP Professional. For both methods, the real computation times are reported in the columns denoted by *Time*. Conversely, $Time_{par}$ corresponds to the time that would be spent in a virtual scenario, where an idealistic “overhead-free” parallelization of our approach is

Table 3. Number of clusters found by CA-TP⁺, and computation times for CA-TP⁺ and the baseline method, for different abstraction functions asb_h^{mode} and $\sigma = 0.4$

Parameters (asb_h^{mode})		CA-TP ⁺			FSM [13]	
mode	h	Cluster#	Time [sec]	Time _{par} [sec]	Time [sec]	
LIST	1	9	16.8	7.4	3.9	
	2	51.3	20.0	9.7	5.6	
	4	63.8	19.6	7.9	10.7	
	8	57.9	20.2	8.1	16.0	
	16	57.9	92.3	32.6	89.8	
	Total		46.8	32	13.1	25.2
BAG	1	9	17	7.3	4.0	
	2	50.7	19.7	9.6	5.5	
	4	64	18.7	7.6	8.4	
	8	57.9	19.8	8.0	10.6	
	16	57.9	79.0	36.0	32.3	
	Total		46.68	30.9	13.7	12.2
Grand Total		46.74	31.4	13.4	18.7	

used for concurrently learning the A -FSM models of all trace clusters. Although, as expected, our approach takes always longer times than the baseline method, the former achieves a satisfactory trade-off between effectiveness and efficiency. We are further comforted by the idealistic estimates $Time_{par}$, which let us be confident in the possibility of strengthen the scalability of our approach by resorting to a parallel implementation of it.

6 Conclusions

In this paper we have proposed an ad-hoc predictive clustering approach to the discovery of performance-oriented models, capable to provide performance forecasts at run time. Several innovative features distinguish our proposal from current literature. In particular, by automatically reckoning process variants with different performance patterns, prediction accuracy can be improved considerably, as witnessed by test results in the paper. Further, as the clustering model is represented via logical rules, the discovered process/context variants can be easily interpreted and validated. This makes our approach helpful in the ex-post analysis (revision, and consolidation) of tacit context-adaptation policies, and in the design of contextualized process models, capable to adapt to context changes. The methodology has been implemented as a plug-in in the ProM framework and validated on a real case study. Empirical results confirm the efficacy of the approach in predicting processing times, and in helping foresee SLA violations, as well as its scalability. As future work, we plan to investigate on making tighter the link between the clustering phase and the induction of cluster predictors, and on the usage of novel methods both for defining environment-related context variables, and for selecting performance-relevant space abstraction, as well as on combining our approach with other basic performance prediction methods (e.g., [6]), and adopting more refined models capable to capture concurrent behaviors more effectively. In particular, it is worth considering the possibility to automatically abstract and merge together similar states (e.g., by suitably

extending methods like those in [4]), in order to obtain more compact and generalized intra-cluster process models. Moreover, further efforts are needed in order to make the approach fully exploitable in practical application contexts. In particular, it would be beneficial to complement the prediction of SLA violations with explanations and suggestions about possible remedial actions.

References

1. CLUS: A predictive clustering system, <http://dtai.cs.kuleuven.be/clus/>
2. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2), 285–297 (1998)
3. Blockeel, H., Raedt, L.D., Ramon, J.: Top-down induction of clustering trees. In: *Proc. of 15th Intl. Conference on Machine Learning (ICML1998)*. pp. 55–63 (1998)
4. Caragea, C., Silvescu, A., Caragea, D., Honavar, V.: Abstraction augmented Markov models. In: *Proc. of 2010 IEEE Int. Conf. on Data Mining (ICDM 2010)*, pp. 68–77 (2010)
5. Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M.: History-Aware, Real-Time Risk Detection in Business Processes. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) *OTM 2011, Part I. LNCS*, vol. 7044, pp. 100–118. Springer, Heidelberg (2011)
6. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle Time Prediction: When Will This Case Finally Be Finished? In: Meersman, R., Tari, Z. (eds.) *OTM 2008, Part I. LNCS*, vol. 5331, pp. 319–336. Springer, Heidelberg (2008)
7. Folino, F., Greco, G., Guzzo, A., Pontieri, L.: Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. *Data & Knowledge Engineering* 70(12), 1005–1029 (2011)
8. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. *IEEE Trans. on Knowl. and Data Engineering* 18(8), 1010–1027 (2006)
9. 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)
10. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace Clustering in Process Mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) *BPM 2008 Workshops. LNBIP*, vol. 17, pp. 109–120. Springer, Heidelberg (2009)
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 & Knowledge Engineering* 47(2), 237–267 (2003)
12. 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., 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)
13. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. *Information Systems* 36(2), 450–475 (2011)
14. de la Vara, J.L., Ali, R., Dalpiaz, F., Sánchez, J., Giorgini, P.: *COMPRO: A Methodological Approach for Business Process Contextualisation*. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) *OTM 2010, Part I. LNCS*, vol. 6426, pp. 132–149. Springer, Heidelberg (2010)

On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery

Joos C.A.M. Buijs, Boudewijn F. van Dongen, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{j.c.a.m.buijs,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. Process discovery algorithms typically aim at discovering process models from event logs that best describe the recorded behavior. Often, the *quality of a process discovery algorithm* is measured by quantifying to what extent the resulting model can reproduce the behavior in the log, i.e. replay fitness. At the same time, there are many other metrics that compare a model with recorded behavior in terms of the precision of the model and the extent to which the model generalizes the behavior in the log. Furthermore, several metrics exist to measure the complexity of a model irrespective of the log.

In this paper, we show that existing process discovery algorithms typically consider at most two out of the four main quality dimensions: *replay fitness*, *precision*, *generalization* and *simplicity*. Moreover, existing approaches can not steer the discovery process based on user-defined weights for the four quality dimensions.

This paper also presents the *ETM algorithm* which allows the user to seamlessly steer the discovery process based on preferences with respect to the four quality dimensions. We show that all dimensions are important for process discovery. However, it only makes sense to consider precision, generalization and simplicity if the replay fitness is acceptable.

1 Introduction

The goal of *process mining* is to automatically produce process models that accurately describe processes by considering only an organization's records of its operational processes. Such records are typically captured in the form of *event logs*, consisting of cases and events related to these cases.

Over the last decade, many such process discovery techniques have been developed, producing process models in various forms, such as Petri nets, BPMN-models, EPCs, YAWL-models etc. Furthermore, many authors have compared these techniques by focussing on the properties of the models produced, while at the same time the applicability of various techniques have been compared in case-studies.

Figure 1 shows four quality dimensions generally used to discuss results of process discovery techniques, namely:

Replay Fitness. Replay fitness quantifies the extent to which the discovered model can accurately reproduce the cases recorded in the log. Typical algorithms guaranteeing perfect replay fitness are region-based approaches [7, 21]

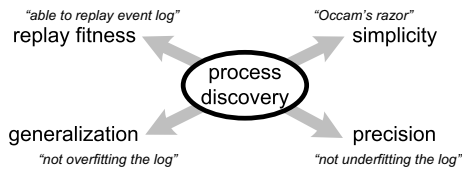


Fig. 1. Different quality dimensions for Process Model Discovery [3]

and the multi-phase miner [10]. Other techniques, such as the heuristics miner [19] and some genetic algorithms [14] use replay fitness as their guiding principle when discovering a process model, but do not guarantee optimal results.

Simplicity. The complexity of a process model is captured by the simplicity dimension. Process discovery algorithms often result in spaghetti-like process models [3], which are process models that are very hard to read. A class of process discovery algorithms that strongly focusses on simplicity is the class of α -algorithms [3, 11, 20], derived from the original α algorithm [5]. These discovery techniques generally result in simple models, but with poor replay fitness and/or precision.

Precision. It is trivial to discover a simple process model that can reproduce the event log. Such a model is generally referred to as the flower-model [16] and is an extreme example of an underfitting process model. A flower model is able to produce any arbitrary finite sequence of events. Therefore, precision quantifies the fraction of the behavior allowed by the model which is not seen in the event log. The region-based algorithms mentioned before [7, 21] are good examples of algorithms that guarantee optimal precision, i.e. they guarantee to allow only minimally more behavior than seen in the log.

Generalization. Finally, generalization assesses the extent to which the resulting model will be able to reproduce future behavior of the process. In that sense, generalization can also be seen as a measure for the confidence on the precision. Consider for example a very precise model that captures each individual case in the log as a separate path in the model. If there are many possible paths, it is unlikely that the next case will fit. Examples of generalizing algorithms are the fuzzy miner [13] and the heuristics miner [19].

The overview above shows that many process discovery algorithms focus on one or two of the dimensions. However, none of these algorithms is able to guide the result towards a particular dimension.

In this paper, we also present the ETM algorithm, which is a genetic algorithm able to optimize the process discovery result towards any of the four dimensions. By making use of so-called *process trees* [8] this algorithm ensures that the resulting model is a sound process model describing the observed log, while at the same time, the model is *optimal with respect to a weighted average over replay fitness, simplicity, precision and generalization*. Using the ETM algorithm, we

can easily explore the effects of focussing on one dimension in isolation and on combinations of these dimensions.

The remainder of this paper is structured as follows. In Section 2, we present our ETM algorithm. Furthermore, we present one metric for each of the four quality dimensions and we present process trees as a convenient means of modeling sound process models. In Section 3, we then present a running example which we use throughout the remainder of the paper. Using this example, we show the quality of various existing process discovery algorithms in terms of the presented metrics. Section 4 then shows the results of focussing on a subset of the quality dimensions during process discovery. Here, we use our ETM algorithm to show that such a narrow focus results in poor models. Section 5 shows the result when considering all dimensions. In Section 6 we apply existing techniques and our ETM algorithm on several real life event logs. Section 7 concludes the paper.

2 Process Trees and the ETM Algorithm

As stated in the introduction, we use the ETM algorithm to see the effects of (not) considering either of the four quality dimensions in process discovery. To this end, we first introduce process trees, which we use throughout the paper.

2.1 Process Trees

Traditional languages like BPMN, Petri nets, UML activity diagrams may be convenient ways of representing process models. However, only a small fraction of all possible models in these languages is *sound*, i.e. many models contain deadlocks, livelocks and other anomalies. Especially for the results presented in this paper, where the focus is on measuring the quality of the resulting models, it is essential that such unsound constructs are avoided. Therefore, we choose to use *process trees* to describe our models since all possible process trees represent sound process models.

Figure 2 shows the possible operators of a process tree and their translation to a Petri net. A process tree contains operator nodes and leaf nodes. Operator nodes specify the relation between its children. Possible operators are sequence (\rightarrow), parallel execution (\wedge), exclusive choice (\times), non-exclusive choice (\vee) and loop execution (\odot). The order of the children matters for the operators sequence and loop. The order of the children of a sequence operator specify the order in which they are executed (from left to right). For a loop, the left child is the ‘do’ part of the loop. After the execution of this ‘do’ part the right child, the ‘redo’ part, might be executed. After this execution the ‘do’ part is again enabled. The loop in Fig. 2 for instance is able to produce the traces $\langle A \rangle$, $\langle A, B, A \rangle$, $\langle A, B, A, B, A \rangle$ and so on.

Although also making use of a tree structure, a slightly different approach is taken by the Refined Process Structure Tree (RPST) [17]. The RPST approach provides “a modular technique of workflow graphs parsing to obtain fine-grained fragments with a single entry and single exit node” [17]. The content of these

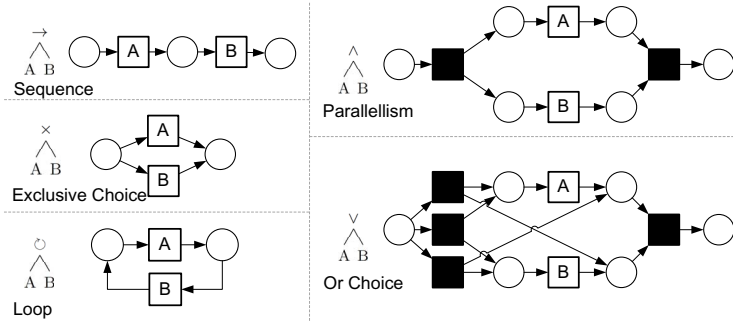


Fig. 2. Relation between process trees and block-structured Petri nets

fragments are graphs themselves and are not necessarily block-structured nor sound. Each operator in a process tree however results in a block structured process part with a single entry and single exit node. However, each block in a process tree can only contain a predefined control flow construct, which are shown in Figure 2. Therefore, workflow graphs decomposed into an RPST can be more expressive than a process tree but an RPST is not necessarily sound while a process tree always is.

2.2 Quality of Process Trees

To measure the quality of a process tree, we consider one metric for each of the four quality dimensions. We based these metrics on existing work in each of the four areas and we adopted them for process trees [1–3, 6, 9, 16]. We do not present the precise formulae that compute the values of these metrics here, as they do not matter for the results in this paper.

Replay Fitness quantifies the extent to which the model can reproduce the traces recorded in the log. We use an alignment-based fitness computation defined in [6] to compute the fitness of a process tree. Basically, this technique aligns as many events as possible from the trace with activities in an execution of the model (this results in a so-called *alignment*). If necessary, events are skipped, or activities are inserted without a corresponding event present in the log. Penalties are given for skipping and inserting activities. The final replay fitness score is calculated as follows:

$$Q_{rf} = 1 - \frac{\text{cost for aligning model and event log}}{\text{Minimal cost to align arbitrary event log on model and vice versa}}$$

where the denominator is the minimal costs when no match between event log and process model can take place (e.g. worst case scenario). This is used to normalize the replay fitness to a value between 0 and 1.

Simplicity quantifies the complexity of the model. Simplicity is measured by comparing the size of the tree with the number of activities in the log. This

is based on the finding that the size of a process model is the main factor for perceived complexity and introduction of errors in process models [15]. Furthermore, since we internally use binary trees, the number of leaves of the process tree has a direct influence on the number of operator nodes. Thus, if each activity is represented exactly once in the tree, that tree is considered to be as simple as possible. Therefore, simplicity is calculated as follows:

$$Q_s = 1 - \frac{\#duplicate\ activities + \#missing\ activities}{\#nodes\ in\ process\ tree + \#event\ classes\ in\ event\ log}$$

Precision compares the state space of the tree execution while replaying the log. Our metric is inspired by [1] and counts so-called escaping edges, i.e. decisions that are possible in the model, but never made in the log. If there are no escaping edges, the precision is perfect. We obtain the part of the statespace used from information provided by the replay fitness, where we ignore events that are in the log, but do not correspond to an activity according to the alignment. In short, we calculate the precision as follows:

$$Q_p = 1 - \frac{\sum_{visited\ markings} \#visits * \frac{\#outgoing\ edges - \#used\ edges}{\#outgoing\ edges}}{\#total\ marking\ visits\ over\ all\ markings}$$

Generalization considers the frequency with which each node in the tree needs to be visited if the model is to produce the given log. For this we use the alignment provided by the replay fitness. If a node is visited more often then we are more certain that its behavior is (in)correct. If some parts of the tree are very infrequently visited, generalization is bad. Therefore, generalization is calculated as follows:

$$Q_g = 1 - \frac{\sum_{nodes} (\sqrt{\#executions})^{-1}}{\#nodes\ in\ tree}$$

The four metrics above are computed on a scale from 0 to 1, where 1 is optimal. Replay fitness, simplicity and precision can reach 1 as optimal value. Generalization however can only reach 1 in the limit i.e., the more frequent the nodes are visited, the closer the value gets to 1. The flexibility required to find a process model that optimizes a weighted sum over the four metrics can efficiently be implemented using a genetic algorithm.

2.3 The ETM Algorithm

As discussed in Section 1 we propose the use of a genetic algorithm for the discovery of process models from event logs. Evolutionary algorithms have been applied to process mining discovery before in [4, 14]. Our approach follows the same high-level steps as most evolutionary algorithms [12], which are shown in Figure 3. The main improvements with respect to [4, 14] are the internal representation and the fitness calculations. By using a genetic algorithm for process discovery we gain flexibility: by changing the weights of different fitness factors we can guide the process discovery.

By using process trees as our internal representation we only consider sound process models. This drastically reduces the search space and therefore improves the performance of the genetic algorithm. Furthermore, we can apply standard tree change operations on the process trees to evolve them further. Finally, in our fitness calculation we consider *all four quality dimensions* for process models: replay fitness, precision, generalization and simplicity. The user can specify the relative importance of each dimension *beforehand*. The *ETM algorithm* (which stands for *Evolutionary Tree Miner*) will then favor those candidates that have the correct mix of the different quality dimensions.

In general, our genetic algorithm follows the process as shown in Fig. 3. The input of the algorithm is an event log describing observed behavior. In the initial step a population of random process trees is generated where each activity occurs exactly once in each tree. Next the four quality dimensions are calculated for each candidate in the population. Using the weight given to each dimension the *overall fitness* of the process tree is calculated. In the next step certain stop criteria are tested such as finding a tree with the desired overall fitness. If none of the stop criteria are satisfied, the candidates in the population are changed by swapping subtrees between them (crossover) or by changing, adding or removing nodes (mutation). After these changes the fitness is again calculated. This is continued until at least one stop criterion is satisfied and the fittest candidate is then returned.

Our genetic algorithm has been implemented as a plug-in for the ProM framework [18]. We used this implementation for all experiments presented in the remainder. The algorithm stops as soon as a perfect candidate was found, i.e. with optimal fitness, or after 1.000 generations. In [8] we have shown that 1.000 generations are typically enough to find the optimal solution, especially for processes with few activities. All other settings were selected according to the optimal values presented in [8].

3 Running Example

Throughout the paper, we use a running example, describing a simple loan application process of a financial institute, providing small consumer credit through a webpage. When a potential customer fills in a form and submits the request on the website, the process is started by activity A which is sending an e-mail

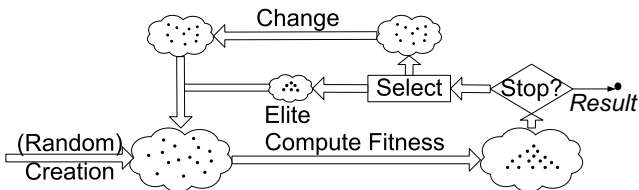


Fig. 3. The different phases of the genetic algorithm

Table 1. The event log

Trace	#	Trace	#
A B C D E G	6	A D B C F G	1
A B C D F G	38	A D B C E G	1
A B D C E G	12	A D C B F G	4
A B D C F G	26	A C D B F G	2
A B C F G	8	A C B F G	1
A C B E G	1		

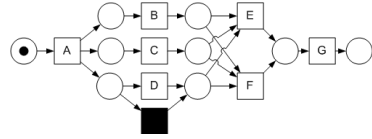


Fig. 4. Petri net of a loan application process. (A = send e-mail, B = check credit, C = calculate capacity, D = check system, E = accept, F = reject, G = send e-mail).

to the applicant to confirm the receipt of the request. Next, three activities are executed in parallel. Activity B is a check of the customer’s credit history with a registration agency. Activity C is a computation of the customer’s loan capacity and activity D is a check whether the customer is already in the system. This check is skipped if the customer filled in the application while being logged in to the personal page, since then it is obsolete. After performing some computations, the customer is notified whether the loan was accepted (activity E, covering about 20% of the cases) or rejected (activity F, covering about 80% of the cases). Finally, activity G is performed, notifying the applicant of the outcome.

A Petri net of the loan application model is shown in Figure 4 and the log we obtained through simulation is shown in Table 1.

3.1 Results of Process Discovery Algorithms

In order to validate that our small example provides enough of a challenge for existing process discovery techniques, we applied several existing techniques, many of which resulted in *unsound* process models. We translated the behavior of each model to a process tree, in order to measure the quality of the result. Where applicable, we stayed as close as possible to the parallel behavior of the original model. Figures 5 to 11 show the results of the various algorithms.

Figures 5 to 11 clearly indicate that, on our small example, only the α -algorithm was able to balance the four quality dimensions well. Several algorithms even produce an unsound result. Moreover, the α -algorithm was “lucky” for this small example. In general, this algorithm produces models that are not fitting or not precise. Therefore, in Section 4, we first investigate combining various dimensions and show that *all of them have to be considered in order to discover a sound, easy to understand process model, accurately describing the log*

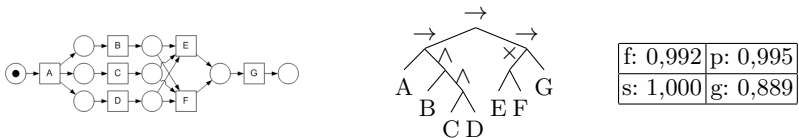


Fig. 5. Result of the α algorithm [5] (sound)

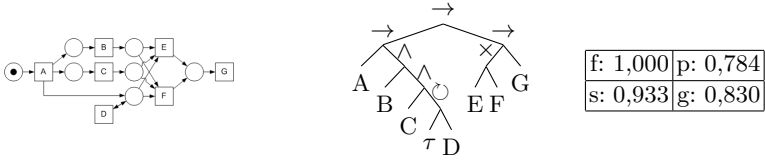


Fig. 6. Result of the ILP miner [21] (Ensuring empty net after completion, sound)

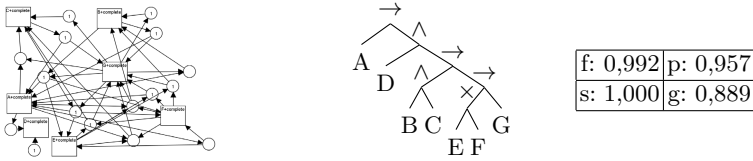


Fig. 7. Result of the language-based region theory [7] (The model is overly complex and incomprehensible, but sound)

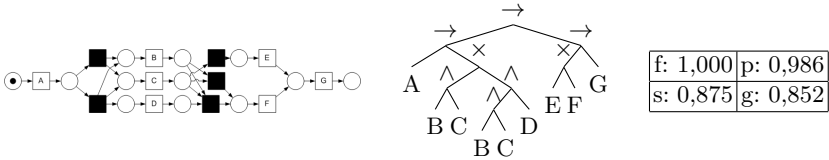


Fig. 8. Result of the heuristic miner [19] (Unsound, tokens are left behind.)

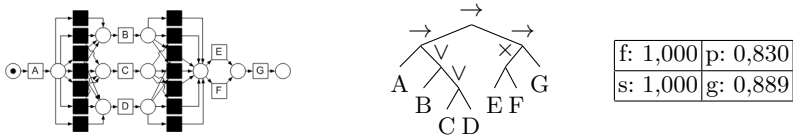


Fig. 9. Result of the Multi-phase miner [10] (Model is guaranteed “relaxed sound” and the tree reflects this.)

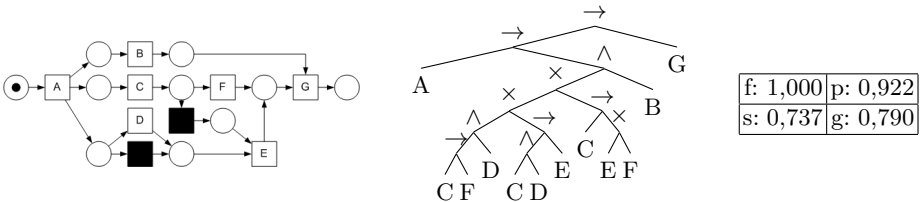


Fig. 10. Result of the genetic miner [14] (Unsound, tokens left behind.)

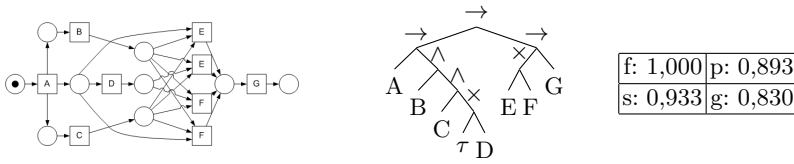


Fig. 11. Result of the state-based region theory [21]

under consideration. Next, in Section 6, we show that only our ETM algorithm is able to balance all quality dimensions for real life event logs.

4 Ignoring Quality Dimensions

The examples shown in figures 5 to 11 show that various existing process mining techniques perform differently on all four quality dimensions and they often provide unsound models. In this section, we use the ETM algorithm to discover a process model on the given log, while varying which of the quality dimensions should be considered. We show some optimal models that resulted from different runs of the algorithm and discuss their properties.

4.1 Considering Only One Quality Dimension

Fig. 12a shows an example process tree that was discovered when focussing solely on the *replay fitness* dimension. Although the tree is able to replay the event log perfectly, the tree allows for more behavior than is seen in the event log. Since adding parts to the process tree might improve replay fitness, and removing parts never does, the process tree will keep growing until perfect replay fitness is reached. This is bad for simplicity since activities will be duplicated (activities B and D in Fig. 12a) and certain parts of the tree will never be used (the rightmost B and the leftmost D are never used when replaying the event log).

In order to obtain trees that do not allow for behavior that is not observed in the event log, we considered only the *precision* dimension in the genetic algorithm. An example of such a tree is shown in Fig. 12b, which has a perfect precision because it can only generate the trace $\langle C, B, C \rangle$. A process tree will have perfect precision if each trace it can generate is used in an alignment. Since the tree of Fig. 12b can only generate one trace, each alignment between event log and the tree, will use this path of execution. However, the low replay fitness score indicates that the tree in Fig. 12b has little to do with the behavior that is recorded in the event log.

When only considering *simplicity*, we get trees such as the one in Fig. 12c, where each activity is included exactly once. However, the tree does not really describe the observed process executions in the event log well, which is indicated by the low scores on replay fitness and precision.

Generalization measures the likeliness that a model contains future, not yet observed behavior. When focussing solely on generalization, Fig. 12d shows the

process tree that has the best generalization score. As mentioned before, generalization cannot reach 1, as this would require all possible behavior to be observed infinitely often. Since generalization takes the number of node visits into account, the score is improved if nodes are visited more often, where the visits are again measured on the closest matching execution of the model for each trace. By placing \circlearrowleft operators high in the tree, and activities F and B in the ‘redo’ part, the loops and the nodes in the ‘do’ part are executed more often, hence improving generalization.

4.2 Always Considering Replay Fitness

The discussion in the previous section showed that none of the quality dimensions should be considered in isolation. Furthermore, we validated the choice of many existing process discovery techniques to put emphasis on replay fitness, i.e. if the replay fitness is not good enough, the other quality dimensions add little value as the discovered model does not describe the recorded behavior. On the other hand, achieving a perfect replay fitness is not always necessary or desired.

When focussing on *replay fitness and precision*, the goal is to find a process model that describes all traces, and not much more, much like the region-based algorithms the results of which are depicted in Figure 6, 7 and 11. In general, a model that contains an initial exclusive choice between all unique traces in the log has perfect precision and replay fitness. Each choice is taken at least once and each trace in the event log is a sequence in the process model. This always results in a perfect replay fitness. For our running example the process tree as shown in Fig. 13a also has both a perfect replay fitness and precision. Each part of the process tree is used to replay a trace in the event log and no behavior that is not present in the event log can be produced by the process tree. However,

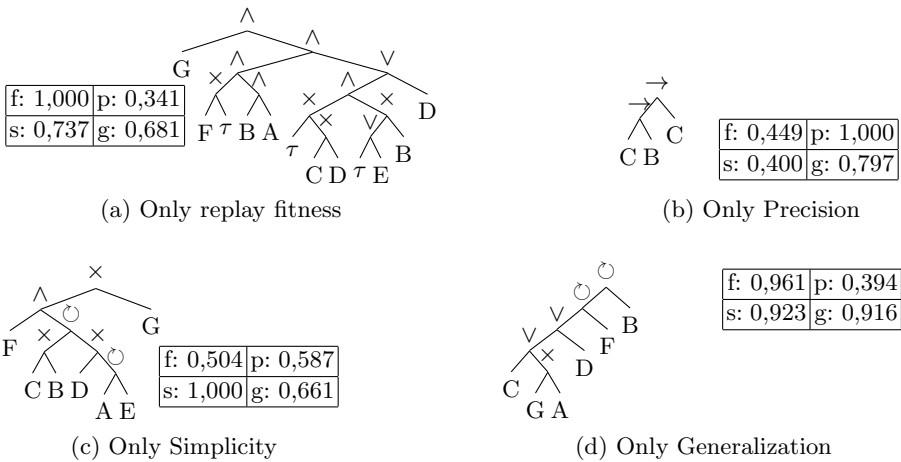


Fig. 12. Process trees discovered when considering each of the quality dimensions separately

since the tree is fairly big, the simplicity score is low and more importantly, the generalization is not very high either. This implies that, although this model is very precise, it is not likely that it explains any future, unseen behavior.

Next we consider *replay fitness and simplicity*, the result of which is shown in Fig. 13b. When considering only replay fitness, we obtained fairly large models, while simplicity should keep the models compact. The process tree shown in Fig. 13b contains each activity exactly once and hence has perfect simplicity. At the same time all traces in the event log can be replayed. However, the process tree contains two \wedge , one \circ and three \vee nodes that allow for (far) more behavior than is seen in the event log. This is reflected in the low precision score in combination with the high generalization.

The process tree that is found when focussing on the combination of *replay fitness and generalization* is shown in Fig. 13c. The process tree shows many similarities with the process tree found when solely considering generalization. Activity E has been added to the ‘do’ part of the \circ to improve the replay fitness. However, it also reduces the generalization dimension since it is only executed 20 times. Furthermore, the tree is still not very precise.

In contrast to the trees in Section 4.1, the various process trees discussed in this section mainly capture the behavior observed in the event log. However, they either are overfitting (i.e. they are too specific) or they are underfitting (i.e. they are too generic). Hence, considering replay fitness in conjunction with only one other dimension still does not yield satisfying results. Therefore, in Section 4.3, we consider three out of four dimensions, while never ignoring replay fitness.

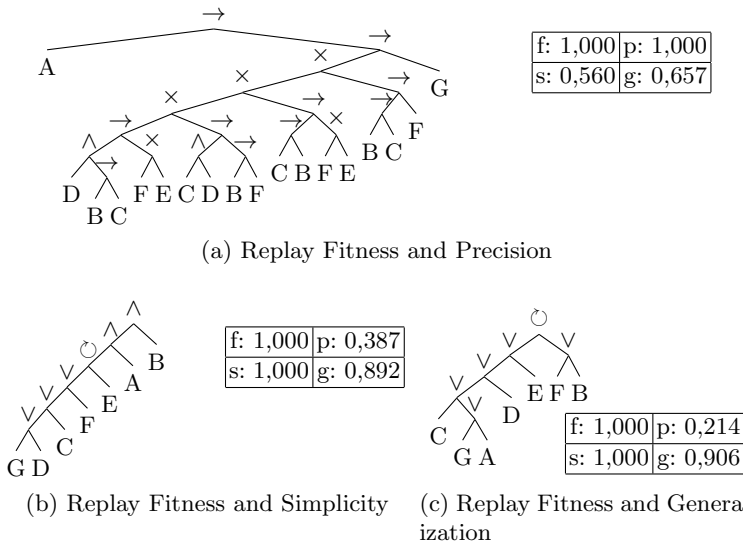


Fig. 13. Process trees discovered when considering replay fitness and one of the other quality dimensions

4.3 Ignoring One Dimension

We just showed that replay fitness, in conjunction with one of the other quality dimensions, is insufficient to judge the quality of a process model. However, most process discovery algorithms steer on just two quality dimensions. Hence we first consider 3-out-of-4 dimensions.

Fig. 14 shows the three process trees that are discovered when ignoring one of the quality dimensions, but always including replay fitness. Fig. 14a shows the process tree found when ignoring precision. The resulting process tree is similar to the one in Fig. 13c, which was based on replay fitness and generalization only. The only difference is that the parent of A and G has changed from \vee to \times . Since this only influences precision, the other metrics have the same values and both trees have the same overall fitness when ignoring precision.

The process tree which is discovered when ignoring generalization is the same as when simplicity is ignored and is shown in Fig. 14b. This is due to the fact that both simplicity and generalization are optimal in this tree. In other words, when weighing all four dimensions equally, this tree is the best possible process tree to describe the process.

Interestingly, this tree is the same as the result of the α -algorithm (Fig. 5). However, as mentioned earlier, the α -algorithm is not very robust. This will also be demonstrated in Section 6 using real life event logs.

5 Weighing Dimensions

The process trees shown in Fig. 14 have trouble replaying all traces from the event log while maintaining a high precision. However, since process discovery is mostly used to gain insights into the behavior recorded in the log, it is generally required that the produced model represents the log as accurately as possible, i.e. that both replay fitness and precision are high. By giving more weight to replay fitness, while still taking precision into account, our genetic algorithm can accommodate this importance. Fig. 15 shows the process tree resulting from our algorithm when giving 10 times more weight to replay fitness than the other three quality dimensions. As a result the process tree is able to replay all traces from the event log while still maintaining a high precision.

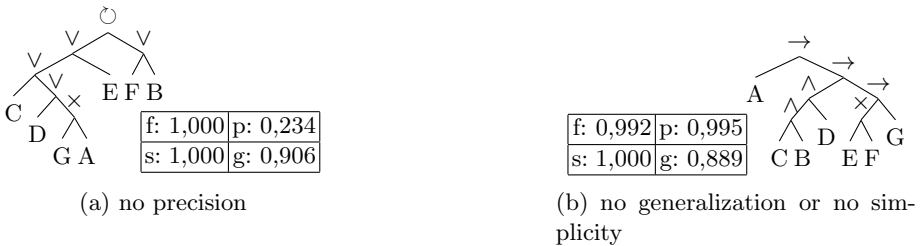


Fig. 14. Considering 3 of the 4 quality dimensions

Let us compare this process tree with the process tree of Fig. 14b, which is also the tree produced when all quality dimensions are weighted equally. It can be seen that the price to pay for improving fitness was a reduction in precision. This can be explained by looking at the change made to the process model: activity D is now in an \vee relation with activities B and C. Replay fitness is hereby improved since the option to skip activity D is introduced. However, the process tree now also allows for skipping the execution of both B and C. Something which is never observed in the event log.

Furthermore, the process tree of Figure 15 performs better than the model we originally used for simulating the event log as can be seen in Figure 16. The original tree performs equal on replay fitness but worse on the other three quality dimensions. Precision is worse because the state space of the original model is bigger while less paths are used. Simplicity is also worse because an additional τ node is used in the original tree, hence the tree is two nodes bigger than optimal. Furthermore, since the τ node is only executed ten times, the generalization reduces as well because the other nodes are executed more than 10 times, thus the average visits per node decreases.

6 Experiments Using Real Life Event Logs

In the previous sections we discussed the results of various existing process discovery techniques on our running example. We also demonstrated that all four quality dimensions should be considered when discovering a process model. In this section we apply a selection of process discovery techniques, and our ETM algorithm, on 3 event logs from real information systems. Using these event logs, and the running example, we show that our ETM is more robust than existing process discovery techniques.

In this section we consider the following event logs:

1. The event log **L0** is the event log as presented in Table 1. L0 contains 100 traces, 590 events and 7 activities.
2. **Event Log L1** contains 105 traces, 743 events in total, with 6 different activities.
3. **Event Log L2** contains 444 traces, 3.269 events in total, with 6 different activities.

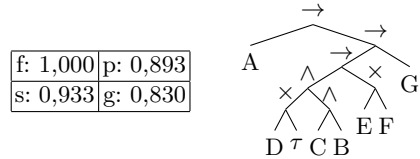
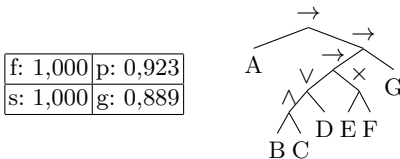


Fig. 15. Process tree discovered when replay fitness is 10 times more important than all other dimensions

Fig. 16. Process tree of the model used for simulation (Translated manually from Figure 4)

Table 2. Petri Net properties of discovered models.**Legend:** s?: whether the model is sound (✓) or unsound (✗);

#p: number of places; #t: number of transitions; #arcs: number of arcs.

	L0				L1				L2				L3			
	s?	#p	#t	#arcs	s?	#p	#t	#arcs	s?	#p	#t	#arcs	s?	#p	#t	#arcs
α -algorithm	✓	9	7	20	✓	3	6	4	✓	3	6	4	✓	6	6	10
ILP Miner	✓	7	7	19	✓	4	6	9	✓	2	6	11	✓	4	6	9
Heuristics	✗	12	12	30	✓	12	15	28	✓	12	16	32	✗	10	11	23
Genetic	✗	10	9	21	✗	13	20	42	✗	11	20	36	✗	10	11	25

4. **Event Log L3** contains 274 traces, 1.582 events in total, with 6 different activities.

Event logs L1, L2 and L3 are extracted from information systems of municipalities participating in the CoSeLoG¹ project. Since some of the existing process discovery techniques require a unique start and end activity, all event logs have been filtered to contain only those traces that start with the most common start activity and end with the most common end activity. Furthermore, activity names have been renamed to the letters A...F.

From the process discovery algorithms discussed in Section 3.1 we selected four well-known algorithms: the α -algorithm [5], the ILP-miner [21], the heuristics miner [19] and the genetic algorithm by Alves de Medeiros [14]. Because we do not have enough space to show all process models we show some important characteristics of the resulting Petri nets in Table 2.

The α -algorithm and ILP Miner produce sound Petri nets for each of the 4 input logs. The Genetic Miner however never produces a sound Petri net and the Heuristics Miner produces a sound solution for 2 out of the 4 event logs.

For each of the discovered Petri nets we created process tree representations, describing the same behavior. If a Petri net was unsound, we interpreted the sound behavior as closely as possible. For each of these process trees the evaluation of each of the 4 metrics, and the overall average fitness, is shown in Table 3.

For event log L1 both the α -algorithm and the ILP miner find process models that can replay all behavior. But, as is also indicated by the low precision, these allow for far more behavior than observed in the event log. This is caused by transitions without input places that can occur an arbitrary number of times. The heuristics miner is able to find a reasonably fitting process model, although it is also not very precise since it contains several loops. The genetic algorithm finds a model similar to that of the heuristics miner, although it is unsound and contains even more loops. The ETM algorithm finds a process tree, which is shown in Figure 17a, that scores high on all dimensions. If we want to improve replay fitness even more we can make it 10 times more important as the other

¹ See <http://www.win.tue.nl/coselog>

Table 3. Quality of Process Tree translations of Several Discovery Algorithms (*italic* results indicate unsound models, the best model is indicated in **bold**)

	L0		L1		L2		L3	
α -algorithm	f: 0,992	p: 0,995	f: 1,000	p: 0.510	f: 1.000	p: 0.468	f: 0.976	p: 0.532
	s: 1,000	g: 0,889	s: 0.923	g: 0.842	s: 0.923	g: 0.885	s: 0.923	g: 0.866
	overall: 0,969		overall: 0,819		overall: 0,819		overall: 0,824	
ILP Miner	f: 1,000	p: 0,748	f: 1.000	p: 0.551	f: 1.000	p: 0.752	f: 1.000	p: 0.479
	s: 0,933	g: 0,830	s: 0.857	g: 0.775	s: 0.923	g: 0.885	s: 0.857	g: 0.813
	overall: 0,887		overall: 0,796		overall: 0,890		overall: 0,787	
Heuristics	<i>f: 1,000</i>	<i>p: 0,986</i>	f: 0.966	p: 0.859	f: 0.917	p: 0.974	<i>f: 0.995</i>	<i>p: 1.000</i>
	<i>s: 0,875</i>	<i>g: 0,852</i>	s: 0.750	g: 0.746	s: 0.706	g: 0.716	<i>s: 1.000</i>	<i>g: 0.939</i>
	<i>overall: 0,928</i>		overall 0,830		overall: 0,828		overall: 0,983	
Genetic	<i>f: 1,000</i>	<i>p: 0,922</i>	<i>f: 0,997</i>	<i>p: 0,808</i>	<i>f: 0.905</i>	<i>p: 0.808</i>	<i>f: 0.987</i>	<i>p: 0.875</i>
	<i>s: 0,737</i>	<i>g: 0,790</i>	<i>s: 0,750</i>	<i>g: 0.707</i>	<i>s: 0,706</i>	<i>g: 0.717</i>	<i>s: 0.750</i>	<i>g: 0.591</i>
	<i>overall: 0,862</i>		<i>overall: 0,815</i>		<i>overall: 0,784</i>		<i>overall: 0,801</i>	
ETM	f: 0,992	p: 0,995	f: 0,901	p: 0,989	f: 0,863	p: 0,982	f: 0,995	p: 1,000
	s: 1,000	g: 0,889	s: 0,923	g: 0,894	s: 0,923	g: 0,947	s: 1,000	g: 0,939
	overall: 0,969		overall: 0,927		overall: 0,929		overall: 0,983	

quality dimensions. This results in the process tree as shown in Figure 17b. With an overall (unweighed) fitness of 0,884 it is better than all process models found by other algorithms while at the same time having a perfect replay fitness.

Event log L2 shows similar results: the α -algorithm and the ILP miner are able to find process models that can replay all behavior but allow for far more behavior. The heuristics miner and genetic miner again found models with several loops. The ETM algorithm was able to find a tree, which is shown in Figure 18a, that scores high on all dimensions but less so on replay fitness. If we emphasize replay fitness 10 times more than the other dimensions, we get the process tree as is shown in Figure 18b. Although replay fitness improved significantly, the other dimensions, especially precision and simplicity, are reduced.

For event log L3 the observations for the last two event logs still hold. Both the α -algorithm and the ILP miner provide fitting process models that allow for far more behavior. Both the heuristics miner and the genetic algorithm result in unsound models. However, the sound interpretation of the heuristics model is the same as the sound process tree found by the ETM algorithm. Although replay fitness is almost perfect, we can let the ETM algorithm discover a process tree with real perfect replay fitness. This requires making it 1000 times more important than the others and results in a process tree that has perfect replay fitness but scores bad on precision. However, as we have seen before, this is a

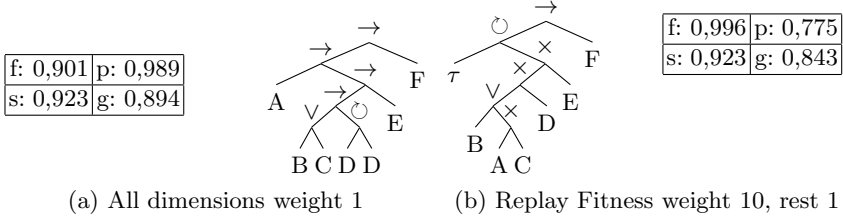


Fig. 17. Process Trees discovered for L1

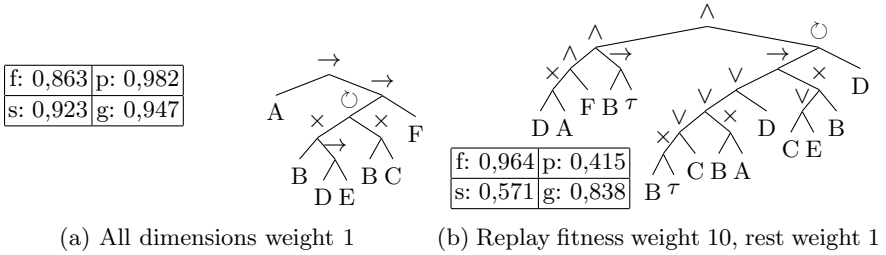


Fig. 18. Process Trees discovered for L2

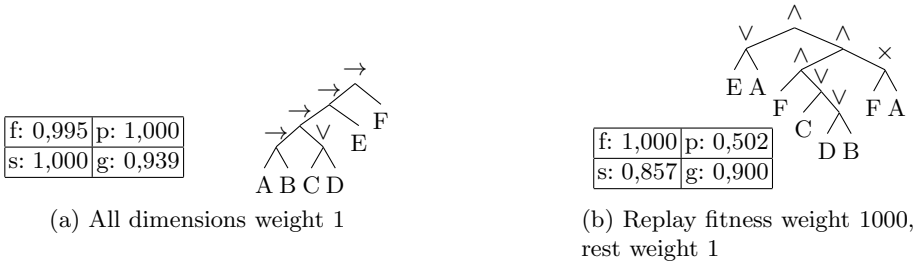


Fig. 19. Process Trees discovered for L3

common trade-off and the process tree is still more precise than the one found by the ILP miner which also has a perfect replay fitness.

Investigating the results shown in Table 3 we see that on two occasions a process model similar to the one found by the ETM algorithm was found by another algorithm. However, the α -algorithm was not able to produce sensible models for any of the three real life event logs. The heuristics miner once produced a process model of which the sound behavior matched the process tree the ETM algorithm discovered. However, our algorithm always produced sound process models superior to the others. Furthermore, the ETM algorithm can be steered to improve certain dimensions of the process model as desired.

7 Conclusion

The quality of process discovery algorithms is generally measured using four dimensions, namely replay fitness, precision, generalization and simplicity. Many existing process discovery algorithms focus on only two or three of these dimensions and generally, they do not allow for any parameters indicating to what extent they should focus on any of these dimensions.

In this paper, we presented the ETM algorithm to discover process trees on a log. This ETM algorithm *can be configured* to optimize for a weighted average over the four quality dimension, i.e. a model can be discovered that is optimal given the weights given to each parameter. Furthermore, the ETM algorithm is *guaranteed to produce sound process models*.

We used our ETM algorithm to show that *all four quality dimensions are necessary* when doing process discovery and that none of them should be left out. However, the fitness dimension, indicating to what extent the model can reproduce the traces in the log, is more important than the other dimensions.

Using both an illustrative example and three real life event logs we demonstrated the need to consider all four quality dimensions. Moreover, our algorithm is able to balance all four dimensions in a seamless manner.

References

1. Carmona, J., van Dongen, B.F., van der Aalst, W.M.P., Adriansyah, A., Munoz-Gama, J.: Alignment Based Precision Checking. BPM Center Report BPM-12-10. BPMcenter.org (2012)
2. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2(2), 182–192 (2012)
3. van der Aalst, W.M.P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, Berlin (2011)
4. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic Process Mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005)
5. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering* 16(9), 1128–1142 (2004)
6. Adriansyah, A., van Dongen, B., van der Aalst, W.M.P.: Conformance Checking using Cost-Based Fitness Analysis. In: *IEEE International Enterprise Computing Conference (EDOC 2011)*, pp. 55–64. IEEE Computer Society (2011)
7. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
8. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A Genetic Algorithm for Discovering Process Trees. In: *Proceedings of the 2012 IEEE World Congress on Computational Intelligence*. IEEE (to appear, 2012)
9. Calders, T., Günther, C.W., Pechenizkiy, M., Rozinat, A.: Using minimum description length for process mining. In: *Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009*, pp. 1451–1455. ACM, New York (2009)

10. van Dongen, B.F.: Process Mining and Verification. Phd thesis, Eindhoven University of Technology (2007)
11. van Dongen, B.F., Alves de Medeiros, A.K., Wen, L.: Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 225–242. Springer, Heidelberg (2009)
12. Eiben, A.E., Smith, J.E.: Introduction to evolutionary computing. Springer (2003)
13. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
14. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery* 14(2), 245–304 (2007)
15. Mendling, J., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P., Neumann, G.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data and Knowledge Engineering* 64(1), 312–329 (2008)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems* 33(1), 64–95 (2008)
17. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. *Data and Knowledge Engineering* 68(9), 793–818 (2009)
18. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
19. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering* 10(2), 151–162 (2003)
20. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining Process Models with Non-Free-Choice Constructs. *Data Mining and Knowledge Discovery* 15(2), 145–180 (2007)
21. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. *Fundamenta Informaticae* 94, 387–412 (2010)

ECO: Event Detection from Click-through Data via Query Clustering

Prabhu K. Angajala, Sanjay K. Madria, and Mark Linderman

Department of Computer Science, Missouri University of Science and Technology, MO, USA
Air Force Research Lab, Rome, NY, USA
madrias@mst.edu

Abstract. In this paper, we propose an algorithm to detect real world events from the click-through data. Our approach differs from the existing work as we: (i) consider the click-through data as collaborative query sessions instead of mere web logs proposed by many others (ii) integrate the semantics, structure, and content of queries and pages, and (iii) aim to achieve the overall objective via query clustering. The problem of event detection is transformed into query clustering by generating clusters using hybrid cover graphs where each hybrid cover graph corresponds to a real-world event. The evolutionary pattern for the co-occurrence of query-page pairs in a hybrid cover graph is imposed over a moving window period. Finally, we experimentally evaluated our proposed approach using a commercial search engine's data collected over 3 months with about 20 million web queries and page clicks from 650,000 users. Our method outperforms the most recent event detection work proposed using complex methods in terms of metrics such as number of events detected, F-measures, entropy, recall etc.

1 Introduction

The approximate size of today's indexed World Wide Web is at least 45.93 billion pages as per existing estimation [1] and is a rich collection of all the real world objects. Web is a great source of knowledge to be mined to learn about topics, stories, events etc. Event detection is becoming increasingly popular because of its applicability in several diversified areas. Therefore, the interpretation of "event" definition is context-dependent. An event can be associated with reporting how many people/cars have entered a building/freeway, web access logs, security logs, object trajectory in video surveillance and business activity monitoring for business intelligence etc. In our perspective and from the viewpoint of Web, an event can be understood as some real-world activity that stirs a large scale querying and browsing activity. That is, it is of more interest to users over a sizable window period which is unusual relative to normal patterns of querying and browsing behavior.

Web is the collaborative work of many people; a few publishing, and all of them querying and retrieving the information. Search engines record these activities in Web logs called the click-through data and reflect the query and clicks activities of users. Click-through data is more or less in the format shown in the Table 1 below:

Table 1. Sample click-through data

AnonID	Query	Query Time	Item Rank	Click URL
7	Easter	2006-03-01 23:19:52	1	http://www.happy-easter.com
7	Easter eggs	2006-03-01 23:19:58	1	http://www.eeggs.com

To briefly explain the fields in Table 1, we begin with AnonID, the anonymous User ID from whom the search engine received the request, followed by the query issued by the user, the time the search engine received the request, the rank of the page item clicked, the page clicked in response to the result among the set returned by the search engine. Note that the click-through data format varies slightly from one search engine to the other.

Three Web data types identified in previous [2] efforts are: content (text and multimedia), structure (links that form a graph) and Web usage (transactions from Web log). Web data mining encompasses a broad range of research topics like improving page ranking, efficient indexing, query clustering, query similarity, query suggestions, extracting semantic relations and event detection etc. All these areas are inter-related and many use the click-through data as a starting point for their analysis. The seamless flow of advancement in developing better approaches in individual areas can be pipelined to improve existing techniques in other inter-related fields. Our effort in this paper is to integrate three Web data types and achieve the overall objective of event detection via query clustering.

1.1 Motivation

The dynamics of click-through data was previously identified in [3]. The frequency of queries and page clicks grow very fast when the real-world event approaches and become weaker gradually after. The co-occurrence of a query-page pair in a given window period is the number of times the pair appear together in the same row of Table 1 in that window period. The dynamics of co-occurrences can sense the arrival and pass over of the events. The work done by Greg [17] et al. gave an inside out perspective about the query space, query sessions, user behavior and content space. Interesting facts were revealed like about 28% of all queries are reformulations of previous queries; an average query is reformulated 2.6 times; users formulate and reformulate a series of queries in pursuit of a single task. The possibilities are new queries, add/remove word(s) to/from queries, change word(s) in a query, get more results for the same query, return to a previous queries etc. Our motivation is to cluster such similar queries with similar evolutionary pattern corresponding to real world events.

Our work differs from the existing work in one or more of the following ways:

(1) We consider the click-through data as collaborative query sessions (a time interval which consists of all the query-page pairs from all users within this time interval) rather than collection of individual entries of query-page pairs considered in [3,4]. A query session captures a series of user interactions with the search engine

with begin and end time period. The advantage of this approach is that in most of the meaningful sessions users issue a series of related queries and click through the web pages in the result set. They are semantically and temporally related to one another. These meaningful query sessions as initial clusters can correspond to real world events. User intentions are better understood by considering click-through data as query sessions. Search engines click-through data is massive and the graphs generated from the click-through data are overwhelmingly large. By considering click-through data as collaborative query sessions, we can substantially reduce the complexity of the problem.

(2) As we see, not every entry in the click-through data corresponds to real-world events. Navigational queries account for 21% of the total query frequency [17] so pruning irrelevant data can prepare a better ground for the approach. As an example, in one sample of data, we found the frequency of queries and page clicks of popular portal pages shown below in Table 2.

The frequencies are high but they really do not correspond to any real-world event. So in the data cleaning, preparation and transformation phases of the web data mining, we incorporate filtering methods to process the data. This step significantly improved the quality of the results.

Table 2. Frequent query-page pairs of popular portals

Query	Click URL	Frequency
Google	http://www.google.com	14236
Yahoo	http://www.yahoo.com	181820
Aol	http://www.aol.com	4774
Myspace	http://www.myspace.com	17104
Ask.com	http://www.ask.com	2213

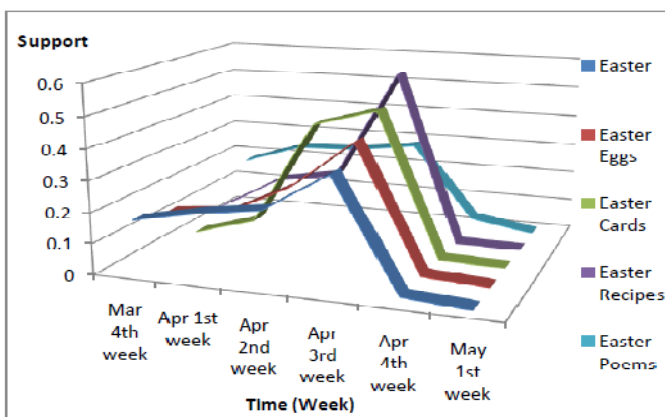


Fig. 1. Demonstration of query-page pair dynamics for “Easter” over a six week period

(3) We achieve the overall objective of event detections via query clustering. Event detection process ends with clusters of query-page pairs that are semantically and temporally related and corresponds to one or more events. We begin this process with queries because the number of queries the search engine receives (number of ways in which real-times queries are framed) are far less than the size of the Web i.e. $Q \ll P$. By this obvious fact, we believe that clustering can be done efficiently if we begin the process with Q . Also, query keywords give some insight about the events. Queries can be formulated in several ways in different contexts, although they all mean the same and correspond to the same event. For example, Figure 1 shows the support of query-pairs {"Easter", www.happy-easter.com}, {"Easter Egg", www.eeggs.com}, {"Easter Cards", www.easter-cards.com}, {"Easter Recipes", www.easter-recipes.com} and {"Easter Poems", www.poemsforfree.com}. All the four query-page pairs have similar evolutionary pattern in the time window and correspond to the same event "Easter" on April 16, 2006. As one can observe, the support increased gradually up to the 3rd week of April and then decreased gradually. By early detection of this kind of query clusters, event detection can be done efficiently. Therefore, we incorporate query clustering into the event detection framework.

2 Related Work

In this section, we review some significant work in the literature on event detection and query clustering. The beginning of event detection originates from the initial work done on (TDT) Topic Detection and Tracking [11] to automatically detect topically related stories within a stream of news media. The objective of the work done on retrospective and on-line detection [12] is to detect stories based on two tasks: retrospective detection and online detection. The retrospective detection aims to discover previously unidentified events in accumulated collection while the on-line detection tries to identify the on-set of news events from live news feeds in real-time. An attempt on burst event detection was done by Fungs et al. [13] from chronologically ordered documents as text streams. They proposed a parameter-free probabilistic approach called feature-pivot clustering to fully utilize the time information to determine set of bursty features in different time windows. The work done by Zhao et al. [16] introduced the dynamic behavior idea to cluster web access sequences (WASs), based on their evolutionary patterns of support counts. The intuition is that often WASs are event/task- driven and partitioning WASs into clusters result in grouping of similar/closer WASs. Later their work in [3] laid to the foundation for visitor-centric approach to detect events by using click-through data. The query-page relationship is represented as the vector-based graph. On the dual graph of a vector-based graph, a two-phased graph cut algorithm is used to partition the dual graph based on (i) semantic-based similarity and (ii) evolution pattern-based similarity to generate query-page pairs that are related to events. However, their work does not perform any data pruning and have query times the number of pages in the space. Later, another approach was introduced by Chen et al. [4] by transforming the click-through data to the 2D polar space by considering the semantic and temporal dimensions of the queries. It then performs a subspace estimation to detect subspaces such that each subspace

corresponds to queries of similar semantics, thus it complicates the solutions by doing first subspace estimation and then the pruning of uninteresting spaces.

The query clustering work by Wen et al. [7] is on the Encarta encyclopedia. Their approach was based on the two principles: (1) if users clicked on the same documents for different queries then the queries are similar. (2) If a set of documents are often selected for a set of queries then the terms in these documents are related to the terms of the queries to some extent. In the effort of extracting semantic relations from query logs, Baeza-Yates et al. [8] proposed a model to project queries in a vector space and deduced some interesting properties in large graphs.

3 Event Detection Framework

The overview of our proposed event-detection framework is shown in Figure 2 and is briefly explained in this section. Given the click-through data, we perform the data cleaning, preprocessing and transformation tasks to refine the data. As shown in Table-2, some portion of the click-through data does not correspond to real-world events (like searching Google, Yahoo as keywords, or some pornographic keywords). The percentage of irrelevant data is highly random depending upon the sample chosen. Filtering this noise is a better step to prepare ground for further process. In order to analyze the dynamics of increase and decrease of co-occurrences of query-page pairs, we partition the click-through data in a sequence of collections based on user-defined time granularity. Time granularities can be like a day, week, month etc. Different time granularities are required to detect events over moving window sizes. Each collection can be represented by a bipartite graph. We summarize the co-occurrences of query-page pairs from all the collections into a summarized bipartite graph. Next, we transform the problem of event detection into query clustering while capturing the relationship among queries and pages. For this purpose, we use the hybrid cover graph [8] and employ a distance-based function that includes the semantics of the query and pages to define the criteria for clustering. The summarized support from bipartite graph (i.e, edges in hybrid cover graph, see Figure 2 and Figure 4) is used to emphasize the dynamics of the queries and pages in the clusters to detect an event.

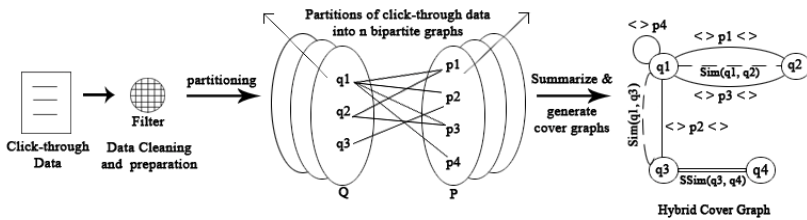


Fig. 2. Event detection framework overview

4 Data Representation

Click-through data is collected in Web logs. As mentioned earlier, we consider the click-through data as collaborative query sessions instead of individual query-page records. The reason for the same is explained earlier in Section 1.1. As informally defined earlier, a query session is wrapped by time boundaries; the beginning and the end time. We segment users’ streams into sessions based on anonymous ID. Another widely used technique [14] is based on the idea that two consecutive actions (either query or click) are segmented into two sessions if the time interval between them exceeds 30 minutes.

Definition 1: (Query session) A query session $S = (Q, P)$, where $Q = \{q_1, q_2 \dots q_m\}$ is a bag of queries issued through the search engine and $P = \{p_1, p_2 \dots p_n\}$ is the set of corresponding pages clicked by the user from the search result set, where n is not m .

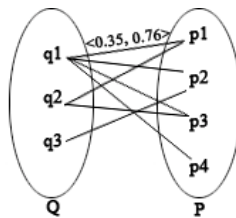


Fig. 3. Summarized bipartite graph

A Bipartite graph, $G = (V, E)$ where nodes in V represent queries and Web pages and edges in E represent the strengths of the query-page pairs. Bipartite graphs are widely used in the Web data mining domain [5, 6] to represent the relationship between queries and pages. An edge between a query and a page is formed if the page is clicked in response to the query. Bipartite graphs can be visualized as mapping between the query set (Q) and the page set (P) as shown in Figure 3. We like in [3] partition the click-through data C into sequence of collections $\langle C_1, C_2 \dots C_n \rangle$ based on user-defined time granularity like hour, day, week and month etc.

Definition 2: (Strength) of a query-page pair $P_{s,t} = (q_s, p_t)$ in collection C_i is $S_i(P_{s,t}) = \frac{|P_{s,t}(C_i)|}{\sum_{i=1}^n |P_{s,t}(C_i)|}$, where $1 \leq i \leq n$ and (s, t) is a query-page pair. Strength is the ratio of the co-occurrence of the query-page pair in collection C_i to C . This ratio so defined keeps the value ≤ 1 and is easy to process rather than showing high co-occurrence values without normalization. Note that in Figure 3 the strength of (q_1, p_1) is summarized as $\langle 0.35, 0.76 \rangle$ for two collections. Noisy query-page pairs that appear sporadically and potentially not related to any event have very low strengths.

In order to cluster queries with pages clicked, we need the efficient data structure for their representation. Several graph algorithms are in existence which can be used for this purpose. For example, Baza-Yates et al. [15] identified several types of query graphs. In all such graphs, queries are nodes and an edge is drawn between two nodes

if (i) the queries contain the same word(s) – word graph (ii) the queries belong to the same session – session graph (iii) users clicked on the same URLs from the result sets – cover graph. Word graph is hard to use because users formulate queries in different ways but it is essential to capture the query semantics. Not all the queries from a session correspond to some event so session graph is not the option. Both word and session graphs fail to capture the semantics of pages clicked. Cover graph can be efficient because for two queries with a commonly clicked page, the edge is represented only once. Reducing the complexity of the graph structure with emphasis on page clicks can simplify the problem and helps in easy representation. We extend the notion of cover graph to hybrid cover graph, which is explained later. The notion of commonly clicked documents [15] is as follows:

Definition 3: Query Instance is a query (set of words or sentences) plus zero or more clicks related to that query. Formally: $QI = (q, u^*)$ where $q = \{\text{words or phrase}\}$, q being the query, u a clicked URL and the query instance of query q is denoted by QI_q and $QI_{c(u)}$ denotes the set of its clicked URLs.

Definition 4: URL Cover is the set of all URLs clicked for a query. So for the query q , $UC_p = \cup_{QI_{qp}} QIc(u)$.

The nodes in the hybrid cover graph are queries from the click-through data. Three types of edges are possible between any two nodes: 1. Cover edge (represented by normal line) is drawn if a page is clicked in common to both the queries 2. Similarity edge (represented by dotted line) represents the similarity of the two queries; page content and user click feedback. 3. Session similarity edge (represented by double line \Rightarrow) is drawn if two queries are related to each other as a result of the association rule mining of most of the sessions, referred as session inference. The criterion for similarity over the similarity edge is based on distance function and session inferences.

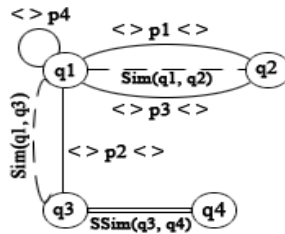


Fig. 4. Hybrid Cover Graph

The hybrid cover graph shown in Figure 4 is formed by incorporating the features of word and session graphs into the cover graph. $Sim(q1, q3)$ is the similarity edge that represents the similarity between the queries $q1$ and $q3$, which have common URLs clicked in response to them. The vectors on each side of the page $p2$, represented as $\langle \rangle p2 \langle \rangle$ indicate the summarized support of $p2$ with the corresponding query nodes. $SSim(q3, q4)$ is the session similarity between $q3$ and $q4$, which will be explained in Section 5.

5 Distance Function

Similarity between two queries correspond to nodes in a graph is based on our approach to integrate the semantics, structure, and content of queries and pages. Our distance criterion is based on work done by Wen et al. [7] to cluster queries.

5.1 Similarity Based on Query Contents

Although short queries are harder to understand, queries are better understood by considering them as keywords, words order and phrases. We perform stemming, stop words elimination, phrase recognition and synonym labeling while adding a query to the query semantics dictionary of a cluster. Let p, q be two queries.

Similarity based on Keywords or Phrases

$$\text{Sim}_{\text{keyword}}(p, q) = \text{KN}(p, q) / \text{Max}(\text{kn}(p), \text{kn}(q))$$

where $\text{KN}(p, q)$ = the number of common keywords in the queries p and q , $\text{kn}(p)$ = number of keywords in p .

Similarity based on String Matching

The comparison is the string-matching problem and can be computed by edit distance, i.e. number of edit operations required to unify two strings.

For letters and characters, $\text{Sim}_{\text{edit}}(p, q) = \text{EditDistance}(p, q)$

$$\text{Similarity}_{\text{content}} = \text{Sim}_{\text{keyword}} / \text{Sim}_{\text{edit}} \quad (\text{Sim}_{\text{keyword}} \text{ is based on the keywords})$$

5.2 Similarity Based on Session Feedback

A query can be expressed as a point in high-dimensional space [15] where each dimension corresponds to a unique URL i.e. a query can be given a vector representation based on all the different URLs in its cover. A weighted representation that takes document frequencies into account is used. If p and q are two queries then $\text{Sim}_{\text{vector}}$ (from the author-centric point of view) is computed as:

$$\text{Sim}_{\text{vector}} = \frac{\overline{p} \cdot \overline{q}}{|\overline{p}| \cdot |\overline{q}|}$$

Session feedbacks from meaningful query sessions can help to relate topically similar URLs. A simple way to take user feedbacks into consideration is by taking the normalized value to see the similarity in terms of the commonly clicked URLs for the queries. Sim_{doc} represents visitor-centric (generated by users browsing activity) point of view and

$\text{Sim}_{\text{doc}} = \text{RD}(p, q) / \text{Max}(|\text{Cover}(p)|, |\text{Cover}(q)|)$ where $\text{RD}(p, q)$ is the number of commonly clicked URLs and $|\text{Cover}(p)|$ is the number of URLs clicked for query p .

$\text{Sim}_{\text{vector}}$ is presented from the author-centric (generated by publishers) point of view, whereas Sim_{doc} is from visitor-centric point of view. This tells how users have chosen to click on these pages.

$$\text{Similarity}_{\text{feedback}} = \text{Sim}_{\text{vector}} * \text{Sim}_{\text{doc}}$$

Content-based measures tend to cluster queries with the same or similar terms whereas session feedback-based measures tend to cluster page clicks related to the same or similar topics.

Similarity $(p, q) = \alpha \text{ Similarity}_{\text{content}} + (1 - \alpha) * \text{Similarity}_{\text{feedback}}$ where α is the weight factor.

$$\text{Distance}(p, q) = 1 / \text{Similarity}(p, q)$$

Larger the similarity, smaller the distance and the weights for content and session feedback similarities are adjusted to obtain better results. An edge between two queries p and q in the hybrid cover graph is drawn if $\text{Distance}(p, q) \leq D_{\text{min}}$, where D_{min} is the minimum distance.

Association Rules [9] can be applied to find queries that are asked together in many of the query sessions. In the problem of finding related queries from query set Q , we are interested in associations like $X \Rightarrow Y$, where X, Y are subsets of $Q, X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ should have a support $\geq S_{\text{min}}$ and confidence $\geq C_{\text{min}}$, where S_{min} and C_{min} are minimum support and confidence values. Suppose the rule $q1 \Rightarrow q4$ given S and C where $S \geq S_{\text{min}}$ and $C \geq C_{\text{min}}$ is found then include the rule in the hybrid cover graph.

6 Clustering Process

For each query $q \in Q$, find the clusters (for first query, there is no cluster to compare so it forms its own cluster, for subsequent queries, find distances among the clusters obtained so far) with which the minimum distance condition is satisfied. Assign q to those clusters. Two queries $q1$ and $q2$ fall into the same cluster if the distance between $q1$ and $q2, D(q1, q2) \leq D_{\text{max}}$. If the threshold distance condition is not satisfied with any of the existing clusters then start a new cluster beginning with q .

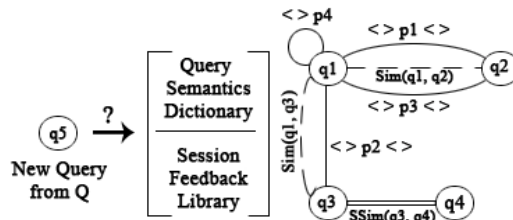


Fig. 5. Clustering Process

For example, as shown in Figure 5, when a new query $q5$ comes in, its content is compared with the semantics of the query dictionary formed from existing queries - $q1, q2, q3, q4$. Then its page clicks from the summarized bipartite graph are compared with the session feedback library of all the pages - $p1, p2, p3, p4$ for a given cluster. If the distance D is $\leq D_{\text{min}}$ then the query is added to the cluster, the query semantics are added to the query semantics dictionary and its page clicks are added to the session feedback library. If not, the query begins forming a new cluster.

6.1 Event Detection Algorithm

There are several challenges in designing a query clustering technique. It should be able to handle all types of attributes, scalable on massive datasets, work with high dimensional data, handle outliers, have reasonable time complexity, be independent of data order, and start without initial parameters (for example, the number of clusters). DBSCAN [10] algorithm and its incremental version meet all the required conditions and its average time complexity is $O(n \cdot \log n)$.

```

Input: Set of Queries Q
Output: Set of Clusters  $\Theta$ 
Initialization:  $\Theta = \emptyset$ , cluster=0;
ClusteringAlg(Q)
Begin
For each query q $\in$ Q do
  If cluster!=0 // clusters existing
  then
    For each existing Cluster Ci
      For each query qj  $\in$  Ci
        if distance(q, qj)  $\leq$  Eps
          Then Ci = Ci  $\cup$  q // add to cluster
             $\Theta = \Theta \cup C_i$  // update cluster set
          End if
        End for
      End for
    End for
  Else
    // no clusters are existing condition
    Then Cnew  $\cup$  = q // form a new cluster
       $\Theta = \Theta \cup C_i$  // add to cluster set
      cluster++; // cluster count tracked
    End Else
  End for
GenerateEventSub-graphs ( $\Theta$ ) // pass cluster set
End

```

Algorithm 1. ECO – Clustering Process

Our algorithm though inspired by the DBSCAN differs significantly from it. First, the function in our approach is not density-based but distance-based and second, we require only one scan of the queries through the click-through data. The criterion for distance function is explained previously in Section 5. The event detection algorithm is presented in two steps. Algorithm1 is for the clustering process and the later one is for generating the hybrid cover graphs. The hybrid cover graphs are drawn with respect to the comprehensive-reachable and comprehensive connected conditions of the DBSCAN algorithm for the terminal nodes. The algorithm runs at different time granularities to detect events of different window size like day, week and month etc.

```

Input: Set of Clusters  $\Theta$ 
Output: Set of hybrid cover graphs
GenerateEventSub-graphs( $\Theta$ )
Begin
  For each Cluster Ci  $\in$   $\Theta$ 
    For all Qi  $\in$  Ci
      DrawCoverGraph();
      Check-Comprehensive-Reachable();
      Check-Comprehensive-Connected();
    End for
  End for
End

```

Algorithm 2. Event Detection ECO – Hybrid Cover Graph

The summarized support values for the query-page pairs are analyzed using histograms to ensure that the hybrid cover graph has an evolutionary pattern. The higher ranking of nodes in the hybrid cover graph can be given for the connected dominating set (nodes that essentially connect the graph), nodes with least distance and with higher supports with their corresponding edges. The page rank of the edge can be obtained as the ItemRank from the click-through data. The edges with better ranks can be regarded as high quality Web pages clicked in relation to events.

Pruning irrelevant data is very important because the click-through data has millions of queries and pages. We reduced the size of the graph qualitatively and quantitatively by eliminating: 1. Queries and pages that have low support values. By doing so, some edges and nodes can be removed from the graph. These queries and pages can be seen sporadically in the data. 2. Multi-topical URLs (pages that talk about several topics or a very generic topic) by removing edges of low weight obtained from criteria in Section 5. Low weight edges are more likely to represent poor quality semantic relations.

7 Working Example

In this section, we explain the overall process by continuing the example initiated in Section 1.1. Figure 1 shows the support of query-pairs P1 {"Easter", www.happy-easter.com}, P2 {"Easter Egg", www.eeggs.com}, P3 {"Easter Cards", www.easter-cards.com}, P4 {"Easter Recipes", www.easter-recipes.com} and P5 {"Easter Poems", www.poemsforfree.com}. The co-occurrence, support for the query page pairs for the 6 week window period is shown in Tables 3 and 4. We show the similarity computation for the queries "Easter" and "Easter Eggs".

Table 3. Co-occurrence of query-page pairs over a 6 week window period

	31-March	7-April	14-April	21-April	28-April	04-May
P1	7000	8700	9900	1510	600	0
P2	9200	10500	16900	2740	1000	200
P3	300	1500	8200	9300	100	0
P4	1000	2900	3500	6900	0	0
P5	9100	8300	8500	9500	1200	0

Table 4. Support of query-page pairs over a 6 week window period

	31-March	7-April	14-April	21-April	28-April	04-May
P1	0.169	0.210	0.239	0.365	0.014	0
P2	0.141	0.161	0.259	0.420	0.015	0
P3	0.015	0.077	0.422	0.479	0.005	0
P4	0.058	0.170	0.205	0.564	0	0
P5	0.181	0.247	0.252	0.282	0.035	0

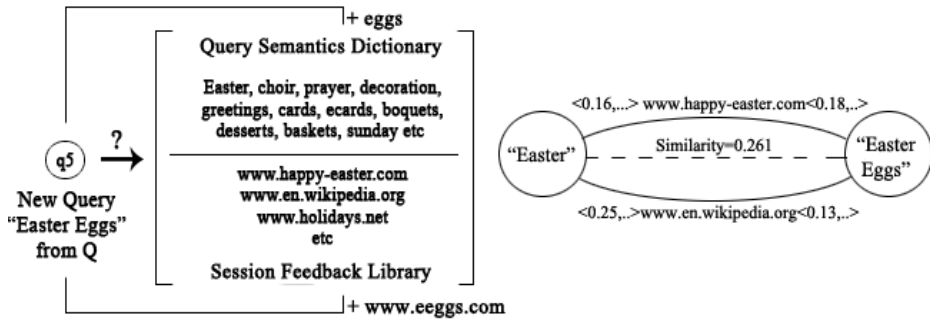


Fig. 6. Illustration of “Easter” and “Easter Eggs” clustering

$Sim_{keyword} = 1/2 = 0.5$; $Sim_{edit} = 4$, $Similarity_{content} = Sim_{keyword} / Sim_{edit} = 0.125$
 We computed $Sim_{vector} = 1.2$, $Sim_{doc} = 177/569 = 0.311$,
 $Similarity_{feedback} = Sim_{vector} * Sim_{doc} = 0.373$
 $Similarity("Easter", "Easter Eggs") = \alpha Similarity_{content} + (1 - \alpha) * Similarity_{feedback}$,
 where α is the weight factor and assume $\alpha = 0.45$.
 $Similarity("Easter", "Easter Eggs") = 0.261$,
 $Distance(p, q) = 1 / Similarity(p, q)$

Let $Distance = 1/0.261 = 3.83$. Assume $D_{min} = 3$ then the queries “Easter” and “Easter Eggs” should fall into the same cluster. The process is illustrated in Figure 6. Note that only the portion of hybrid cover graph with nodes “Easter” and “Easter Eggs” is shown because of the complexity of the graph. All the four query-page pairs are semantically and temporally related and have similar evolutionary patterns in the window period and corresponding to the same event “Easter” on April 16, 2006. As one can observe, the support increased gradually in the 3rd week of April and then decreased gradually. The criterion for distance function is explained in Section 5 and the clustering process is explained in Section 6.

8 Performance Study

In this section, we study the performance of our event detection approach. First, we describe the characteristics of the dataset used in experiments. Then we present the experimental results and their comparison with some of the existing work.

8.1 Data Set

A real click-through dataset obtained from AOL search engine is used in our experiments. The data is from March 2006 to May 2006, comprised of 500k query sessions, consisting ~20 web million queries and click-through activities from 650k users. As described in [17], each line in the data represents one of two types of activities: (i) a query that was not followed by the use clicking on a result item. (ii) a click through on an item in the result list returned from a query. In the later case, the pages appear

as successive entries in the data. In our approach, as a query session is obtained as successive pages corresponding to the same query from the same user. The timestamp of the first page click in a query session is taken as the start time of the session.

8.2 Result Analysis

Our approach can also detect pre and post period events, where the current period is referred to March through May, 2006. As discussed in Section 1.2 the co-occurrence of query-page pairs corresponding to an event does not stop abruptly right after the event but slows down at a faster rate. So pre and post period events can be detected by analyzing such kind of a behavior. For example pre-period event “Winter Olympics Torino, Italy” happened during February 10 through 26. We observed significant interest decreasing at a faster rate in regard to this in early March data. Post-period event “FIFA World Cup, Germany” during June 9 through July 9 is detected with increasing interest at the end of the May data.

Our algorithm can detect events of different time granularity like day, week and month. For an event, the traffic spreads around the event juncture like few days, weeks, and months in time granularity before and after the event. Day events like the death of Jack Wild, a famous British actor on March 1, the St. Patrick’s Day on March 17 etc. are detected. Week events like the Philadelphia flower show, (a big indoor flower show) during the week March 5 through 12, the Fleet week (public can see USA Navy and Coast guard ships) during the week May 24 through 30 etc. Monthly events span across bigger time frames and appeared throughout the data. The famous American Idol 5 episode appeared March 1 through May 24 (finale), the Internal Revenue Service (IRS) tax filing appeared March 1 through 31. Note that some of the events are regular and previously known like the St. Patrick’s Day; Good Friday etc. which recur every year. Some are previously unknown; like Simon Lindley, an Organist received the “Coveted Spirit of Leeds” award on May 3, the release of the movie V for Vendetta on March 17 etc. These events are not periodic and do not recur. Our approach could detect both types of events and of different time granularities. Our approach detected a lot of events that are not recognized previously by the existing work [3, 4] on the same dataset. Due to space restrictions we are not able to include the full list of events detected. The complete list of events detected is shown in Appendix.

8.3 Experimental Analysis

DECK [4] outperformed two-phase-clustering algorithm [3] so we compare the performance of ECO with the DECK, DECK-NP [4] and DECK-GPCA [4] on the same dataset. Number of events detected is a simple way to compare different approaches. ECO could detect 96 events where as DECK detected only 35 events. ECO could not detect 5 events in the list of 35 events detected by the DECK. On the other hand, DECK did not detect 61 events that ECO could detect. On time granularity comparison, ECO could detect 80 day events, 8 week events and 8 month events. In the events listed by DECK, 32 are day events, 3 are week events and no month events.

As mentioned earlier, our approach could detect 1 pre-period, 83 current period and 12 post period events. The experimental results are shown in Figures 7 below.

The evaluation metrics, precision, recall, F-measure (F-1 score) and entropy are used along with the number of events detected to compare the performance. Precision is the ratio of number of correctly detected events to the overall discovered clusters. Recall is the ratio of number of correctly detected events to the total number of events. F-measure is computed based on the precision and recall as the weighted harmonic mean of precision and recall. $F\text{-measure} = 2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$. For each generated cluster i , we compute P_{ij} as the fraction of query-page pairs (query sessions) representing the true event j . Then the entropy of the cluster i is $E_i = - \sum_j P_{ij} \log P_{ij}$. The total entropy can be calculated as the sum of the entropies of each cluster weighted by the size of each cluster: $E = \sum_i^m \frac{n_i * E_i}{n}$, where m is the number of clusters, n is the total number of query-page pairs (query sessions) and n_i is the size of the cluster i . The experimental results are shown in Figures 8. ECO did fairly well in terms of precision and recall for up to half of the data size. As the number of query sessions increased, the number of query patterns increased so the number of noisy query clusters increased which resulted in slight down fall in precision but not recall and increased in entropy.

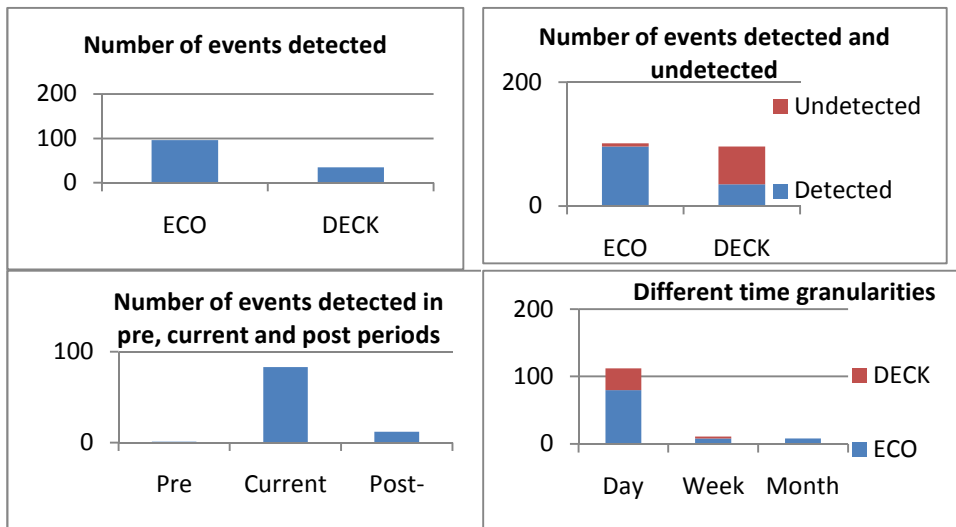


Fig. 7. Comparison of ECO with DECK on number of events detected

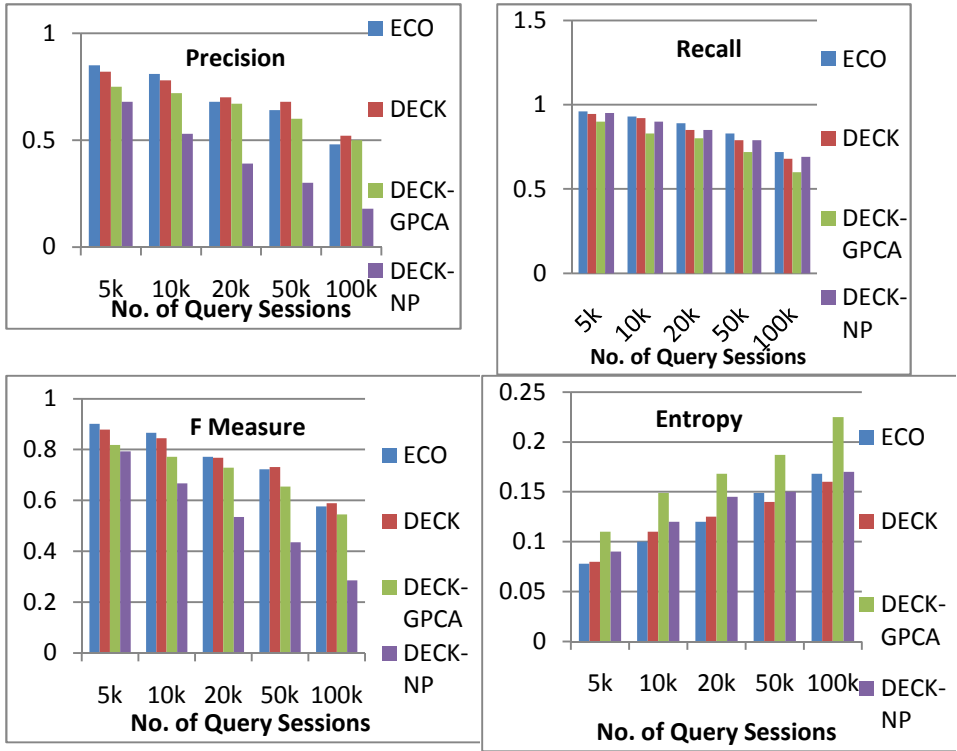


Fig. 8. Precision, recall, F-measure and entropy of ECO and DECK

8.4 Effect of α

The factor α decides the weights for content-based similarity and feedback-based similarity. We ran experiments varying the value of α , which is shown in Figure 9. The number of events detected varied accordingly. At $\alpha=0.15$, 31 events are detected. As the weight for feedback-based similarity increased we started identifying new clusters of events. At $\alpha=0.45$ we got the best results in terms of events detected. As the weight for feedback-based similarity increased further, the performance degraded.

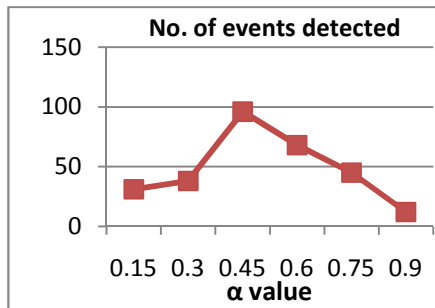


Fig. 9. Impact of α on Event Detection

9 Conclusions

In this paper, we proposed an approach called ECO for detecting events from the click-through data. Firstly we performed data cleaning, transformation and preparation process to filter the noise and then partitioned the click through data into collections of user defined granularity. Then we transformed the problem into query clustering, simultaneously trying to integrate the content, structure and semantics of the queries and clicked URLs. We introduced the hybrid cover graph to efficiently represent the clusters of query- page pairs. The evolutionary pattern of the query-page pairs is embedded into the hybrid cover graph as vectors over the edges to incorporate the dynamics. Our results outperform the existing work [3,4] in terms of the number of detected events, entropy measure, F-measure and recall.

References

- [1] De Kunder, M.: The size of the World Wide Web. World Wide Web Size (September 04, 2009), <http://www.worldwidewebsize.com>
- [2] Baeza-Yates, R.: Web Mining in Search Engines. In: Proceedings of the 27th Australasian Conference on Computer Science, New Zealand, vol. 26 (2004)
- [3] Zhao, Q., Liu, T.-Y., Bhowmick, S., Ma, W.-Y.: Event Detection from Evolution of Click-through Data. In: Proceedings of KDD, Philadelphia, PA, USA (2006)
- [4] Chen, L., Hu, Y., Nejdl, W.: DECK: Detecting Events from Web Click-Through Data. In: Eighth IEEE International Conference on Data Mining (ICDM), pp. 123–132 (2008)
- [5] Beeferman, D., Berger, A.: Agglomerative clustering of a search engine query log. In: SIGKDD (2000)
- [6] Xue, G.-R., Zeng, H.-J., Chen, Z., Yu, Y., Ma, W.-Y., Xi, W., Fan, W.: Optimizing web search using web click-through data. In: ACM Proceedings of CIKM, pp. 118–126 (2004)
- [7] Wen, J., Mie, J., Zhang, H.: Clustering user queries of a search engine. In: Proceedings of the 10th International World Wide Web Conference (2001)
- [8] Baeza-Yates, R., Tiberi, A.: Extracting Semantic Relations from Query Logs. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 76–85 (2007)
- [9] Federal, B.F., Fonseca, B.M., De Moura, E.S.: Using Association Rules to Discover Search Engines Related Queries. In: Proceedings of the 1st Conf. on Latin American Web Congress (2003)
- [10] Ester, M., Kriegel, H.-P., Jörg, S., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: 2nd International Conference on Knowledge Discovery, pp. 226–231 (1996)
- [11] Allan, J., Rapka, R., Lavarenko, V.: On-line New Event Detection and Tracking. In: SIGIR (1998)
- [12] Yang, Y., Pierce, T., Carbonell, J.G.: A Study of Retrospective and On-line Event Detection. In: SIGIR 1998 (1998)
- [13] Fung, G.P., Yu, J.X., Yu, P.S., Lu, H.: Parameter Free Bursty Events Detection in Text Streams. In: Proceedings of VLDB (2005)
- [14] White, R.W., Drucker, S.M.: Investigating Behavioral Variability in Web search. In: Proceedings of WWW, pp. 21–30 (2007)

- [15] Baeza-Yates, R.: Graphs from Search Engine Queries. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 1–8. Springer, Heidelberg (2007)
- [16] Zhao, Q., Bhowmick, S.S., Gruenwald, L.: CLEOPATRA: Evolutionary Pattern-Based Clustering of Web Usage Data. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 323–333. Springer, Heidelberg (2006)
- [17] Pass, G., Chowdhury, A., Torgeson, C.: A Picture of Search. In: the First ACM International Conference on Scalable Information Systems, Hong Kong (2006)

Appendix: List of Events Detected

Event	Time-stamp
Pre-period events	
Winter Olympics (Torino 2006)	February 26 th
Current-period events	
Ash Wednesday	March 1 st
Jack Wild died	March 1 st
World Baseball Classic	March 3 rd -20 th
48th Annual Heard Museum Fair	March 4 th , 5 th
78th Academy Awards	March 5 th
Triple Six Mafia won Academy Award	March 5 th
Philadelphia flower show	March 5 th -12 th
Dubai Tennis Open ends	March 6 th
Big 12 Women's Basketball Championship	March 7 th -12 th
Big Ten Conference Men's Basketball Tournament	March 9 th -12 th
NCAA men's Division I Basketball Tournament	March 14 th -April 3 rd
Ides of March	March 15 th
John West salmon commercial	March 15 th
Ram Bahdur Bomjon	March 16 th

disappeared	
V for Vendetta movie released	March 17 th
Saint Patrick's day	March 17 th
NCAA Women's Division I Basketball Tournament	March 18 th -April 4 th
Los Angeles Marathon	March 19 th
Washington D.C. Cherry Blossom Festival	March 25 th
27th Annual Young Artist Awards	March 25 th
Buck Owens died	March 25 th
Rocio Durcal died	March 25 th
Bataan Memorial Death March	March 26 th
Indy racing league season started	March 26 th
Solar eclipse in North Africa	March 29 th
Basic Instinct 2 movie released	March 31 st
April fool's day	April 1 st
Liberty Bell Classic	April 2 nd
140th anniversary of Baptist Union Baptist Church	April 2 nd
Good Friday	April 14 th
Scary movie 4 released	April 14 th
Easter	April 16 th

Boston Marathon	April 17 th
Stanley Cup Playoffs	April 21 st
Launch of lucky lines by Oregon Lottery	April 23 rd
Italian Social Republic	April 25 th
Dolphins Massacre at Zanzibar	April 28 th
Steve Howe died	April 28 th
The 33rd Annual Day-time Emmy Awards	April 28 th
Pleasant valley baseball tournament	April 29 th
The Hobbit movie started	April 31 st
27 th Sports Emmy Awards	May 1 st
David Blaine performance at Lincoln Center	May 1 st
Brooklyn Academy added to NHRP	May 2 nd
10000 days album release	May 2 nd
Simon Lindley received "Coveted Spirit of Leeds" award	May 3 rd
National Teachers day	May 4 th
Advanced Placement Test	May 1 st -10 th
Cindo de Mayo	May 5 th
Men's World Ice Hockey Championship	May 5 th -21 st
132 nd Kentucky Derby	May 6 th
29th Annual Five Boro Bike Tour	May 7 th
Fort Collins Old Town Marathon	May 7 th
Chris Daughtry eliminated from American Idol 5	May 10 th
Alligator attacks	May 14 th
Mother's day	May 14 th
Tony Awards nominations	May 16 th
The Amazing Race finale	May 17 th

Penny saved 1000\$ worth	May 17 th
Cannes Film Festival	May 17 th -28 th
Big Island Film Festival	May 18 th -21 st
The Davinci Code movie release	May 19 th
82nd Air Borne Division show	May 20 th
NASCAR Sprint All-Star Challenge	May 20 th
Strawberry Festival	May 21 st , 22 nd
10.5 Apocalypse Movie release	May 21 st
41st Annual Country Music Awards	May 23 rd
American Idol 5 ends	May 24 th
Fleet week	May 24 th -30 th
Africa day	May 25 th
31st Annual Million Dollar Beauty Ball	May 26 th
Ultimate Fighting Championship 60: Hughes vs. Gracie	May 27 th
The 90th Indianapolis 500	May 28 th
Memorial day	May 29 th
Post-period events	
The Omen movie release	June 6 th
06/06/06 Doomsday	June 6 th
FIFA World Cup (Germany)	June 9 th
National Golden glove boxing championship	June 9 th -13 th
60 th Annual Tony Awards	June 11 th
Juneteenth Day	June 17 th
Antique car show in Alabama	June 20 th

USA Outdoor Track and Field Championships	June 21 st -25 th
Air shows New England	June 24 th , 25 th
Ann Arbor art fair	July 19 th -21 st
58th Annual Primetime Emmy Awards	August 27 th
Albuquerque Balloon Festival	October 6 th -15 th
Month events	
NBA Basketball playoff	March, April
The Shoe show series aired on Resonance FM	March, April, May
American Idol	March, April,

	May
Annual walleye run in Fremont Ohio	March, April, May
IRS tax filing	March, April
Greenland ice melt by 250%	March, April
College Student Survey	March, April
1199 home care worker pay increase negotiations	March, April
Business Opportunities	
Summer - restaurants, resorts, cruises, islands etc	April, May

Requirements-Driven Qualitative Adaptation

Vítor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos

Department of Inf. Engineering and Computer Science, University of Trento, Italy
{vitorsouza,lapouchnian,jm}@disi.unitn.it

Abstract. Coping with run-time uncertainty pose an ever-present threat to the fulfillment of requirements for most software systems (embedded, robotic, socio-technical, etc.). This is particularly true for large-scale, cooperative information systems. Adaptation mechanisms constitute a general solution to this problem, consisting of a feedback loop that monitors the environment and compensates for deviating system behavior. In our research, we apply a requirements engineering perspective to the problem of designing adaptive systems, focusing on developing a qualitative software-centric, feedback loop mechanism as the architecture that operationalizes adaptivity. In this paper, we propose a framework that provides qualitative adaptation to target systems based on information from their requirements models. The key characteristic of this framework is extensibility, allowing for it to cope with qualitative information about the impact of control (input) variables on indicators (output variables) in different levels of precision. Our proposal is evaluated with a variant of the London Ambulance System case study.

Keywords: requirements, goal models, adaptive systems, feedback loops, qualitative reasoning.

1 Introduction

For software systems, as for humans and organizations alike, uncertainty is a given: at any time, the system is uncertain about all the details of its environment, or what might happen next. To cope with it, biological and social agents are capable of adapting their behavior and their objectives. Consistently with this, adaptation for software systems has become a focus of much research, addressing questions such as “How do we design adaptive systems?”, “What runtime support is needed?”, “How do we ensure that they have desirable properties, such as stability and quick convergence to an optimal behavior?”

We are interested in developing a set of design principles for adaptive software systems. We define adaptation as the process of the system switching from one behavior to another in order to continue to fulfill its requirements. Thus in adaptation, requirements remain unchanged and an adaptation strategy consists of choosing a suitable change of behavior to restore requirements fulfillment. Our proposed framework assumes that requirements should be at the very center of an adaptation mechanism, determining what constitutes normal behavior, what is to be monitored and what are possible compensations in case of deviations.

In earlier work, we have characterized a class of requirements, called *Awareness Requirements* (*AwReqs*) that determine what needs to be monitored by an adaptive system [25]. In addition, we extended goal models (which represent system requirements, as proposed in [15]) by including control-theoretic information concerning control variables and indicators, along with qualitative differential relations that specify the impact of the former on the latter [23].

The main objective of this paper is to “close the loop” by proposing a framework within which a failure of a monitored *AwReq* leads to a new behavior that consists of selecting a new variant of the system’s goal model, and/or new values for its control variables. The proposed mechanism is inspired by control theoretic concepts, notably the PID controller [13], recast in qualitative terms and using goal models to define both the desired output and the space of possible behaviors for getting it. Its key features are the use of requirements models for run-time adaptation and being highly extensible, allowing for different adaptation algorithms to be used depending on the availability and precision of information. To validate our proposal, we have implemented our framework and simulated our adaptation algorithms using different scenarios.

The rest of the paper is organized as follows: Section 2 summarizes our previous research, which serves as the baseline for this work; Section 3 presents the main contribution of this paper: an extensible framework for qualitative adaptation called *Qualia*; Section 4 describes how the framework was implemented and evaluated using simulations; Section 5 compares our approach with related work; Section 6 discusses challenges in handling multiple concurrent failures, introducing some of our on-going and future work; finally, Section 7 concludes.

2 System Identification for Adaptive Systems

In our previous work [23,25], we have applied a requirements engineering perspective to the problem of designing adaptive systems, focusing on developing a qualitative software-centric, feedback loop mechanism as the architecture that operationalizes adaptivity. Feedback loops introduce functionality to a system proper, providing monitoring of specified indicators and making the system aware of its own failures (i.e., aware of when not fulfilling its mandate). In these cases, a possible adaptation solution is to change the value of one or more system parameters which are known to have a positive effect on the necessary indicators.

In Control Theory, quantifying the effects of control input on measured output is a process known as *system identification* [13]. In some cases (e.g., a thermostat), and given the necessary resources, it is possible to represent the equations that govern the dynamic behavior of a system from first principles (e.g., quantitative relations between the amount of gas injected in the furnace and the change in temperature produced by it). For most adaptive information systems, however, such models are overly complex or even impossible to obtain. For this reason, in [23], we have proposed a systematic system identification method for adaptive software systems based on qualitative reasoning.

Our proposal is based on Goal-oriented Requirements Engineering (GORE), which is founded on the premise that requirements are stakeholder *goals* to be

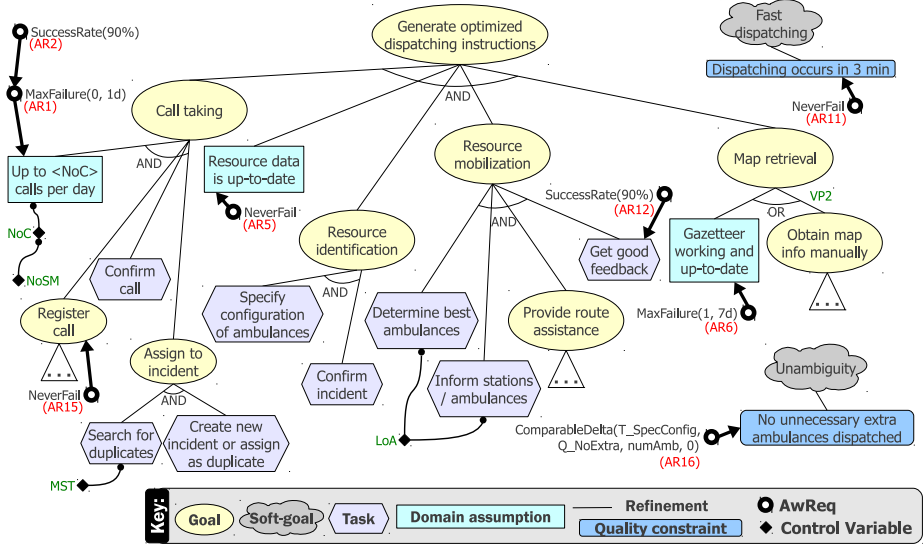


Fig. 1. Part of the goal model for the A-CAD system [22] after system identification

fulfilled by the system-to-be along with other actors. Goals are elicited from stakeholders and are analyzed by asking “why” and “how” questions [6]. Such analysis leads to goal models which are partially ordered graphs with stakeholder requirements as roots and more refined goals lower down, following obvious AND/OR Boolean semantics for goal satisfaction. *Goals* are refined until they reach a level of granularity where there are *tasks* an actor (human or system) can perform. On the other hand, *softgoals* are special types of goals that do not have clear-cut satisfaction criteria, and thus refined to measurable *quality constraints* for satisfaction. Finally, *domain assumptions* indicate states of the world that we assume to be true in order for the system to work. All of these elements are part of the ontology for requirements proposed by Jureta et al. [15].

An example of a goal model representing system requirements can be seen in Figure 1, which shows parts of the goal model of an *Adaptive Computer-aided Ambulance Dispatch system* (A-CAD), used as a running example throughout this paper. In the figure, triangles with points of ellipsis represent goal subtrees that are not relevant for the explanations contained herein and, thus, were removed to make the diagram simpler to read. The interested reader can refer to [22] for complete models and descriptions of the A-CAD.

Other than the aforementioned goal model elements, the diagram also shows some of the **indicators** and system **parameters** identified for the A-CAD. In our research, we use *Awareness Requirements* (*AwReqs*) [25] to define indicators of requirements convergence. *AwReqs* represent undesirable situations to which stakeholders would like the system to adapt, in case they happen (e.g., failure of critical requirements). Figure 1 shows eight of the sixteen *AwReqs* identified for the A-CAD, e.g., quality constraint *Dispatch occurs in 3 min* should never fail (*AR11*), *AwReq AR1* should have 90% success rate (*AR2*), etc.

Although *AwReqs* are not performance measurements per se, they are defined in terms of these measures (in the above examples, dispatch time and success rate), setting targets for requirement satisfaction. Currently, our framework uses strictly *AwReqs* as indicators and, therefore, in this paper the terms *indicator* and *AwReq* will be used interchangeably. The approach presented here could, however, be adapted to other kinds of indicators, as long as they are monitored and the system is made aware of their failures.

Parameters can be of two flavors. *Variation points* consist of OR-refinements which are already present in high variability systems (i.e., systems that offer different means of satisfying certain goals) and merely need to be labeled. E.g., the value of VP2 specifies if the system should assume the *Gazetteer is working and up-to-date* or if staff members should *Obtain map info manually*.

Control variables are abstractions over large/repetitive variation points and are represented by black diamonds attached to the elements of the model to which they refer. For instance, LoA represents the *Level of Automation* of tasks *Determine best ambulances* and *Inform stations/ambulances*, abstracting over the (repetitive) OR-refinements that would have to be added to them in order to represent such variability. LoA is an example of an enumerated variable (possible values are *manual*, *semi-automatic* and *automatic*), whereas MST (*Minimum Search Time*), NoSM (*Number of Staff Members working*) and NoC (*Number of Calls*, which is dependent on NoSM) are instead numeric.

Having identified the indicators to monitor and the parameters that can be tuned at runtime, we can finally model the effect changes in the latter have on the former in a qualitative way, which is done by means of differential relations. Considering indicator *AR11* as example, the A-CAD specification contains the following relations (and subsequent descriptions):

$$\Delta (AR11/NoSM) [0, MaxSM] > 0 \quad (1)$$

$$\Delta (AR11/LoA) > 0 \quad (2)$$

$$\Delta (AR11/MST) [0, 180] < 0 \quad (3)$$

$$\Delta (AR11/VP2) < 0 \quad (4)$$

- (1) More staff members increases the chance of satisfying *AR11*;
- (2) The higher the level of automation, the faster the dispatching;
- (3) Increasing the minimum search time makes dispatching take longer;
- (4) Obtaining maps manually contributes negatively to a fast dispatching.

As the examples above show, the syntax $\Delta (i/p) > 0$ represents the fact that if parameter p is increased, so is indicator i . This syntax borrows from Calculus the concept of differential equations: if $i = f(p)$, a positive differential $f' > 0$ means that the greater the value of p , the greater the value of i . Negative differentials are analogous. Note that for variation points the convention is that they “increase” from left to right. See [23] for further details.

Moreover, equations (1) and (3) exemplify the specification of boundaries for the specified effect, the former using a variable that represents the maximum number of staff members the ambulance service’s facilities can hold (to be specified later), the latter using numerical boundaries directly.

Finally, relations referring to the same indicator can be refined to specify: (a) if a change in one parameter has greater effect than another; and (b) if changing more than one parameter at the same time has cumulative effect on the indicator (which is assumed to be the default behavior). In our running example, an order has been established among the effects that different parameters have towards *AR11*, as shown in Equation (5). Absolute values are used in order to properly compare positive and negative effects.

$$\begin{aligned} |\Delta(AR11/VP2)| &> |\Delta(AR11/LoA)| > \\ |\Delta(AR11/MST)| &> |\Delta(AR11/NoSM)| \end{aligned} \quad (5)$$

In the field of Qualitative Reasoning, there is a spectrum of choices of qualitative representation languages, each of them providing a different level of *precision* (sometimes referred to as *resolution*) [10]. Some examples of qualitative quantity representation languages are [10]:

- *Status abstraction*: represents a quantity by whether or not it is normal;
- *Sign algebra*: represents parameters according to the sign of their underlying continuous parameter — positive (+), negative (–) or zero (0). It is the weakest form of representation that supports some kind of reasoning;
- *Quantity space*: represents continuous values through sets of ordinal relations, providing variable precision as new points of comparison are added;
- *Intervals*: similar to quantity space representation, consists of a variable-precision representation that uses comparison points but also includes more complete information about their ordinal relationship;
- *Order of magnitude*: stratify values according to some notion of scale, such as hyper-real numbers, numerical thresholds or logarithmic scales.

The proposed representation for the information elicited through system identification allows analysts to start with a very low level of precision (e.g., $\Delta(I/P) > 0$, similar to sign algebra) and evolve this specification when more information becomes available (e.g., $\Delta(I/P)[a, b] > 0$, using *landmarks* as boundaries of intervals). Such evolution can happen either horizontally (more information at the same level of precision) or vertically (increasing the precision of a specific information, e.g., $\Delta(I/P) = 2$, meaning $I = 2 \times P$, quantitative precision). The framework proposed in this paper can accommodate different levels of precision, enabling more elaborate adaptation algorithms when more precise information is available. The process and algorithms for refining precision of differential relations among indicators and control variables as the system operates remains an open problem on our to-do list.

3 A Framework for Qualitative Adaptation

As we have just seen, system identification adds to a requirements model qualitative information on how changes in system parameters affect indicators that are deemed important by the stakeholders. With this information, it is already

possible to propose an adaptation algorithm for when *AwReqs* fail at runtime, for instance: (1) find all parameters that affect the failed *AwReq* positively; (2) calculate the one(s) with the least negative impact on other indicators; (3) return a new system configuration changing the value of this/these parameter(s).

In this paper, we address two particular limitations of our current approach:

- There are still a few pieces of information missing regarding the requirements for adaptation. E.g., considering the algorithm proposed in the previous paragraph, the following questions (among others) are still unanswered: how many parameters should be changed and by how much? When calculating negative impact to other indicators, should priorities (e.g., [17], § 3.3) among them be considered? What if the *AwReq* fails again, should the previous adaptation attempts be taken into account when deciding a new one?
- Moreover, the adaptation algorithm exemplified above is just one of many possible algorithms that can be used given the available qualitative information about the system's dynamic behavior. Among the many possible algorithms, the choice of which to use should belong to the stakeholders and domain experts and, thus, be part of the system requirements specification. The adaptation framework should be able to accommodate this.

Therefore, in this paper we propose a framework to operationalize adaptation at runtime based on this qualitative information. We call this framework *Qualia* (**Q**ualitative **a**daptation). When made aware of a failure in an indicator, *Qualia* adapts the system by conducting eight activities, as shown in Figure 2 and described below (the numbers below match the ones in the figure):

1. One or more parameters modeled during system identification are chosen;
2. Based on the relation of this/these parameter(s) with the failed indicator, *Qualia* decides by how much it/they should be changed;
3. The chosen parameter(s) are then incremented (consider decrements as negative increments for simplicity) by the calculated value(s);
4. The framework waits for the change to produce any effect on the indicator;
5. *Qualia* evaluates the indicator again after the waiting time;
6. In each cycle, *Qualia* may learn from the outcome of this change, possibly evolving the adaptation mechanism and updating the model;
7. Finally, it decides whether the current indicator evaluation is satisfactory and either concludes the process or starts over;
8. If it decides to start over, it reassesses the way adaptation was conducted in the previous cycles, possibly adapting itself for the following cycle.

To accommodate the different levels of precision, we propose an extensible framework by defining an interface for each activity in the process of Figure 2 and providing default implementations that assume only the minimum amount of information is available. Then, we allow designers to create and plug-in new *procedures* into *Qualia*, possibly requiring more information about the system in order to be applicable. We use the term *adaptation algorithm* to refer to the set of procedures chosen to support the adaptation process. In the requirements specification, analysts should indicate which *adaptation algorithms* to use in response to each indicator failure.

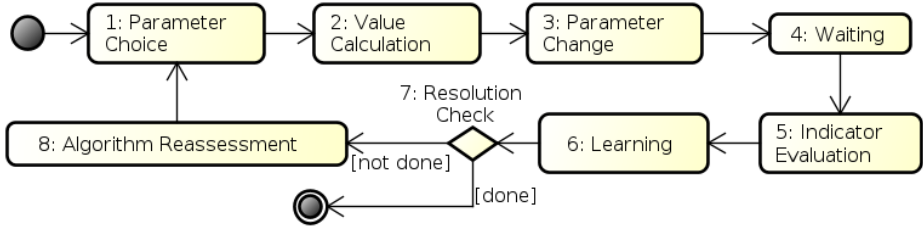


Fig. 2. The adaptation process followed by the *Qualia* framework

In the following sub-sections, we present three *adaptation algorithms*: the Default Algorithm (§ 3.1), the Oscillation Algorithm (§ 3.2) and the PID-based Algorithm (§ 3.3). To illustrate how *Qualia* can accommodate different levels of precision, we also propose different procedures (§ 3.4) for the first step of its process (*Parameter Choice*). An important remark here is that we do not make any claim on which adaptation algorithm is better suited for any particular context, but instead we just illustrate how this framework can be extended as needed. The choice of algorithm to use is the responsibility of the analysts.

3.1 The Default Algorithm

As mentioned earlier, when adapting the system, *Qualia* executes the algorithm that has been associated with the failure at hand by stakeholders or domain experts. When a particular algorithm is not specified, *Qualia* executes the *Default Algorithm*, which requires minimum information from the requirements models:

- **Indicators:** *Qualia* has to be notified of indicator failure, hence the model should specify what are the relevant indicators in a way such that another component of the feedback loop is able to monitor them. For this purpose, we use *AwReqs* (cf. Section 2) and its monitoring infrastructure [25];
- **Parameters:** to adapt to an indicator failure, there should be at least one related parameter. Section 2 also described how this information is specified through differential equations;
- **Unit of increment:** each numeric parameter must specify its unit of increment, because *Qualia* will not be able to guess it.

The *unit of increment* is important for the comparison among indicator/parameter relations. E.g., the comparison $|\Delta (AR11/MST)| > |\Delta (AR11/NoSM)|$ presented earlier as part of Equation (5), should be complemented by $U_{NoSM} = 1$ and $U_{MST} = 10 \text{ seconds}$, meaning that changing MST by 10s improves AR11 more than changing NoSM by 1 staff member. Moreover, enumerated parameters must be ordered (cf. [23]) and their unit of increment defaults to choosing the next value in the order.

The *Default Algorithm* is composed of eight default procedures, one for each activity of the process depicted earlier in Figure 2 (again, the numbers below match the numbers in the figure):

1. *Random Parameter Choice*: picks one parameter randomly from the set of parameters related to the failed indicator, considering those which can still be incremented by at least one unit (i.e., are within their boundaries).
2. *Simple Value Calculation*: decides the increment value for the chosen parameter, by multiplying the value of the parameter's unit of increment U by the indicator's increment coefficient K , returning the value $V = K \times U$.

The increment coefficient is an optional parameter (with default value $K = 1$) that can be associated to each indicator in the specification to determine how critical it is to adapt to their failures. Higher values of K will produce more significant changes, but the requirements engineer should be aware of the risks of overshooting. Note also that parameters should never exceed their boundaries.

3. *Simple Parameter Change*: changes the chosen parameter by the calculated value, at the *class* level.

The *class/instance* terminology is inherited from our previous work [25]: changes at the *class* level will affect the system "from now on", whereas changes at the *instance* level only affect the current execution of the system.

4. *Simple Waiting*: waits until the next time the indicator is evaluated by the monitoring component of the feedback loop.
5. *Boolean Indicator Evaluation*: verifies if, after executing the first four steps of the process, the next time the indicator succeeded.
6. *No Learning*: in the *Default Algorithm*, learning is skipped.
7. *Simple Resolution Check*: stops the process if the outcome of the indicator evaluation (step 5) was positive, otherwise it iterates.
8. *No Algorithm Reassessment*: the *Default Algorithm* does not reassess or adapts itself, but always executes the same procedures in every iteration.

Let us illustrate the above algorithm using the A-CAD. Imagine that for a given emergency call received, an ambulance was not dispatched within three minutes, breaking indicator (*AwReq*) *AR11* (quality constraint *Dispatching occurs in 3 min* should never fail). Available parameters to improve this indicator are *NoSM*, *LoA*, *MST* and *VP2* (assuming all are within boundaries). For this example, consider that the *Random Parameter Choice* procedure chose *MST*.

Imagine further that the specification says that $K_{AR11} = 2$ and we know that $U_{MST} = 10s$ and, moreover, Equation (3) says that *MST* contributes negatively to *AR11*. Therefore, the *Simple Value Calculation* procedure decides to decrease *MST* by $V = 2 \times 10s = 20s$ and, as a consequence, the *Simple Parameter Change* procedure does so at the *class* level, i.e., for all dispatches following the one that did not satisfy *AR11*, until further notice.

Since *AR11* is evaluated at every dispatch, the next dispatch will resume the process (*Simple Waiting* procedure) and the *Boolean Indicator Evaluation* procedure will check if, after *MST* was reduced by 20s, the next dispatch took less than 3 minutes to complete. If the 20s reduction was effective, then the *Simple Resolution Check* procedure will stop the process; otherwise it will repeat the same procedures as above.

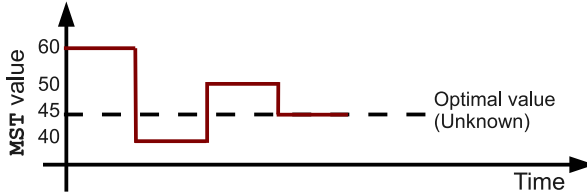


Fig. 3. A scenario of use of the *Oscillation Algorithm* in the A-CAD

As mentioned earlier, the requirements engineer should include in the requirements specification which algorithm — i.e., which set of procedures — should be used for each system failure. The *Default Adaptation Algorithm* can be represented by the empty set \emptyset , meaning that all the default procedures described above will be used. Other algorithms, as will be described next, are represented by naming the non-default procedures that compose them: the specified procedures replace their default counterparts (the one with the same interface), keeping the default ones that have not been replaced.

3.2 The Oscillation Algorithm

One of the desired characteristics of control systems is to avoid overshooting its control inputs. For instance, if an ambulance dispatch takes $3min10s$, we decide to reduce MST from $60s$ to $0s$ and the next dispatch takes only $2min10s$, we have overshoot MST's decrement by $50s$. Granted, this overshoot could be corrected whenever some other indicator (e.g., *AR16*, which controls if unnecessary ambulances are sent to incident sites) fails and MST is chosen to be incremented. Still, a good adaptation algorithm tries to avoid overshooting in the first place and, in what follows, we present one such algorithm.

The *Oscillation Algorithm* works as depicted in Figure 3: back to the *AR11* / MST scenario, imagine that given the current circumstances, the optimal¹ value for MST is $45s$. The controller obviously does not know it, so when *AR11* fails, it decreases MST to $40s$, which actually solves the problem. However, instead of stopping here, the algorithm **assumes to have overshoot the change**, and thus starts incrementing the same parameter in the opposite direction, using **half of the previous increment value**. When MST is set to $50s$, *AR11* fails again, which makes the controller switch increment direction and halve the increment value one more time. This process goes on until one of the following conditions:

- The parameter is incremented to a value that it has already assumed before, which means that we should be very close to the optimal value. E.g., if we

¹ Here, we consider “optimal” the smallest change that fixes the problem, because we assume every adaptation brings negative side effects to other indicators. If this is not the case, one could just set the parameter to its maximum (minimum) value from the start and no adaptation is necessary.

continue the oscillations shown in Figure 3, MST will assume values 47s, 46s, 45s and then stop;

- The algorithm has already performed the *maximum number of oscillations*, which is an optional attribute that can be assigned to a specific *AwReq* or to the entire goal model. Here, we consider each inversion of increment direction to be an oscillation (three, in the figure);
- The increment value is halved to an amount that is lower than the *minimum change value* of the parameter at hand (optional). For instance, Figure 3 represents the case in which this value is 5s. Note that, for integer variables such as MST, 1 is the lowest possible value.

In order to tune this algorithm, the framework also allows for the specification of parameters' *halving factors* different from the default value of 0.5. When oscillating, the increment value will be multiplied by the specified factor. The table below summarizes the *Oscillation Algorithm*:

Oscillation Algorithm	
Specification	{ <i>Oscillation Parameter Choice</i> , <i>Oscillation Value Calculation</i> , <i>Oscillation Resolution Check</i> }
Properties	<ul style="list-style-type: none"> – <i>Maximum number of oscillations</i> (optional); – <i>Minimum change value</i> (optional); – <i>Halving factors</i> (default = 0.5).

The *Oscillation Resolution Check* procedure assumes to have overshoot when the problem is fixed and begins the oscillations, whereas the *Oscillation Value Calculation* procedure is responsible for determining when the value should be increased or decreased and when it should be halved. The *Oscillation Parameter Choice* procedure replaces the default, random one by choosing the parameter randomly at first, but then maintaining the choice until the end of the oscillations. Later, in Section 3.4, other parameter choice procedures will be illustrated, some of which could also be used here.

3.3 The PID-Based Algorithm

As mentioned in Section 1, our framework's controller is inspired by control-theoretic concepts, notably the Proportional-Integral-Differential (PID) controller. This controller is widely used in the process control industry and provides an efficient algorithm (described in most books on Control Theory, e.g., [13], Chapter 9) to keep a single output of the target system as close as possible to the specified, single reference input.

The question that arises then is the following: given its proven efficacy, would it be possible to use the actual PID algorithm in our models? First, since the PID algorithm works with *single input/single output* (SISO) and information systems usually have *multiple inputs/multiple outputs* (MIMO), this algorithm would work well only when the analyst can identify, for a given indicator, one single parameter whose changes have a significant effect in the indicator's outcome. Moreover, since this algorithm requires a numeric value for the control error

and *AwReqs* (our indicators) are somewhat of a Boolean nature (*success = true|false*), we need a way to extract a numeric value from them.

As detailed in [25], *AwReqs* can be divided in three categories: *Delta AwReqs* impose constraints over properties of the domain (e.g., “number of ambulances at the incident should not be greater than the number specified”), *Aggregate AwReqs* determine requirements’ success rates (“75% of the ambulances should arrive within 8 minutes”), and *Trend AwReqs* impose constraints over aggregated success rates over time (“success rate of *Get good feedback* should not decrease two weeks in a row”). *Qualia* will extract numeric control errors from these types of *AwReqs* as follows:

- *Delta AwReqs* : if the property is numeric, calculate the difference between desired and monitored values. In the above example, they are the specified number and the actual number of ambulances at the incident;
- *Aggregate AwReqs* : calculate the difference between the desired and actual success rates. Note that *AwReqs* of the form “*R* should never fail” can be translated into “*R* should have 100% success rate”;
- *Trend AwReqs* : calculate the difference between the last two measured success rates. In the above example, if the rate decreases in 7% in the first week and then again by 4% in the second, the control error is 4%.

If the *AwReq* in question follows one of these patterns, the *PID Algorithm* can be used. As the table below indicates, the algorithm affects *Qualia*’s procedures for value calculation, indicator evaluation and resolution check.

PID-based Algorithm	
Specification	{ <i>PID Value Calculation, PID Indicator Evaluation, PID Resolution Check</i> }
Properties	None.

3.4 Other Procedures

In the beginning of Section 3, we have stated that *Qualia* supports different levels of precision by allowing for new procedures to be implemented and plugged in to the framework for each of the eight activities in its adaptation process (Figure 2). To illustrate how our proposed framework can be extended, we focus here on the *Parameter Choice* activity and describe new procedures that execute it differently from the default one, especially in the presence of more precise information in the specification:

- *Shuffle Parameter Choice*: with the same amount of information used by the *Random Parameter Choice* procedure, this procedure randomly puts the system parameters in order during the first cycle and picks the next one using this pre-defined sequence when switching parameters is required.

A new property — *repeat policy* — determines when the parameter choice procedure should repeat the last used value or switch to a different one. Its default value is *repeat while incrementable*, but it can be set to *repeat M times*, where *M* is also configurable.

- *Ordered Effect Parameter Choice*: if differential relations regarding the indicator in question have been refined to provide comparison of their effect (as explained in Section 2), this procedure orders the parameters according to their effect on the indicator and uses them in this order.

Other than the *repeat policy* property, an *order* property is also relevant to this procedure, specifying if relations should be placed in *ascending* or *descending* order of effect (depending if stakeholders would like to start with the parameters that have the greatest or the smallest effect on the indicator). Moreover, if the set of relations concerning an indicator is only partially ordered, the *remaining parameters* property specifies if the non-ordered relations should be excluded from the list or shuffled at the end of it.

- *Ordered Side Effect Parameter Choice*: in case priorities among indicators are given (using, e.g., [17]), this procedure orders the parameters according to the priority of the indicators to which the parameter change would contribute negatively. It is particularly suitable for lower-priority indicators that can, in general, be sacrificed to maintain high-priority ones.

The *side effect calculation* property specifies if the *average* of the priorities of the indicators that suffer side effects should be calculated or if only the *highest* priority should be considered. The *remaining parameters* and *order* properties are also relevant here.

- *Ordered Maturation Time Parameter Choice*: domain experts can specify an optional attribute to differential relations called *maturation time*, which indicates how long it takes for the changes in the related parameter to take effect in the related indicator. Take, for instance, the scenario described in Section 3.1, and say *Qualia* has chosen NoSM instead of MST to adapt for the failure in *AR11*. Hiring and training a new staff takes a few days and, thus, the framework should wait for this specified time before continuing. Hence, this procedure will order the relations by their *maturation times*. As with the other ordered parameter choice procedures, the *order* parameter is also relevant here. Notice that relations' *maturation time* attribute also affects the *Default Waiting* procedure, illustrated earlier.

Finally, all of the procedures presented above can be further customized by the *number of parameters* property, which defaults to 1, but can be set to any positive integer, or even *all* parameters, mimicking the behavior of a *multiple input, single output* (MISO) control system. As demonstrated throughout this section, our proposed framework can be extended as needed by requirements engineers, depending on stakeholder requirements.

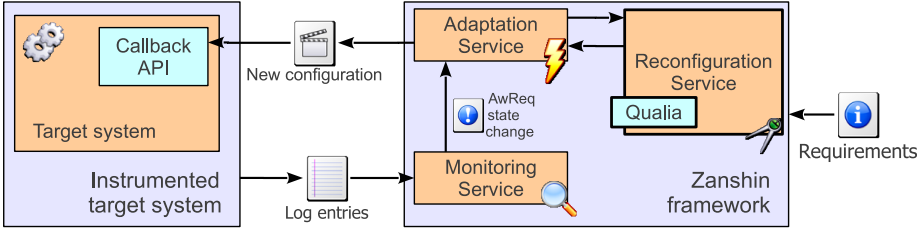


Fig. 4. Overview of the *Zanshin* framework and the addition of *Qualia*

4 Implementation and Evaluation

To evaluate *Qualia*, the framework described in Section 3, we have implemented it as a component of the *Zanshin* framework. Proposed in [24] (and available at <http://github.com/vitorsouza/Zanshin>), *Zanshin* applies an Event-Condition-Action (ECA)-based process to adapt to *AwReq* failures by effecting changes in other requirements in the model. Figure 4 shows an overview of the framework, highlighting with thicker borders the components added by this paper.

The monitoring infrastructure of our previous work [25] has been used to identify *AwReq* (indicator) failures from the *log entries* of the *instrumented target system*. The *Monitoring Service* will then notify *Zanshin*'s *Adaptation Service* about *AwReq* state changes (e.g., *AR11* has succeeded, *AR11* has failed, etc.). In some cases, based on the system *requirements*, this component might conclude that reconfiguration should be used, and asks the *Reconfiguration Service* for one of its registered *reconfiguration strategies*. *Qualia* is registered as a strategy, but *Zanshin* allows for other reconfiguration frameworks to be plugged-in (e.g., some existing frameworks are described as related work in Section 5). After the selected *reconfiguration strategy* produces a new configuration, the *Adaptation Service* sends it to the *target system* through a *callback API*.

The framework was implemented as a set of OSGi bundles and its requirements meta-models were specified using the Eclipse Modeling Framework (EMF), as shown in Figure 5. Because of space constraints, the meta-model for requirements specifications in *Zanshin* will not be reported here, but the reader can refer to [24] for its description. Figure 5 shows four elements from A-CAD's goal model, which were depicted earlier in Figure 1: root goal *Generate optimized dispatching instructions*, softgoal *Fast dispatching*, its quality constraint (QC) *Dispatching occurs in 3 min* and *AwReq AR11*, which targets that QC.

In the `<strategies>` tag, we can see that *Qualia* has been selected as *reconfiguration strategy* for failures of *AR11*. Further below, the `<configuration>` tag specifies parameter *MST* as a numeric control variable (ncv), with U_{MST} set to 10 and initial value 60. Finally, the `<relations>` tag represents the differential relation shown back in Equation (3): $\Delta (AR11/MST) [0, 180] < 0$.

Based on experimental evaluation methods of Design Science [14], we developed simulations to mimic the behavior of the A-CAD in different possible runtime scenarios, in order to evaluate the framework's response to system failures.

```

<?xml version="1.0" encoding="UTF-8"?>
<acad:AcadGoalModel ...>
  <rootGoal xsi:type="acad:G_GenDispatch">
    ...
    <children xsi:type="acad:S_FastDispatch"/> <!--7-->
    <children xsi:type="acad:Q_Dispatch" softgoal="//@rootGoal/@children.7"/>
      <!--12-->
    ...
    <children xsi:type="acad:AR11" target="//@rootGoal/@children.12"
      incrementCoefficient="2">
      <condition xsi:type="model:ReconfigurationResolutionCondition"/>
      <strategies xsi:type="model:ReconfigurationStrategy" algorithmId="qualia"
        >
        <condition xsi:type="model:ReconfigurationApplicabilityCondition"/>
      </strategies>
    </children> <!--26-->
  </rootGoal>
  <configuration>
    <parameters xsi:type="acad:CV_MST" type="ncv" unit="10" value="60" metric
      ="integer"/>
    </configuration>
    <relations indicator="//@rootGoal/@children.26" parameter="//
      @configuration/@parameters.0" lowerBound="0" upperBound="180"
      operator="ft" />
  </acad:AcadGoalModel>

```

Fig. 5. Part of the A-CAD requirements specified as an EMF model

The simulations send logging messages to the *Monitoring Service*, equivalent to the ones that would have been sent by a real system, indicating a failure. For instance, one of the implemented simulations produces log entries that indicate that *Dispatching occurs in 3 min* was not satisfied, which triggers a failure of *AR11*. Based on the EMF model of Figure 5, *Zanshin* activates *Qualia*, which executes its *Default Algorithm*, described and illustrated in Section 3.1.

The result of this particular simulation is shown in Figure 6. In this output, *S* represents the simulation (i.e., the *target system*), *Z* is *Zanshin* and *Q* is for *Qualia*. Figure 5 shows that *Qualia* selected *MST* and reduced its value to 40s, but another failure in *AR11* followed, and therefore the parameter was again reduced to 20s, which solved the problem.

Another simulation uses a randomly generated goal model with different number of parameters (from 100 to 1000, scaling up by 100 elements each time), all of them related to a failing *AwReq*. *Zanshin* and *Qualia* were timed in ten sequential executions of this simulation and average times for each number of parameters, as shown in Table 1, indicate linear scalability. In effect, by analyzing *Qualia*'s default algorithm, one can conclude that its complexity is $O(N \times R)$, where N is the *number of parameters* to choose and R is the number of differential relations in the model. With proper data structures, however, this complexity can be further reduced. In [24], we showed that *Zanshin* also scales linearly to goal models of increasing number of elements.

Qualia and *Zanshin* are part of a broader research proposal for the design of adaptive systems using a control theoretic perspective founded on requirements. Further evaluation efforts are in our future research plans, including experiments with actual running systems, user surveys to evaluate our methods and modeling language, then finally full-fledged case studies with partners in industry.


```

S: A dispatch took more than 3 minutes!
Z: State change: AR11 (ref. Q_Dispatch) -> failed
Z: (S1) Created new session for AR11
Z: (S1) Selected strategy: ReconfigurationStrategy
Z: (S1) Exec. ReconfigurationStrategy(qualia; class)
Q: Parameters chosen: [CV_MST]
Q: To inc/decrement in the chosen parameters: [20]
S: Instruction received: apply-config()
S: Parameter CV_MST should be set to 40
Z: (S1) The problem has not yet been solved...
-----
S: A dispatch took more than 3 minutes!
Z: State change: AR11 (ref. Q_Dispatch) -> failed
Z: (S1) ...
Q: Parameters chosen: [CV_MST]
Q: To inc/decrement in the chosen parameters: [20]
S: Instruction received: apply-config()
S: Parameter CV_MST should be set to 20
-----
S: A dispatch took less than 3 minutes.
Z: State change: AR11 (ref. Q_Dispatch) -> succeeded
Z: (S1) Problem solved. Session will be terminated.

```

Fig. 6. Result of the A-CAD simulation in which *AR11* fails

Table 1. Average time (in milliseconds) for executions of *Qualia* and *Zanshin*

Elements	<i>Qualia</i>	<i>Zanshin</i>	Elements	<i>Qualia</i>	<i>Zanshin</i>
100	40.4	1,187.5	600	5,212.6	14,323.4
200	1,064.4	4,416.3	700	6,244.4	15,568.3
300	2,098.5	10,122.3	800	7,283.3	18,811.1
400	3,132.2	11,851.1	900	8,314.6	20,621.6
500	4,164.8	13,097.7	1000	9,169.0	26,135.4

5 Related Work

In the field of requirements-driven adaptation two well-known proposals are the RELAX framework [27] and FLAGS [1], the former based on structured natural language whereas the latter uses goal models. Both of them use fuzzy logic in order to transform “crisp” (invariant) requirements into “relaxed” ones in order to capture uncertainty. Additionally, in FLAGS, adaptive goals define countermeasures to be executed when goals are not attained, using ECA rules. The GAAM approach [21] models quantifiable properties of the system as attributes, while specifying the order of preference of adaptation actions towards goals in a preference matrix, and the desired levels of attributes of each goal in an aspiration level matrix.

Several approaches in the literature propose adaptation through *reconfiguration*, i.e., switching the system’s behavior by finding a new configuration for its parameters. Wang & Mylopoulos [26] propose algorithms that suggest a new configuration without components that have been diagnosed as responsible for a failure; Nakagawa et al. [19] developed a compiler that generates architectural configurations by performing conflict analysis on goal models; Fu et al. [11] use reconfiguration to repair systems based on an elaborate state-machine diagram

that represents the life-cycle of goal instances at runtime; Peng et al. [20] assign preference rankings to softgoals and determine the best configuration using a SAT solver; Khan et al. [16] apply Case-Based Reasoning to find the best configuration; Dalpiaz et al. [5] propose an algorithm that finds all valid variants to satisfy a goal and compares them based on their compensation/cancelation cost and benefit (e.g., contribution to softgoals).

Like us, Filieri et al. [8] have also applied control theory to the problem of designing adaptive systems with a requirements perspective, focusing on adapting to failures in reliability and modeling requirements using Discrete Time Markov Chains (DTMCs). There, transitions are labeled with control variables, whose values can be set by a controller that decides the system's settings in order to keep satisfying the requirements. Well established control theoretic tools are used to design such controller and the authors claim the approach can be extended to deal with failures of different nature. An extension [9] proposes a more efficient solution for dynamic binding of components and an auto-tuning procedure.

Our work is also related to design-time trade-off approaches, considering that they could be tailored for the type of reasoning needed for run-time adaptation. For instance, Heaven & Letier [12] use stochastic simulation in order to generate quality values which are used to compute objective functions over a goal model, simulating design decisions in order to compare and optimize them.

Compared to the above approaches, the novelty in our proposal is the use of qualitative information about requirements, allowing analysts to start with the minimum information at hand and add more as further details about the system become available. In many cases, quantitative approaches might be difficult or even impossible to apply accurately and reliably due to the relativity of numerical values, incorrect mathematical judgment, non-linearity of value functions, etc. [7]. Furthermore, we advocate for expressive, but simple requirements models, believing that heavy formalisms such as linear temporal logic, fuzzy logic and DTMCs can, in some cases, place unnecessary burden on developers.

Qualitative reasoning has also been used by others to analyze system requirements in a similar fashion to what we propose. Menzies & Richardson [18] propose a matrix that depicts the contribution of process actions to interesting indicators (positive, negative, unknown or none) and use stochastic simulation to analyze this matrix and decide the best choice of actions, considering stakeholder-assigned utility values for each indicator. The proposed matrix conveys the same kind of information as our differential equations, albeit our models have considerably more expressive power. Elahi & Yu [7] also focus on requirements trade-offs at design-time, making pair-wise comparison of alternatives with respect to goals that were selected as indicators. We propose a more concise and expressive means to represent such comparisons, namely differential equations. Furthermore, both approaches focus on design-time decisions whereas our proposal targets run-time adaptation.

Finally, the use of control-theoretic concepts in our research (advocated by recent survey/roadmap papers such as [3,4]) comes from the fact that, in order to be adaptive, systems need to implement some kind of monitor-adapt feedback

loop. Given our Requirements Engineering perspective, our approach makes explicit in the models both requirements for monitoring (*indicators/AwReqs*) and adaptation (the chosen *adaptation algorithms*), allowing developers to design adaptive systems all the way from requirements to implementation.

6 Discussion and Future Work

The models proposed in this approach are a first step towards a comprehensive method for the specification of adaptation requirements based on GORE and qualitative reasoning techniques. Moreover, the *Qualia* framework offers a prototype for the operationalization of such requirements at runtime, alleviating developers of most of the effort of implementing the features of a feedback loop. Nonetheless, there is still a lot of work to be done, especially if we intend to apply this research in practice, on real software development projects.

One assumption that might threaten the applicability of our proposal in more complex systems is that of variable independence. Our proposed language (cf. § 2) represents how changes in single *parameters* affect single *indicators*, whereas in complex, adaptive systems, parameters (or indicators) cannot be assumed to be independent of one another. Nonetheless, this simplification is not accidental. State-of-the-art methods for modeling and controlling *multiple inputs/multiple outputs* (MIMO) control systems — such as state/output feedback and Linear Quadratic Regulator (see [28], § 3.4) — can be very complex and many software projects may not dispose of the necessary (human/time) resources to produce models with such degree of formality. As mentioned in the previous section, our approach is intended to be less heavy-handed in the formalism, while at the same time allowing analysts to model the requirements for the system's adaptation based on a feedback loop architecture.

Another considerable limitation of our current approach is the fact that its adaptation process responds to failures of single indicators (*AwReqs*) and does not consider the scenario in which multiple indicators fail concurrently and one failed indicator's adaptation might have an influence in another's. Procedures like *Ordered Side Effect Parameter Choice*, together with the specification of indicators' priorities (e.g., [17]), can help in avoiding undesirable situations such as focusing on less-critical failures or even deadlocks, but more direct consideration of concurrent failures is necessary to guarantee some level of consistency.

Therefore, we are currently working on extending *Qualia* by including a *priority queue* that would make the framework deal with more important failures first (in case, e.g., large *maturation times* create long-running adaptation cycles); the introduction of *locks* (as in database transaction processing) that would prevent certain parameters from being changed because they affect indicators that have been locked; and the ability of dealing with *multiple failures in a single adaptation loop*. The latter would require new procedure implementations, especially for the activities of *Parameter Choice* (e.g., choose parameters that do not have negative effects on all failed indicators), *Value Calculation* (e.g., considering multiple *increment coefficients*), *Waiting* (e.g., consider the *maturation time* of all failed indicators) and, obviously, *Indicator Evaluation*.

On the methodology side, improvements such as the elaboration of a graphical representation in the goal model might make the adaptation specifications easier to read; pre-defined policies can abstract the choice of adaptation algorithm and its many attribute values in mnemonics such as “aggressive”, “conservative”, etc.; moreover, a CASE tool would also greatly help analysts in following our proposed approach.

Finally, more experiments, especially with real systems, would help us examine the kinds of adaptation scenarios that are possible and, thus, propose sensible implementations for the *Algorithm Reassessment* and *Learning* activities, which have received little attention so far. These would involve a repository of past experiences, which would record failures, what was done to adapt and the outcome of the adaptation. Then, on-line or off-line learning procedures could query this repository in order to evolve the specification in general.

7 Conclusions

In this paper, we have proposed a framework within which a failure of requirements leads to a new behavior obtained by selecting a new variant of the system’s goal model, and/or new values for its control variables. The proposed controller is inspired by control theoretic concepts, notably the PID controller, recast in qualitative terms and using goal models to define the desired output and the space of possible behaviors for obtaining it. To validate our work, we have implemented our framework and simulated its algorithms using different scenarios.

Our proposal is founded on the thesis that requirements should be at the very center of any adaptation mechanism, determining what constitutes normal behavior, what is to be monitored and what are possible compensations in cases of deviations. Following Berry et al.’s *envelope of adaptability* [2], systems are only able to adapt to “the extent to which the adaptation analyst can anticipate the domain changes to be detected and the adaptations to be performed”.

Moreover, by separating the standard, “normal behavior” from the requirements for monitoring and adaptation, our approach provides abstractions that can facilitate modeling and communication of requirements for systems that are supposed to have several adaptation capabilities. As with any abstraction in Software Engineer, our proposals should be applied when the benefits of having these concepts in the models outweigh the cost of using the approach.

As the discussions illustrated earlier, the work presented here is the first step towards a full qualitative adaptation framework that can operationalize most stakeholder requirements for adaptation using a generic feedback loop.

Acknowledgments. We are grateful to our Trento colleagues for their feedback to this work, which has been supported by the ERC advanced grant 267856 “Lucretius: Foundations for Software Evolution” (unfolding during the period of April 2011 – March 2016) — <http://www.lucretius.eu>

References

1. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy Goals for Requirements-driven Adaptation. In: Proc. of the 18th IEEE International Requirements Engineering Conference, pp. 125–134. IEEE (2010)
2. Berry, D.M., Cheng, B.H.C., Zhang, J.: The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In: Proc. of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality, pp. 95–100 (2005)
3. Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., Shaw, M.: Engineering Self-Adaptive Systems through Feedback Loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) *Software Engineering for Self-Adaptive Systems*. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009)
4. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) *Software Engineering for Self-Adaptive Systems*. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
5. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Adaptive socio-technical systems: a requirements-based approach. In: *Requirements Engineering*, pp. 1–24 (2012)
6. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed Requirements Acquisition. *Science of Computer Programming* 20(1-2), 3–50 (1993)
7. Elahi, G., Yu, E.S.K.: Requirements Trade-offs Analysis in the Absence of Quantitative Measures: A Heuristic Method. In: Proc. of the 2011 ACM Symposium on Applied Computing, pp. 651–658. ACM (2011)
8. Filieri, A., Ghezzi, C., Leva, A., Maggio, M.: Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements. In: Proc. of the 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 283–292. IEEE (2011)
9. Filieri, A., Ghezzi, C., Leva, A., Maggio, M.: Reliability-driven dynamic binding via feedback control. In: *Private Communication* (2012)
10. Forbus, K.D.: Qualitative Reasoning. In: *Computer Science Handbook*, 2nd edn., ch. 62. Chapman and Hall/CRC (2004)
11. Fu, L., Peng, X., Yu, Y., Zhao, W.: Stateful Requirements Monitoring for Self-Repairing of Software Systems. Tech. rep., FDSE-TR201101, Fudan University, China (2010), <http://www.se.fudan.sh.cn/paper/techreport/1.pdf>
12. Heaven, W., Letier, E.: Simulating and Optimising Design Decisions in Quantitative Goal Models. In: Proc. of the 19th IEEE International Requirements Engineering Conference, pp. 79–88. IEEE (2011)
13. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: *Feedback Control of Computing Systems*, 1st edn. Wiley (2004)
14. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. *MIS Quarterly* 28(1), 75–105 (2004)
15. Jureta, I., Mylopoulos, J., Faulkner, S.: Revisiting the Core Ontology and Problem in Requirements Engineering. In: Proc. of the 16th IEEE International Requirements Engineering Conference, pp. 71–80. IEEE (2008)

16. Khan, M.J., Awais, M.M., Shamaal, S.: Enabling Self-Configuration in Autonomic Systems using Case-Based Reasoning with Improved Efficiency. In: Proc. of the 4th International Conference on Autonomic and Autonomous Systems, pp. 112–117. IEEE (2008)
17. Liaskos, S., McIlraith, S., Sohrabi, S., Mylopoulos, J.: Representing and reasoning about preferences in requirements engineering. *Requirements Engineering* 16(3), 227–249 (2011)
18. Menzies, T., Richardson, J.: Qualitative Modeling for Requirements Engineering. In: Proc. of the 30th Annual IEEE/NASA Software Engineering Workshop, pp. 11–20. IEEE (2006)
19. Nakagawa, H., Ohsuga, A., Honiden, S.: gocc: A Configuration Compiler for Self-adaptive Systems Using Goal-oriented Requirements Description. In: Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 40–49. ACM (2011)
20. Peng, X., Chen, B., Yu, Y., Zhao, W.: Self-Tuning of Software Systems through Goal-based Feedback Loop Control. In: Proc. of the 18th IEEE International Requirements Engineering Conference, pp. 104–107. IEEE (2010)
21. Salehie, M., Tahvildari, L.: Towards a Goal-Driven Approach to Action Selection in Self-Adaptive Software. *Software: Practice and Experience* 42(2), 211–233 (2012)
22. Silva Souza, V.E.: An Experiment on the Development of an Adaptive System based on the LAS-CAD. Tech. rep., University of Trento (2012), <http://disi.unitn.it/~vitorsouza/a-cad/>
23. Silva Souza, V.E., Lapouchnian, A., Mylopoulos, J.: System Identification for Adaptive Software Systems: A Requirements Engineering Perspective. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 346–361. Springer, Heidelberg (2011)
24. Silva Souza, V.E., Lapouchnian, A., Mylopoulos, J.: (Requirement) Evolution Requirements for Adaptive Systems. In: Proc. of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 155–164. IEEE (2012)
25. Silva Souza, V.E., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness Requirements for Adaptive Systems. In: Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 60–69. ACM (2011)
26. Wang, Y., Mylopoulos, J.: Self-Repair through Reconfiguration: A Requirements Engineering Approach. In: Proc. of the 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 257–268. IEEE (2009)
27. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.M.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In: Proc. of the 17th IEEE International Requirements Engineering Conference, pp. 79–88. IEEE (2009)
28. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., Shin, K.: What Does Control Theory Bring to Systems Research? *ACM SIGOPS Operating Systems Review* 43(1), 62 (2009)

Analyzing Design Tradeoffs in Large-Scale Socio-technical Systems through Simulation of Dynamic Collaboration Patterns

Christoph Dorn¹, George Edwards², and Nenad Medvidovic³

¹ Institute for Software Research, University of California, Irvine, CA 92697-3455
cdorn@uci.edu

² Blue Cell Software, Los Angeles, CA 90069, USA
george@bluecellsoftware.com

³ Computer Science Department, University of Southern California,
Los Angeles, CA 90089, USA
nen@usc.edu

Abstract. Emerging online collaboration platforms such as Wikipedia, Twitter, or Facebook provide the foundation for socio-technical systems where humans have become both content consumer and provider. Existing software engineering tools and techniques support the system engineer in designing and assessing the technical infrastructure. Little research, however, addresses the engineer's need for understanding the overall socio-technical system behavior. The effect of fundamental design decisions becomes quickly unpredictable as multiple collaboration patterns become integrated into a single system.

We propose the simulation of human and software elements at the collaboration level. We aim for detecting and evaluating undesirable system behavior such as users experiencing repeated update conflicts or software components becoming overloaded. To this end, this paper contributes (i) a language and (ii) methodology for specifying and simulating large-scale collaboration structures, (iii) example individual and aggregated pattern simulations, and (iv) evaluation of the overall approach.

Keywords: Design Tools and Techniques, System Simulation, Collaboration Patterns, Large-scale Socio-Technical Systems.

1 Introduction

During the last two decades, numerous web-based, large-scale collaboration services have emerged for social networking (e.g., Facebook), collaborative tagging (e.g., Digg), content sharing (e.g., Youtube), knowledge creation (e.g., Wikipedia), crowdsourcing (e.g., Amazon Mechanical Turk), and source code production (e.g., GitHub). Recent research efforts have analyzed these systems in terms of user incentives, participation, recruitment, decision making, and information management [18,5]. Engineered for diverse purposes, these systems differ widely in the underlying collaboration patterns of their users [6]. For example, Amazon

Mechanical Turk follows the *master/worker* pattern for task outsourcing, Facebook links people in *peer-to-peer* social networks, Wikipedia manages *shared artifacts* for collaborative editing, and Twitter provides *publish/subscribe* news distribution.

Engineers of such systems have currently little support for anticipating the system's overall (i.e., socio-technical) behavior in large-scale deployments, in terms of metrics such as the timeliness and load of messages, artifact changes, queries, and so on. For example, a system architect for a knowledge creation platform might be interested in the number of write conflicts given particular contributor characteristics. Similarly, a crowdsourcing platform architect needs to consider effects of task assignment strategies on task response time. A microblogging system architect may want to estimate the event propagation speed for a given user subscription topology.

In general, an engineer aims to avoid negative behaviors such as information overload, decision making based on stale data, accidental information disclosure, or performance bottlenecks. These behaviors manifest themselves both within user collaborations and within the software itself. Subscribing to many Wiki articles, for example, may flood the user with updates and simultaneously overload the software that aggregates change events (information overload). On the other hand, a code repository user who is unaware of updates submitted by other users may encounter numerous burdensome write conflicts when submitting changes (information scarcity). The presence of multiple, aggregated patterns within a single system further complicates the problem, as complex interactions result in undesirable emergent behaviors that cannot be detected by observing individual patterns in isolation. System designers thus need sophisticated analysis to identify such undesirable behavior and the conditions that create it.

Once engineers understand the implications of particular combination of user behavior, collaboration patterns, and system configuration, they can apply system constraints at design-time that prevent the undesired effects completely or devise run-time adaptation mechanisms that mitigate those effects dynamically. The resulting systems are more robust and may feature more coordination and awareness mechanisms that are well-understood and governed. Without such support, systems may be brittle or restricted in terms of the provided collaboration mechanisms. For example, the successful mass-collaboration systems mentioned above apply limits for technical or collaborative reasons.¹ We are, however, interested in the effect of design decisions beyond simple constraints.

The primary technical challenge addressed in this paper is reasoning about the expected emergent behavior in large-scale collaborative systems prior to implementation and deployment. To this end, we propose *simulating the behavior of web-scale collaboration systems* in terms of collaborator structures, their actions, and the supporting software infrastructure. Several prior research efforts target important but only partial aspects of this problem, and thus fall short of delivering collaboration-centric design support. Existing work on simulating workflows or

¹ Facebook has an upper limit of 5000 friend connections, Twitter places an initial follow limit of 2000, and Wikipedia enforces rate limits on write requests.

crowdsourcing, for example, provides valuable insights into performance-improving algorithms [14,17,22], but addresses only a single subdomain. Simulations of software architectures and their implementation [8,2,15] focus on the software level rather than human interaction. Finally, in the domain of statistical mechanics, simulation and analysis of large-scale social networks remains very abstract [11,1]. General guidelines [12] for facilitating collaboration and user participation provide a starting point for designing large-scale systems. They, for example, recommend enabling users to edit and share data, but are insufficient for determining the specific effects within a given collaboration environment.

The primary contribution of this paper is a principled method for analyzing complex collaboration architectures through simulation. In support of this contribution, the paper describes:

- enhancements to an existing language for modeling collaboration patterns that enable dynamic analysis (Sec. 3),
- example models of several individual collaboration patterns as well as a model of their composition in a single system,
- specific techniques for scoping and targeting simulations to yield the most useful results, and
- an evaluation of the overall approach demonstrating its feasibility and usefulness (Sec. 4).

The following section provides a motivating scenario (Sec. 2); with related work in Section 5 and conclusion and outlook in Section 6.

2 Motivating Scenario

Building monitoring and security requires extensive collaboration among members of a security team. These teams range in scope from a small group that monitors an office building to hundreds of personnel in back offices and on-site that monitor critical, geographically distributed infrastructure. Facility monitoring systems that enable large-scale, flexible collaboration are subject to diverse coordination requirements. In this paper, we will illustrate key concepts based on such an example system composing collaboration patterns found in Twitter, Wikipedia, and Amazon Mechanical Turk.

Consider a large-scale facility monitoring system (Fig. 1) involving several different user roles: *sensors*, *field agents*, *back-office agents*, *back-office analysts*, and *team leaders*. Sensors may be hardware and software components or people with “eyes on the ground.” Field agents are located on site and directly monitor data from sensors. Field agents are organized into teams assigned to a specific building, floor, or area. Field agents may send alerts of suspicious activity to back-office agents and flag relevant sensor data. Back-office agents investigate suspicious behavior by aggregating information and assessing whether a threat exists. Analysis of raw data from multiple video feeds, still images, and voice recordings may overwhelm an assigned back-office agent and require additional staff members on demand, in which case tasks can be assigned to a pool of back-office analysts. The team leader is responsible for determining the appropriate

response to an incident based on the aggregated, filtered information provided by back-office agents.

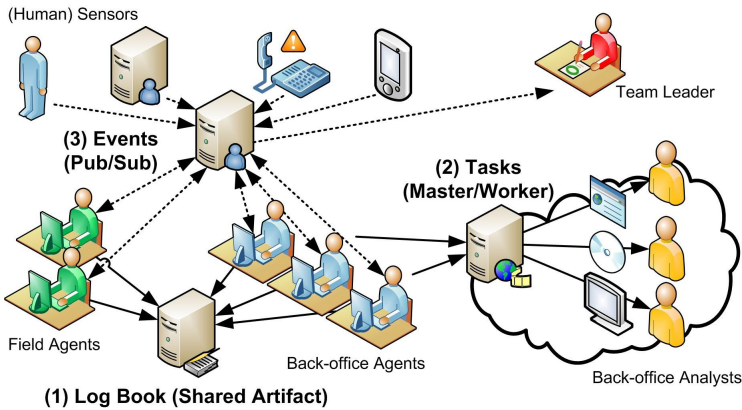


Fig. 1. Aggregating Collaboration Patterns for Infrastructure Monitoring

Interaction among the various users and user groups in this scenario exhibits several collaboration patterns. An engineer identifies following three potential collaboration patterns that match the underlying coordination requirements: **1**

Field agents will be provided with an interface for raising alerts and flagging sensor feeds that may indicate suspicious behavior. Back-office agents will select from a list of field agents from which they wish to receive alerts and flagged sensor data. Field agents need not be aware of which or how many back-office agents have selected to receive their alerts. Thus, information distribution will be achieved through the *publish/subscribe* collaboration pattern.

2 Virtual log books will be used to record the occurrence of events and user activities. Multiple log books may be created, each covering a different team, time-period, or subject. Users will be able to retrieve the latest log book on demand and make additions or modifications to it. Thus, the log books will implement the *shared artifact* collaboration pattern.

3 Back-office agents who require a threat assessment will add work items to a to-do list. Items in the to-do list will be automatically assigned to available back-office analysts, who will review the relevant sensor data and indicate whether a threat exists. In some cases, to minimize the potential for human error, the same task may be assigned to multiple analysts so that their conclusions can be compared for consistency. Thus, analysis tasks will be coordinated using the *master/worker* collaboration pattern.

The system design needs to achieve a balance of providing flexible, unrestricted collaboration mechanisms that facilitate staff members reacting to unforeseen situations while at the same time maintaining desirable system behavior. Staff members, for example, must not experience log book write conflicts, messages

and alerts need to be delivered to all interested parties but simultaneously avoid overloading the recipient, and tasks must finish in a timely manner and yield the required quality. The following questions highlight some specific issues the system designer might face:

1 How many field agents can a back-office agent reliably monitor before becoming overloaded? Should a limit be placed on how many field agents a back-office agent can select to monitor?

2 How should access to the log book be regulated to prevent write conflicts? Should staff members be required to obtain a lock before performing a write? If not, how often can we expect conflicts to occur?

3 How many users should share each log book? What happens if a large number of users are all trying to use the same log book?

4 How should tasks from the to-do list be allocated to available back-office analysts? First-in-first-out or some other way?

To answer all these questions, analysis of individual patterns is insufficient. The designer needs to consider system-wide, cascading implications such as the effect of event bursts on crowd-based situation assessment, the effect of overloaded crowd workers on timely event analysis, and the spike of write conflicts as staff members condense event observations into shared log books.

3 Modeling and Simulation of Collaboration Patterns

Large-scale collaboration systems heavily rely on humans as providers and consumers of information. Consequently, human behavior becomes an intrinsic aspect of the overall system. Given the inherent unpredictability of human behavior, static analysis techniques (e.g., [16]) are insufficient for making informed decisions about emergent behavior in a large-scale system. Instead, we focus on dynamic analysis in the form of system simulation.

In contrast to detailed modeling of software components, we propose simulating the interplay of humans and technology. In this context, the modeling effort focuses on collaboration patterns [6], such as master/worker, shared artifact, and publish/subscribe, rather than software architectures or architectural styles, such as SOA or 3-tier client-server [19]. Our approach treats collaboration patterns as “human architectures” consisting of people (*human components*) and the systems they use to facilitate collaboration (*collaboration connectors*). This achieves a clear distinction between work-centric roles (components) and coordination-centric roles (connectors), as described in our previous work [7]; emphasizes modeling of human interactions independently from the underlying design of the collaboration system used; and facilitates the identification of loci for system collaboration constraints.

Our process for simulating large-scale collaborations consists of three basic steps, usually performed in multiple iterations: (i) capturing collaboration patterns in an executable model, (ii) defining scenarios, assumptions, and configurations for individual simulation runs, and (iii) evaluating and interpreting

simulation results. These steps are described in Section 3.2. As a precondition, a suitable modeling language for capturing collaboration patterns must be specified. We have created such a language as an extension of the human Architecture Description Language (hADL [7], described in Section 3.1). This extended language is sufficiently flexible to allow engineers to model their own patterns or compositions of patterns, or they may extend the language further with pattern-specific elements or properties.

We used the DomainPro ² modeling and simulation tool suite to create models of collaboration patterns that conform to the extended hADL language and run simulations of those models. DomainPro enables engineers to create custom simulation languages through metamodeling and supports agent-based and discrete event simulation semantics. In the following sections, the model diagrams and simulation shown were all developed in DomainPro. However, our overall approach is generic and could be easily applied using a different simulation environment.

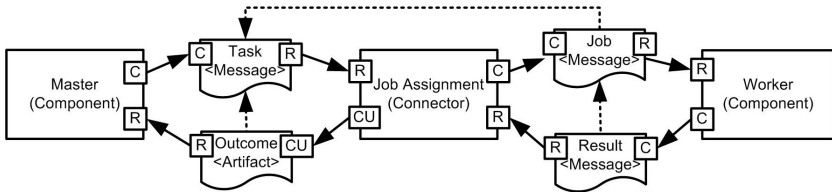


Fig. 2. Master/Worker pattern in hADL. Action labels represent CRUD privileges: Create, Read, Update, Delete. Collaboration objects exhibit optional references to related objects.

3.1 Modeling Language for Collaboration Patterns

In order to create simulation models for human collaboration, we extended an existing language, the human Architecture Description Language (hADL) [7], with additional features and properties to capture dynamic system behavior. As we will show, our extended hADL language can be used to capture a variety of individual and composite collaboration patterns.

We will briefly revisit the relevant parts of hADL as it represents the foundation for our simulation approach. In hADL, a collaboration pattern consists of two types of active entities: *human components* and *collaboration connectors*. A model of the master/worker pattern is shown in Fig. 2. The communication media used by components (e.g., Master, Worker) and connectors (e.g., Job Assignment) is represented by *collaboration objects*, such as messages (Task, Job, Result), streams, and artifacts (Outcome). *Human actions* and *object actions* restrict the interaction amongst components and connectors in terms of **C**reate, **R**ead, **U**ppdate, and **D**eleete manipulation capabilities. Connecting human actions and matching object actions gives rise to a collaboration pattern.

² <http://www.bluecellsoftware.com/>

Just as software-centric architecture description languages provide a high-level view of the system, hADL provides a view of human components, collaboration connectors, collaboration objects, and their wiring. However, execution of collaboration patterns remains outside of hADL’s scope, and hADL does not address the interdependencies amongst multiple patterns in a complex collaboration system.

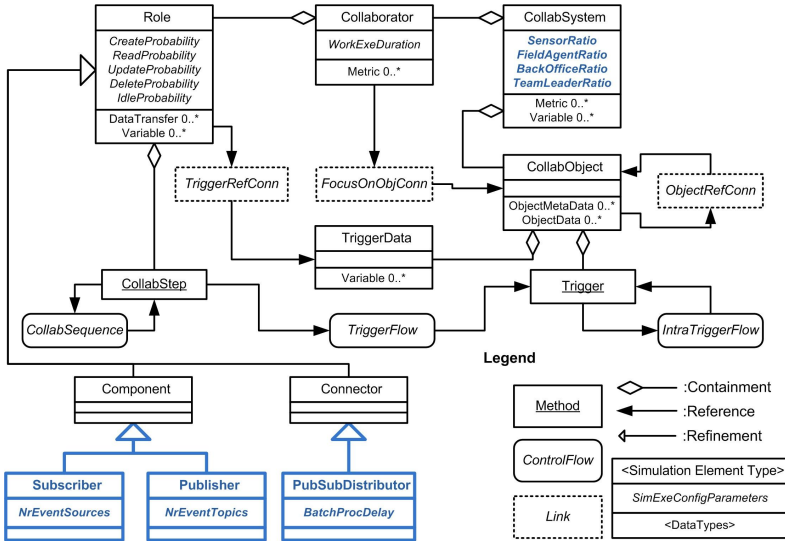


Fig. 3. The metamodel (extended hADL language) for defining collaboration patterns. Domain-specific extensions are highlighted in bold/blue.

Fig. 3 shows how hADL was extended to support dynamic behavior and simulation. A *CollabSystem* consists of *Collaborators*, which in turn play one or more *Roles*. A role may be either a *Component* or a *Connector*, thus enabling the *Collaborator* to assume a component role in one pattern and a connector role in another pattern. Each *Role* includes a set of *CollabSteps* that define the actions performed by the role. Each *CollabStep* represents a logical unit of work (e.g., sending a message, retrieving document content, processing a set of events). *CollabSequences* define the sequence (control flow) of *CollabSteps*. A *CollabSystem* also includes *CollabObjects*, the means of communication in hADL. *Triggers* represent events, such as timer timeouts, that initiate collaborator responses. Additional links (i.e., *TriggerFlow*, *FocusOnObjConn*, *TriggerRefConn*, and *ObjectRefConn*) provide object references to complete the core model.

The types enumerated above contain properties (shown in italics) that can be varied to achieve different simulation behaviors. For example, *CollabSteps* have an associated duration (time needed to complete), and *Roles* define probabilities for engaging in different optional behaviors. These types and properties can

be further extended with pattern-specific types or properties if engineers wish to investigate additional aspects of a pattern. Fig. 3, for example, highlights additional Component and Connector subtypes as well as Configuration properties for the Publish/Subscribe pattern in bold/blue. Note that we excluded extensions for the other patterns used in the scenario and evaluation for sake of clarity.

3.2 Modeling Collaboration Patterns

To develop a model of a collaboration pattern or composition of patterns, an engineer may utilize our extended hADL language or a pattern-specific variant of it. The engineer defines the structure and behavior of the pattern(s) by instantiating the appropriate components and connectors and defining their interactions. The model usually also includes a set of data structures that encapsulate the system’s dynamic state. Specifically, the simulation designer needs to identify:

Structure: Just as software engineering patterns [10] provide best-practises in programming, so do collaboration patterns [6] provide reusable structures of human components, collaboration connectors, actions, collaboration objects, and their relationships. Note that it is infeasible to explicitly model every Collaborator instance in large-scale systems. Hence, CollabObjects serve not only as carriers of information, but may also perform an addressing function (e.g., recording subscriptions to topics). Fig. 4 depicts the simulation model for a topic-centric publish/subscribe CollabSystem. The model combines both publisher and subscriber Roles within a single *Agent*. The *PubSubMW* Collaborator assumes the role of a collaboration connector for event delivery. *PubMsg* and *NfyMsg* provide the means of communication between Agent and PubSubMW.

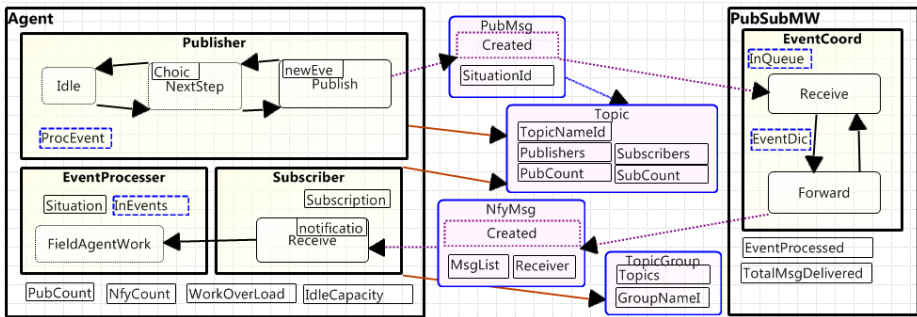


Fig. 4. Topic-centric Publish/Subscribe pattern in DomainPro Designer

Output: Behavior metrics provide engineers with information about the system’s dynamic behavior. For example, the Agent workload when processing incoming events may be of interest to engineers. For the PubSubMW, engineers may be concerned with the number of notification messages/events processed during various batch processing intervals.

Logic: The overall system behavior emerges from the interactions among individual Collaborators' behavior as well as the control flow among them. To capture timing behavior, the simulation requires each CollabStep to specify its duration, which may be a stochastic or random value. In Fig. 4, CollabSteps such as Publish, Receive, and Forward contain the logic to create, read, and process collaboration object content. Data is transferred between CollabSteps either directly as input or indirectly by adding data or objects to an inbox (e.g., the EventCoord's InQueue, or the EventProcessor's InEvents).

Environment: CollabSteps and CollabSequences can be parameterized to vary the simulation behavior. Example properties are the publisher's event fire rate, the middleware's delivery delay, and the subscriber's number of topics.

Having defined the simulation structure and behavior, a major challenge still remains before executing and analyzing the simulation model. Complex, large-scale collaboration systems typically exhibit a considerable set of configuration parameters, such as connectivity, work duration, action probabilities, and simulation duration, which quickly leads to an explosion in possible simulation configurations. We propose two main mechanisms for reducing the configuration space (which we then exemplarily demonstrate in the next subsection).

First, we suggest introducing dependencies between multiple configuration parameters. For example, all work durations can be defined as ratios of a core execution duration. This limits testing efforts to determination of sensible dependencies and limits simulation executions to a greatly reduced set of core parameters.

Second, we recommend the separate evaluation of independent patterns before aggregating them into the overall system. Doing so reveals the fundamental behavior and functional limits of a particular pattern, subsequently reducing the complexity of evaluating them as they are integrated into a composite system.

3.3 Scenario Model

Following the design methodology in the previous subsections, we provide one possible simulation model for the monitoring system from the scenario.

Structure: The motivating scenario introduced five user types: *sensors*, *field agents*, *back-office agents*, *back-office analysts*, and *team leaders* (recall Sec. 2). Fig. 5 depicts the interaction topology.³ Field agents publishing information about the same topic also update a common status report using the shared artifact pattern. They thus adopt the publisher role in the pub-sub pattern and the contributor role in the shared artifact pattern. Similarly, back-office agents exhibit roles in three different patterns: publisher and subscriber, contributor, and master. Back-office analysts perform the worker role in the master-worker pattern. Collaboration connector tasks such as event distribution, artifact access control, and task assignment are implemented in software.

Output: In the building monitoring system, one global performance metric is the duration between when a critical situation occurs and when it is recognized

³ A readable visualization of the overall simulation model would exceed this paper's page margins. View it at: <http://wp.me/P1xPeS-2F>

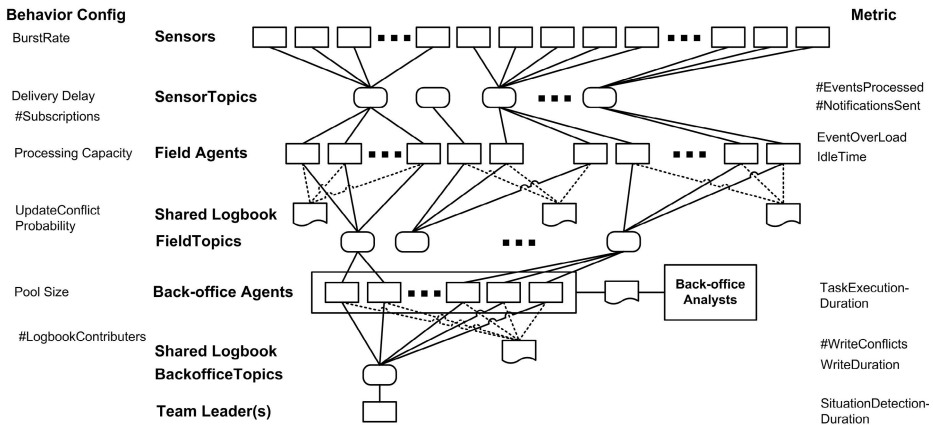


Fig. 5. Monitoring system simulation overview (simplified: collaboration connectors omitted for sake of clarity)

and responded to by the team leader. Suppose engineers are interested in the behavior under various load levels. The primary, external drivers of system load are the frequencies of sensor events, notification messages, analysis tasks, and log book updates. To make the simulation as realistic as possible, our model will include periods of regular, low-level activity interrupted by bursts of activity.

Using our approach, engineers can compare the effect of design decisions on system behavior, in terms of reaction to and recovery from increasing load levels. For example, we will show how engineers can examine the effect of (i) dropping events instead of processing all events, (ii) processing tasks in a last-in-first-out (LIFO) manner versus first-in-first-out (FIFO), and (iii) obtaining a write lock for the shared log book rather than updating on demand and resolving write conflicts later. Relevant metrics in the model capture agent load/idle time, task duration, time waiting to obtain a lock, and number of update conflicts. We discuss the system’s emergent behavior and associated metrics in more detail for each pattern in the following subsections.

Logic: The behavior of individual agents and roles in the model is specified as follows. Within each collaborator, we separate activities associated with each role played by the collaborator to allow for individual pattern analysis (recall that some collaborators play multiple collaboration roles). Yet, creating such composite patterns requires integrating control and data flow within a single collaborator. Whereas the particular mechanism depends on the application, in general events will trigger processing while data passes through “inboxes” to the other pattern. As shown in Fig. 4, the Subscriber role enqueues all received PubMsgs to the EventProcessor’s *InEvents* box and notifies the *FieldAgentWork* method about new events. The publisher component in turn will dispatch new PubMsg events from the EventProcessor via its *ProcEvents* box or otherwise idle and/or create a random new message on its own.

Environment: Modeling different design alternatives in separate models is not an option when aggregating multiple patterns, each having several configu-

ration options. Instead, we introduce parameters available for configuration at simulation time to switch between design variants. In the shared artifact pattern, for example, *DoArtifactLocking* determines whether *CollabSequences* to *GetLock*, *WaitLock*, and *ReleaseLock* methods (shaded) will be active or bypassed (Fig. 6).

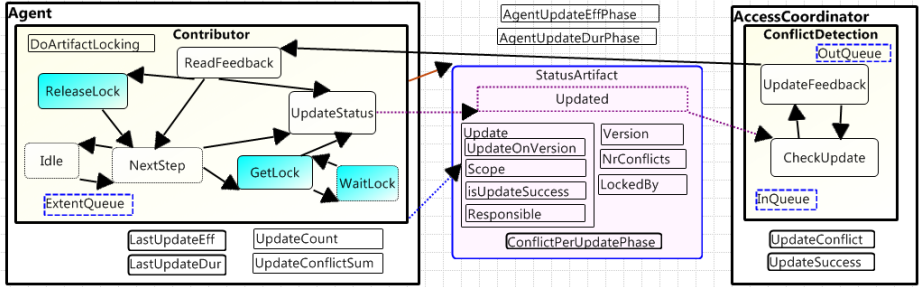


Fig. 6. Model excerpt for evaluating opportunistic write access and lock-based write access for updating a shared artifact

Following the recommendations in Section 3.2 on introducing configuration dependencies and thus minimizing the number of tuning parameters, all human execution methods derive their duration as a ratio of a core agent *WorkExecutionDuration* setting. Similarly, we defined the amount of topics and log books as a fixed rate of the number of sensors, respectively field agents.

4 Evaluation

The evaluation of our approach is two-fold. First, we show that modeling a complex large-scale collaboration system is indeed feasible (Sec. 4.1). Second, we demonstrate that our approach is beneficial during system architecture development. To this end, we analyze design decision trade-offs for individual collaboration patterns (Sec. 4.2) and for the composite pattern (Sec. 4.3) from the motivating scenario.

4.1 Feasibility

A feasible modeling approach should simultaneously facilitate simulations of small as well as complex models without involving considerable design overhead. We take the number of collaborators (*Coll*), components (*Comp*), connectors (*Conn*), collaboration steps (i.e., methods) (*CStep*), collaboration objects (*CObj*), collaboration sequences (*CSeq*), triggers (*Trig*), and trigger flows (*TFlow*) as an indicator for the modeling complexity. For the individual patterns we count only elements involved in the respective simulation run. Table 1 demonstrates that even a composite model needs only a few elements to model complex behavior. In our example, we minimized the number of collaborators by integrating all agent behavior in a single collaborator type.

Table 1. Model element count for individual and aggregated patterns. Note that each simulation contains elements for determining the active and calm event intervals.

Pattern	Coll	Comp	Conn	CStep	CObj	CSeq	Trig	TFlow
Publish/Subscribe	2	3	1	7	5	7	4	6
Master/Worker	3	2	1	7	3	7	4	6
Shared Artifact	2	1	1	9	2	14	5	6
Composite	5	6	3	25	8	36	9	15

4.2 Simulating Individual Patterns

Master/Worker Simulation: Suppose engineers wish to evaluate the effect of assigning tasks to back-office analysts in FIFO versus LIFO order. The master/-worker model measures the number of open tasks, the number of idle workers, and the task execution duration across time. We are interested in average task duration and also how predictable duration is (i.e., whether two sequential tasks tend to yield similar duration time).

Fig. 7 shows the data gathered from a simulation in which ten back-office agents (masters) each generate a task every time unit with 40% probability or otherwise idle. The pool of workers consists of twenty back-office analysts who require three time units (t) to complete a task. Task bursts are generated every $56t$ and last for $28t$. The simulation indicates whether the worker pool recovers from the added load in a timely manner (i.e, whether the recovery duration is less than the burst duration). During each burst, back-office agents double their task creation likelihood, and cut their idle time in half (resulting in a fourfold load increase). For each subsequent burst, the number of back-office agents in burst mode increases by 10%. Once 3000 tasks have been generated, no new tasks are created, allowing the analyst pool to finish all remaining tasks and obtain comparable metrics (FIFO and LIFO yield overall $15.1t$ average duration).⁴

Table 2. Average task duration (dur) and average sequential duration difference (diff) for FIFO and LIFO for the three phases of Fig.7.

		FIFO		LIFO		
Interval (t)		dur	diff	dur	dur < 100	diff
Phase 1	Low 0-112	3.04	0.06	3.04	3.04	0.06
Phase 2	Med 113-280	7.57	0.25	10.21	6.92	6.51
Phase 3	High 281-450	26.14	0.29	24.04	12.44	13.82

FIFO and LIFO perform equally well for a low task load (Phase 1). Towards the end of Phase 2, the worker pool reaches its recovery limit, as it cannot process the load received during the burst phases rapidly enough. Note that LIFO

⁴ Due to page constraints, we cannot display the very similar behavior observed for 1,000 agents, 2,000 analysts, and 300,000 tasks.

processing results in the first few tasks (duration $> 100t$) to be completed during simulation “cool down.” Considering all tasks, FIFO appears to be superior to LIFO in this scenario. However, tasks that remain unprocessed for a very long time may become irrelevant. For example, if we ignore task as useless after $100t$, LIFO provides better average task duration for relevant responses. (Table 2: $\text{dur} < 100$). On the downside, LIFO causes large fluctuations in the durations of sequential tasks (Table 2 diff), regardless of load level. A task completed in minimum time might be followed by a task that potentially expires.

The simulation highlights various control parameters to steer the system behavior under load. Besides the main choice between FIFO and LIFO, engineers may decide to limit task creation rates, limit the number of back-office agents, increase the size of the analyst pool, or enforce task expiration dates. Evaluation of more sophisticated mechanisms such as task priorities, variations of worker performance, or dynamic switching between FIFO and LIFO merely requires some additional modeling work.

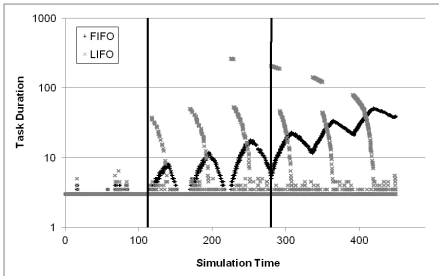


Fig. 7. Task execution duration for FIFO and LIFO with increasingly intense burst intervals

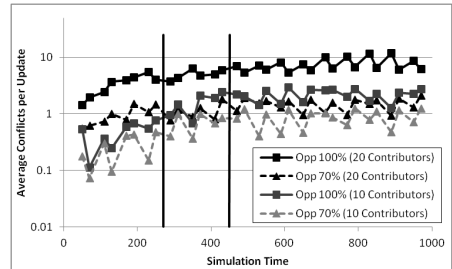


Fig. 8. Average number of conflicts until successful update

Shared Artifact Simulation: Use of a common logbook by multiple agents in the monitoring system raises the question of how to control write access. Engineers face a trade-off between lock-based access and opportunistic access with conflict detection. The former guarantees write success and constant write effort, but may cause long wait times to obtain the lock. The latter promises immediate write access at the chance of creating write conflicts, potentially imposing additional work to resolve conflicts. Relevant metrics in the shared artifact model (Fig. 6) capture the average duration to successfully complete an update and the average probability of a conflict occurring during an update.

In the shared artifact simulation, a set of agents (playing the role of contributors) access a single logbook. Periodically, each agent updates the logbook with 20% probability or idles (i.e., works on something else) for $6t$. An update attempt takes $1t$ and specifies the last known artifact version and the update scope. The scope $([0; 100])$ describes the extent of the update (e.g., the portion of the logbook modified) and is equivalent to the likelihood of a conflict among different versions (0% = never conflicting, 100% = always conflicting). The system

behavior is simulated under fluctuating load, exhibiting bursts lasting $40t$ followed by periods of baseline behavior lasting $20t$. The remaining burst behavior configuration and generation is identical to the master/worker simulation above. Fig. 9 shows the update duration for twenty and ten contributors, respectively.

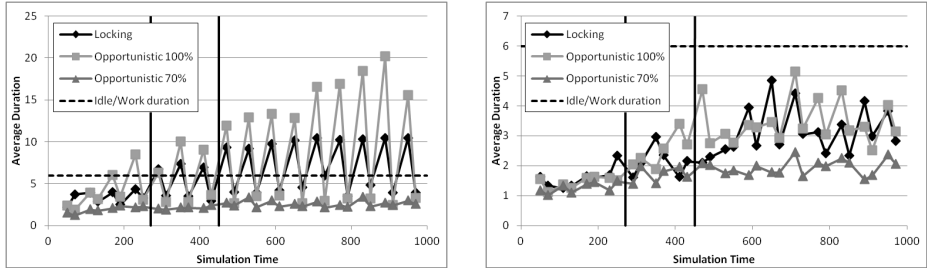


Fig. 9. Update duration for 20 (left) and 10 (right) contributors per artifact for lock-based and opportunistic access at 100% and 70% conflict likelihood

The two strategies perform similarly at low load as lock waiting times remain low and conflicts are infrequent. At medium load opportunistic updates show significant duration benefit for 70% conflict likelihood. Note that with conflict probability of 70% and 20 contributors, however, every update still fails on average more than once, which for humans is typically considered unacceptable (Fig. 8). At high load, neither strategy remains sensible. Locking for this configuration of contributors and write access duration hits its theoretical limit, in which agents are constantly waiting for a lock (i.e., average update duration of 10). Opportunistic access even yields average update durations well beyond twice the regular idle duration.

Engineers might consider some or all of the following options in this scenario: (i) (dynamically) restricting the number of contributors per artifact, (ii) enforcing update rate limits, and/or (iii) facilitating shorter access durations. Fig. 9 (right) highlights how reducing the contributor base (compared to Fig. 9 (left)) results in a disproportionately large reduction in duration and conflicts. Even then, with 10 contributors, 70% conflict probability produces one conflict per update on average during burst intervals (Fig.8).

4.3 Simulating Composite Patterns

Having gained an understanding of the master/worker and shared artifact patterns individually, we now focus on their aggregation, along with the pub/sub pattern, to assess the complex system described in Section 4.1. Specifically, our composite model simulates the effect of activity bursts on the analyst pool and log book to reveal the impact on the time required to detect unsafe or insecure situations. A typical simulation run for this scenario involves 600 collaborators, 100k events, 10k tasks, and 8k artifact updates within $1000t$.

Activity bursts in the composite model are identical to those in the previous simulations. Sensor events are tagged with a different situation ID for each $40t$ interval of base or burst behavior. We measure the duration from the beginning of an interval until the time the team leader first detects the ID from incoming events. The load on field agents determines the scope of logbook updates (i.e., the conflict likelihood) and the number of outgoing events. Those events result in tasks being assigned to back-office analysts for processing before being propagated to the team leader by back-office agents.

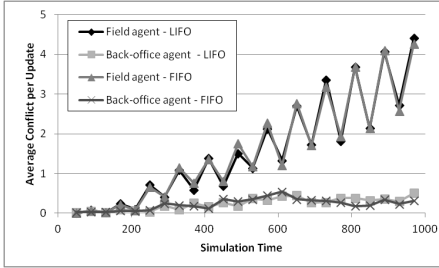


Fig. 10. Access conflicts for field agents and back-office agents

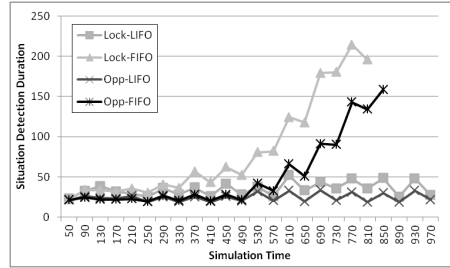


Fig. 11. Situation detection duration for lock-based and opportunistic access

Agents update the log-book after each iteration of event processing. Whereas lock-based access allows the agent to continue working while waiting, opportunistic access requires full attention and delays event processing while the update is occurring. Lock-based access requires the agent to complete each update before attempting a subsequent one.

In the composite model, field agents operate at the lock-based limit: 10 agents per artifact, updating about every $5t$, with update duration $1t$. As Fig. 11 highlights, the lock waiting time directly impacts timely situation detection. However, opportunistic access comes at the cost of rising conflicts for field agents with increasing event load. Back-office agents are less affected as only four share a single artifact and are additionally shielded from high load by previous field agent processing (Fig. 10).

Given the level of redundant events, LIFO clearly outperforms FIFO for medium and high load (Fig. 11). The increasing task load delays situation detection with FIFO to such extent, that the last few situations remain undetected within the simulation time.

In summary, the composite system simulation supports system design decisions by highlighting (i) the impact of access strategies on the detection duration, (ii) the impact of task queue style on situation detection success, (iii) the bottlenecks (i.e., shared artifacts for field agents rather than back-office agents), (iv) and the potential for dynamically switching between strategies. For example, applying FIFO and opportunistic access at low load and switching to LIFO and lock-based access at high load.

Limitations: A simulation can only give detailed recommendations about the optimum expert pool size, or the optimum number of experts per artifact for precisely defined models of individual patterns. Simulations of complex socio-technical systems can only cover particular aspects of interest, never all details. Thus any results in terms of absolute numbers are unsuitable to be applied directly in a real world systems. Instead, the simulation enables system engineers to compare the impact of different design decisions and decide what trade-offs need to be made. The simulation outcome provides an understanding what mechanisms might fail earlier, which strategies behave more predictably, and which configurations result in a more robust system. At the same time, a simulation raises awareness of system metrics that are best suited for serving as indicators of looming performance deterioration.

5 Related Work

Simulating system aspects for gaining an understanding of its behavior has been proposed in many diverse areas. Scenario-driven dynamic analysis of distributed architectures enables the system architect to compare design trade-offs [8]. Extensions to UML models such as sequence diagrams allows for tracking of performance metrics [2]. Simulation ranges from modeling individual software components [4] to large-scale service oriented systems [15] for the prediction of system reliability or the development of adequacy criteria and test cases for distributed systems [21]. Simulation of business process and workflows aims for detecting bottlenecks, predicting cost and time, evaluating quality and flexibility, and determining other performance metrics [14,23,17]. Research on crowdsourcing applies simulation to demonstrate the effect of assessment tasks on skill evolution [22], to evaluate the impact of collaboration policies [20], or determine the optimum number of replicated jobs per task [3].

These research efforts target important but only partial aspects of socio-technical systems. Focusing only on a subdomain or only on the technical part, they thus fall short of delivering collaboration-centric design support.

Social network analysis observes and analyzes general emerging system properties such as the power-law network topology [1] as well as system specific properties such as the structure of discussion threads on Slashdot [11]. Such research provides insights on how to simulate realistic structure and behavior of large-scale collaborative efforts.

On the small scale end of the spectrum, team automata formalize the interactions amongst multiple participants in groupware systems [9]. Although team automata initially targeted Computer Supported Cooperative Work (CSCW) systems to rigorously define and enforce collaboration protocols [16], their nature lend them more to the analysis and design of security mechanisms. We believe that team automata are unsuitable for simulating socio-technical systems where the exact participant behavior is a-priori unknown. The *Construct* group simulation tool [13] overcomes these limitations but remains severely restricted in the maximum amount of simultaneously active collaborators.

6 Conclusions

This paper presented a method for simulating complex collaboration structures in support of understanding system design trade-offs. We extended the human Architecture Description Language to obtain an executable model of collaboration patterns. Exemplary simulations of the master-worker pattern, the shared-artifact pattern, and their integration with the publish-subscribe pattern demonstrate feasibility and benefit to the system designer.

Future work will focus on exploring these patterns in more detail and applying the simulation framework for evaluating dynamic switching between strategies (e.g., *FIFO* \Leftrightarrow *LIFO*) at runtime and evaluate our work in the scope of a real world system. Simultaneously we intend to include additional human component aspects such as learning, forgetting, skills, trust, or social connections.

Acknowledgment. This work is supported in part by the Austrian Science Fund (FWF) under grant number J3068-N23.

References

1. Albert, R., Barabasi, A.L.: Statistical mechanics of complex networks. *Reviews of Modern Physics* 74, 47 (2002)
2. Balsamo, S., Marzolla, M.: Simulation modeling of uml software architectures. In: Proc. of ESM 2003, the 17th European Simulation Multiconference, pp. 562–567. SCS–European Publishing House (2003)
3. Brun, Y., Edwards, G., Young Bang, J., Medvidovic, N.: Smart redundancy for distributed computation. In: Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS 2011), pp. 665–676 (2011)
4. Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L.: Early prediction of software component reliability. In: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 111–120. ACM, New York (2008)
5. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. *Commun. ACM* 54, 86–96 (2011)
6. Dorn, C., Taylor, R.N.: Analyzing runtime adaptability of collaboration patterns. In: International Conference on Collaboration Technologies and Systems (CTS). IEEE Computer Society, Los Alamitos (2012)
7. Dorn, C., Taylor, R.N.: Architecture-Driven Modeling of Adaptive Collaboration Structures in Large-Scale Social Web Applications, Tech. Rep. UCI-ISR-12-5, University of California, Irvine (May 2012), http://www.isr.uci.edu/tech_reports/UCI-ISR-12-5.pdf
8. Edwards, G., Malek, S., Medvidović, N.: Scenario-Driven Dynamic Analysis of Distributed Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 125–139. Springer, Heidelberg (2007)
9. Ellis, C.: Team automata for groupware systems. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, GROUP 1997, pp. 415–424. ACM, New York (1997)
10. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)

11. Gómez, V., Kaltenbrunner, A., López, V.: Statistical analysis of the social network and discussion threads in slashdot. In: WWW 2008: Proceeding of the 17th International Conference on World Wide Web, pp. 645–654. ACM, NY (2008)
12. Gregg, D.G.: Designing for collective intelligence. *Commun. ACM* 53, 134–138 (2010)
13. Hirshman, B.R., Carley, K.M., Kowalchuck, M.J.: Specifying Agents in Construct. Tech. Rep. CMU-ISRI-07-107, Carnegie Mellon University (July 2007)
14. Hlupic, V., Robinson, S.: Business process modelling and analysis using discrete-event simulation. In: Proceedings of the 30th Conference on Winter Simulation, WSC 1998, pp. 1363–1370. IEEE Computer Society Press, CA (1998)
15. Juszczyk, L., Dustdar, S.: Script-based generation of dynamic testbeds for soa. In: Proceedings of the 2010 IEEE International Conference on Web Services, ICWS 2010, pp. 195–202. IEEE Computer Society, Washington, DC (2010)
16. Kleijn, J.: Team Automata for Csw - a Survey. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) *Petri Net Technology for Communication-Based Systems*. LNCS, vol. 2472, pp. 295–320. Springer, Heidelberg (2003)
17. Li, J., Fan, Y., Zhou, M.: Performance modeling and analysis of workflow. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 34(2), 229–242 (2004)
18. Malone, T.W., Laubacher, R., Dellarocas, C.: Harnessing crowds: Mapping the genome of collective intelligence. *Technology* 1(2), 327–335 (2010)
19. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Companion of the 30th International Conference on Software Engineering, pp. 899–910. ACM, New York (2008)
20. Psailer, H., Skopik, F., Schall, D., Juszczyk, L., Treiber, M., Dustdar, S.: A programming model for self-adaptive open enterprise systems. In: Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing, MW4SOC 2010, pp. 27–32. ACM, New York (2010)
21. Rutherford, M.J., Carzaniga, A., Wolf, A.L.: Evaluating test suites and adequacy criteria using simulation-based models of distributed systems. *IEEE Trans. Softw. Eng.* 34(4), 452–470 (2008)
22. Satzger, B., Psailer, H., Schall, D., Dustdar, S.: Stimulating Skill Evolution in Market-Based Crowdsourcing. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) *BPM 2011*. LNCS, vol. 6896, pp. 66–82. Springer, Heidelberg (2011)
23. Tumay, K.: Business process simulation. In: *Simulation Conference Proceedings*, pp. 55–60 (December 1995)

Semantic and Locality Aware Consistency for Mobile Cooperative Editing*

André Pessoa Negrão, João Costa, Paulo Ferreira, and Luís Veiga

INESC-ID/Instituto Superior Técnico
Rua Alves Redol 9, Lisboa, Portugal
{andre.pessoa,joao.da.costa}@ist.utl.pt,
{paulo.ferreira,luis.veiga}@inesc-id.pt

Abstract. This paper presents CoopSLA (Cooperative Semantic Locality Awareness), a consistency model for cooperative editing applications running in resource-constrained mobile devices. In CoopSLA, updates to different parts of the document have different priorities, depending on the relative interest of the user in the region where the update is performed; updates that are considered relevant to the user are propagated frequently, while less important ones are postponed. As a result, the system makes a more intelligent usage of the network resources, since 1) fewer accesses to the network are issued, 2) bandwidth savings are obtained by merging the delayed updates, and 3) reduced bandwidth available is used more efficiently by propagating more relevant updates sooner. These properties are of vital importance in the mobile environments we are addressing, in which devices have limited bandwidth and battery power. We have implemented a collaborative version of Text editor TextMaker using the CoopSLA approach. We present evaluation results that support our claim that CoopSLA is very effective in reducing the overhead of replica synchronization without imposing limitations to application models.

Keywords: Cooperative Editing, Optimistic Replication, Data Consistency, Interest Management, Divergence Bounding.

1 Introduction

Cooperative editing applications enable geographically distributed users to concurrently edit a shared document space over a computer network[3]. Recently, these applications experienced an increase in popularity as a result of the expansion of the Internet and the rapid proliferation of mobile devices, such as smart phones, PDAs and tablets[10]. These modern devices are now sophisticated enough to allow its users to execute cooperative editing applications and participate in editing sessions alongside more powerful devices – like desktops or laptops – possibly mediated by cloud infrastructures.

* This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113993/2009 and PEst-OE/EEI/LA0021/2011.

A critical technique to support these new heterogeneous environments – that mix resource constrained and powerful devices, interacting over wired and wireless networks – is to replicate the application data at the users’ devices and resort to optimistic protocols to manage the consistency of the shared state. Optimistic replication[12] has the potential benefit of improving performance, availability and usability by allowing faster (local) access to the data. It also makes a more efficient use of the resources since it does not require constant access to the network for synchronization purposes. Optimistic mechanisms have been extensively applied to cooperative editing, in particular the Operational Transformation paradigm[2,15,14,7] and, more recently, Commutative Replicated Data Types[9,8,18,11,19].

While the state-of-the-art solutions provide a fair compromise between consistency and performance, they still neglect two important aspects that can be leveraged to improve the performance and overall usability and experience of cooperative editing applications. First, they do not consider the variable and highly dynamic characteristics of group work, in which different users are interested (and work on) different parts of the document space: i) a user is more interested in the zone(s) of the document that he is editing and a few other *observation points*, rather than the whole document space equally, and ii) his interest in the different sections of the document space varies over time. Second, optimistic systems based on eventual consistency are typically prone to some level of uncertainty and disruption: while the system is ensured to converge in the future, there is limited or no support to determine how current is the data observed by the user, and to establish and enforce clear bounds or guarantees on that currentness.

In this work we argue that it is possible to make a more efficient and scalable usage of the network resources by taking the users’ interest into account. To address this issue, we propose CoopSLA (Cooperative Semantic Locality Awareness), a consistency model that unifies several well-know concepts of the distributed systems field: interest management, locality-awareness and bounded divergence. In CoopSLA updates are assigned a per-user priority level based on its *semantic distance* to the user’s *observation point(s)* (Interest Management). Updates to regions closer to the observation points are considered more relevant and, thus, are awarded higher priority; priority decreases as the distance to the observation point increases (Locality-Awareness). In each priority level, updates are managed according to a parametrizable, multidimensional consistency space that determines when they must be propagated, establishing clear and well-known bounds to the divergence between the different replicas of the system (Bounded Divergence).

With CoopSLA, we are able to make a more intelligent and semantically meaningful usage of the network resources: by postponing updates with less priority, the system can merge and aggregate them, minimizing accesses to the network and reducing bandwidth; as a result of message aggregation, the latencies of the more important messages are reduced at the expense of the less relevant ones. These properties are of particular importance when mobile devices are in use,

because wireless networks provide low bandwidth with high latency, which has a significant impact on performance and interactivity[6], and frequent access to the network resources greatly increases battery consumption[4]. Another important aspect of CoopSLA is that it establishes bounds on the amount of replica deviation allowed, providing users with stronger guarantees regarding the actual consistency state of the document space.

We designed and implemented a middleware layer that enforces the CoopSLA model on behalf of the applications. This allows current single-user applications to be more easily adapted to support collaborative features and relieves programmers from the daunting task of designing and programming complex network and replication protocols. Using the CoopSLA middleware, we implemented a collaborative version of the popular Tex editor *TexMaker*. We present experimental results that support our claim that CoopSLA is very effective and flexible in reducing the overhead of replica synchronization without imposing limitations to application models and traditional semantics.

The paper is organized as follows. Section 2 introduces relevant concepts and describes the main assumptions of our work. Section 3 describes the CoopSLA consistency model in detail. Section 4 presents the main architectural aspects of the CoopSLA middleware. Section 5 overviews the implementation of our solution. Section 6 presents and discusses the experimental results. Related work is presented in Section 7. Finally, Section 8 concludes the paper.

2 System Model

In a cooperative editing session, multiple geographically distributed users concurrently edit a shared *document space* – for example, a Latex project, a wiki or a Word document. Common to these scenarios is a hierarchical structure of *semantic regions*, which are logical sub-divisions of the document space (like a folder, a file, or a `\section` of a Latex document). Also, semantic regions may have logical references to other regions (e.g., a Latex `\ref` or a link on a webpage). When considering the interest of a user, both the structure of the document space and the references between its components must be taken into account.

We model the document space as a rooted directed graph $G = (V, E)$. Each vertex V corresponds to a *semantic region* of the document space and there is an edge (v_i, v_j) between vertices v_i and v_j iff there is a *relation* between the two. We consider two types of relations: a *structural* edge connects two vertices that have a parent/child relation that is part of the hierarchical organization of the document (in a Latex document, for example, a `\chapter` has a structural relation with each of its child `\section`); a *semantic* edge is a non-structural edge that connects two vertices that have an application-specific reference between them (e.g., a `\ref` in a Latex document or a link in a web page). Structural edges define a subgraph S of G corresponding to the tree structure of the document space.

We define a function $d : V \times V \rightarrow \mathbb{N}_0$ over the graph that represents the *semantic distance* between two vertices of the graph. The semantic distance

indicates the degree of correlation between two semantic regions of a document according to some application-dependent and user-aware criteria. A lower value denotes high correlation, while a higher value denotes low correlation.

We denote the participants of a cooperative editing session as *cooperation group* and each participant is called *node*. Each node of the cooperation group has a local *view* consisting of a full *local replica* of the shared application state that may have bounded inconsistencies with relation to the latest state of the application. Each replica consists of the document space graph G in which each vertex also holds the contents of the corresponding semantic region. In this context, we refer to a vertex of the graph as an *object*. Each object has one primary/master replica that holds its most recent value and one or more secondary replicas that may have stale values. The consistency model makes no restrictions as to which nodes of the cooperation group can be the master of an object.

3 A Semantic and Locality Aware Consistency Model

To capture the interest of a user in the different regions of a document, the CoopSLA model incorporates the notion of a *pivot*. A pivot is a special object that corresponds to a user's observation point and according to which the consistency requirements of the user's view is managed. Broadly speaking, the pivot determines, on a per-user basis, i) when an update to an object is allowed to be postponed, and ii) under which conditions previously postponed updates are required to be propagated to the user.

Different users have different pivots and each user may have multiple pivots, each corresponding to a semantic region in which he is interested. In this section, we describe the pivot-based CoopSLA consistency model in detail. For clarity, we first describe the main concepts of the CoopSLA model considering only one pivot (Sections 3.1 and 3.2). We briefly describe a generalization of the model for multiple pivots in Section 3.3.

3.1 Consistency Field

Each user's pivot p has a position in the application graph ($p \in V$). p can change throughout the execution of the application, mirroring the dynamic interest of the user in the document space. Moreover, a pivot generates a discrete consistency-field composed of n *consistency zones* z_1, \dots, z_n where: z_1 is the inner zone of radius r_1 and centered in p ; each zone $z_i \in \{z_2, \dots, z_{n-1}\}$ corresponds to the area with outer radius r_i and inner radius r_{i-1} (the radius of zone z_{i-1}); the outer zone z_n corresponds to the area beyond r_{n-1} , the radius of zone z_{n-1} . It follows that the consistency zone z_o of object o at semantic distance $d(p, o)$ from pivot p in the consistency-field is given by

$$z_o = \begin{cases} z_1 & \text{iff } d(p, o) \leq r_1 \\ z_i, 1 < i \leq n - 1 & \text{iff } r_{i-1} < d(p, o) \leq r_i \\ z_n & \text{iff } r_{n-1} \leq d(p, o) \end{cases} \quad (1)$$

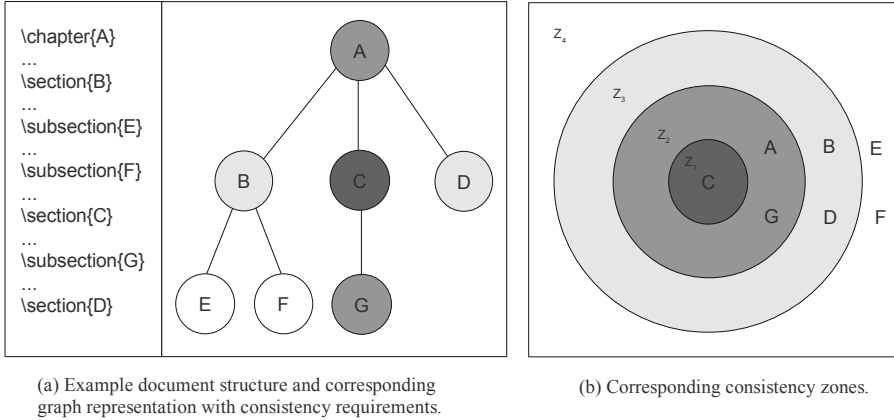


Fig. 1. Example of Consistency Zones in a document structure

Figure 1 shows a simple example of a consistency field. The user’s descending order of interest, based on which the consistency field is defined, is described in Table 1; the table explains how, taking into account current pivot position, each section of document is mapped to a given consistency zone.

In this example, the user is editing section C of a Latex document (left side of Figure 1(a)); as a result, the pivot is placed in the corresponding C vertex of the application graph (right side of Figure 1(a)). The consistency field generated (left side of Figure 1(b)) assigns the highest priority to updates to section C, the current editing section. Updates to sections A e G (respectively, the parent and child sections of C) have lower priority than C, but higher than sections B and D – the sections that are two graph hops away from C. Updates to section E and F have the lowest priority.

3.2 Consistency Requirements

Each consistency zone z_i has a corresponding *consistency degree* c_i that specifies the consistency requirements of the objects located within that zone. Consistency degrees respect the property $c_i > c_{i+1}$, meaning that consistency degree c_i of consistency zone z_i enforces stronger consistency than degree c_{i+1} of zone

Table 1. Example user interest

Priority	Description	Zone
1	Current editing section	z_1
2	Parent and child sections of current editing section	z_2
3	Close sections – sections two graph hops away.	z_3
4	Remaining sections	z_4

z_{i+1} . It follows from this property that consistency degrees (and, consequently, requirements) become weaker as their semantic distance to the pivot increases. Consistency degrees are described by 3-dimensional *consistency vectors* (κ) that limit the maximum divergence between the local replica of an object and the latest state of the object:

- *Time* (θ): Specifies how long (in seconds) an object is allowed to remain without being refreshed with its latest value.
- *Sequence* (σ): Defines the maximum number of updates to an object that are allowed to be postponed (missing updates).
- *Value* (ν): Specifies the maximum percentage divergence between the contents of the local replica of an object and its primary replica. *Value* is an application-dependent metric calculated by a special purpose function defined by the application’s programmers.

CoopSLA guarantees that an object is updated whenever at least one of the previous criteria is about to be violated. Consider, for example, a consistency vector $\kappa = [0.25, 6, 20]$; an object within the consistency zone corresponding to κ is guaranteed to be, at most, 0.25 seconds outdated, 6 updates behind the primary replica or with contents diverging 20% from the object’s latest value.

3.3 Model Generalization

CoopSLA allows the definition of multiple pivots for each user. For example, if a user is editing multiple files, each one of the editing points in the different files may correspond to a different pivot. Furthermore, different pivots may have different consistency requirements, as it is natural that the current editing point is more relevant to the user. In a multi-pivot setup, an object’s consistency zone is assigned with relation to its closest *pivot*.

The model also allows the definition of multiple views per user, which allows different sets of objects to be characterized with different consistency requirements regarding the same *pivot*. Consider a pivot that corresponds to the paragraph currently being edited by the user. In this scenario, a user may be more interested in sections he created than in sections created by other users, regardless of how close they are to the user. With multiple views, user created objects may be assigned more strict consistency requirements.

4 Architecture

CoopSLA is implemented by a middleware layer that abstracts the programmers from the aspects related to network communication and consistency enforcement. The middleware follows a client-server architecture, in which clients edit the shared document space and the server propagates updates to each client according to its consistency specifications. In this section we describe the main architectural aspects of the CoopSLA middleware. We start by presenting an overview of the system, after which we describe how it represents the document space internally and how the consistency model is enforced.

4.1 Overview

Following the CoopSLA replication model (see Section 2), the server holds a full replica of the shared application state (i.e., the application graph G). The server stores the primary replica of every object in the system and, thus, always has the most recent version of the objects. When the server receives a client update it applies it to its primary replica immediately; in contrast, it postpones propagating the received updates to the clients, as long as their consistency requirements are respected.

A client consists of the editing application stacked on top of the middleware. It receives the input from the user, applies it to its local replica and submits the corresponding update to the server. Clients do not communicate directly with each other; update propagation is exclusively performed by the server. Each client holds a full replica of the data; however, unlike the server, these are secondary and, as a result, may have stale values that are managed according to each client's consistency specification.

The main task of the server is to enforce the CoopSLA consistency model. This requires it to continuously monitor client updates and collect information about the current consistency state of each client. Periodically, and when updates are received at the server, it executes a validation algorithm that uses the collected data to verify if the consistency requirements of the clients are still met; if not, the server propagates the postponed (possibly merged) updates to the clients that would, otherwise, violate their consistency specification.

4.2 Data Representation

The CoopSLA middleware represents the contents of each object of the graph as a TreeDoc Commutative Replicated Data Type (CRDT)[9]. By doing so, we enable replicas to converge without the need for complex conflict resolution protocols, which further enhances the relaxed synchronization properties provided by CoopSLA. As a result of using TreeDoc, our system follows an *operation transfer* design. This means that the update messages exchanged during an editing session consist of *add* or *remove* operations, instead of the actual data.

Updates that have not yet been propagated to a client are stored at the server in a per-client *update queue*. When adding a new update to a queue, the server automatically merges add/remove operations that cancel each other. Even by just using this mechanism, the results (Section 6) proved to be very encouraging. Alternative (or complementary) merging solutions are still being implemented and are out of the scope of this paper.

4.3 Monitoring Client Activity

To enforce the consistency model the server stores, for each client c_i , the client's consistency specification (pivots, zones and degrees) and *consistency state* table ψ_{c_i} . The latter stores, for each object o_i : 1) the time elapsed since c_i last received updates regarding o_i ($\psi_{c_i}[\theta, o_i]$), 2) the number of updates to o_i that have not

yet been sent to c_i ($\psi_{c_i}[\sigma, o_i]$), and 3) the *value* of o_i the last time updates to it were sent to c_i ($\psi_{c_i}[\nu, o_i]$).

Enforcing each client's consistency specifications requires the server to keep track of the following critical events:

Content Updates. When the server receives a content update (and *add* or *remove* request) it adds it to the update queues of the clients and updates the *sequence* state $\psi_{c_i}[\sigma]$ of every client c_i . Next, it verifies if the new value of the *sequence* metric of c_i has reached the bound specified in c_i 's consistency specification; if so, it marks o_i as *dirty* in c_i 's *dirty table*. When processing the next round of the validation algorithm, the server verifies that o_i is dirty and, as a result, refreshes the client's state by propagating the updates needed to ensure the client's consistency specifications are met.

Structure Updates. Modifications to the structure of a document change the distances between its regions. As a result, the placement of the objects within the consistency fields of the clients change and new consistency requirements have to be considered; thus the server is required to update its internal data structures accordingly. Furthermore, because a structure update is also an update to the document, the server also updates and re-evaluates the *sequence* state $\psi_{c_i}[\sigma]$ of each client for every object that moved to a different consistency zone.

Pivot Movement. As with structure updates, when a pivot moves the composition of its consistency zones change, resulting in new consistency requirements. As a result, the server has to update the internal data structures representing that client accordingly, as well as re-evaluating, for every object that moved to a different consistency zone, the *sequence* state of the client. Note that in this case $\psi_{c_i}[\sigma]$ is not updated, since the movement of the pivot does not modify the document space.

4.4 Consistency Enforcement

In each periodically executed round of the consistency validation algorithm, the server checks if any update received since the last round resulted in a violation of a client's consistency specification. If so, the identified updates are propagated to the client.

The validation algorithm verifies, for each client c_i and each object o_i , if the object is within the limits imposed by the consistency zone defined by the client's pivot(s). Because c_i may have multiple views and multiple pivots, the server must first identify which pivot p_i enforces the strongest consistency requirements for o_i . Then, it identifies the consistency zone of p_i where o_i lies, retrieving the corresponding consistency vector κ_i .

Next, each dimension of the identified κ_i is tested. Verifying *time* (θ) and *sequence* (σ) is straightforward: for σ , the server simply checks if the object has

been previously marked as dirty (Section 4.3); for θ , it tests if the time elapsed since the last time c_i was updated with the latest version of o_i exceeds θ_{κ_i} .

To verify ν , on the other hand, the server has to compare the current value of o_i with the client's $\psi_{c_i}[\nu, o_i]$, which would require it to store, for every client, a copy of every object. To avoid the memory overhead of such a solution, the server takes a snapshot of an object whenever updates regarding that object are propagated to a client. Before taking a snapshot, however, it first verifies if a snapshot of the object already exists; if it does, the server uses it, avoiding an unnecessary copy of the object and saving memory.

5 Implementation

We implemented a prototype of the CoopSLA middleware and extended the Linux version of the LaTeX editor *Texmaker*¹ on top of it. In this section we describe the main implementation details of the middleware (Section 5.1) and the extension to Texmaker (Section 5.3) and explain how programmers interact with the middleware and specify the CoopSLA consistency settings (Section 5.2).

5.1 Middleware

Each semantic region of the application graph is represented by an SRegion object that contains a list of children subregions and a TreeDoc with the contents of the region it represents. SRegions are uniquely identified by the server; this identifier is used by clients to access the semantic region represented by the object. The list is ordered by the semantic order of the children subregions in the document space. SRegions also hold a programmer provided DataUnit object containing application-specific information. It may be used, for example, to implement links and references between SRegions.

Implementing the object snapshot approach described in Section 4.4 requires rounds, snapshots and objects to be *versioned*. The round number r_v is an integer number that is incremented in every round. Snapshot and object versions are assigned based on round versions: snapshot versions correspond to the r_v of the round in which the snapshot was taken; object versions correspond to the r_v of the round in which the object was last updated. To dispose of snapshots that are no longer referenced we hold a list of the clients that reference the snapshot and collect the latter when the list becomes empty.

To save memory, the per-client pending updates queues do not store actual updates. Instead, it points to the updates stored in the global queue. Thus, the global queue holds the updates received since the last round, as well as any update that has not yet been sent to a client (i.e., is still referenced by a client's queue). When an update is no longer referenced by any client's queue, it is discarded.

¹ <http://www.xmlmath.net/texmaker/>

5.2 Interfacing with Programmers

In our current implementation, programmers describe the consistency requirements using an XML file. When the client application starts, it invokes an API registration function with the path to the XML file as its argument. The CoopSLA client then parses the file and sends a registration request to the server.

Programmers control the structure of the document by adding or removing semantic regions using API functions (`addSRegion/removeSRegion`). Both functions receive the parent of the new region, the region *type* and, optionally, a set of semantic links to other regions. The region type is an application-dependent string value used to identify the objects consistency zone for a particular pivot (explained later in this section).

Programmers must provide two additional functions to be called by the middleware when checking a client’s consistency: **valueDiff** and **getConsistencyZone**. The **valueDiff** function is used to verify the *value* metric. It returns the (application-specific) percentage difference between two versions of an object. **getConsistencyZone** is a functions that, given a pivot, a graph object and the graph path between the pivot and the object, returns the consistency zone of the object regarding the pivot.

5.3 CoLaTeX

To validate our system, we have extended the Tex editor *Texmaker* with cooperative capabilities using its add-ons feature. Our add-ons consist of simple functions that intercept the user’s modifications to the local replica of the shared document, insert the updates received from the server into the Latex document and track the user’s editing position.

We defined one pivot for each open file, each corresponding to the semantic region being edited by the user within that file. We implemented an add-on that allows the user to manually assign a region of the document to a consistency zone. Our **valueDiff** function returns the percentage difference in number of characters between two versions of an object. Table 2 describes the consistency zones we used defined; the **getConsistencyZone** function returns the consistency zone of an object based on the following considerations:

- *Consistency Zone 0* includes the region in which the pivot is placed and its direct children, i.e., the ones that are one graph-hop below the pivot.

Table 2. Consistency Zones

Zone	<i>Time</i> (θ)	<i>Sequence</i> (σ)	<i>Value</i> (ν)
1	1 sec.	1 update	1%
2	10 sec.	15 updates	5%
3	40 sec.	100 updates	30%
4	2 min.	750 updates	60%
5	5 min.	1000 updates	90%

- *Consistency Zone 1* contains the direct parent – the region one graph-hop above the pivot – and indirect children – identified by traversing the graph downwards from the pivot, excluding the direct children – of the pivot.
- *Consistency Zone 2* comprises the regions that belong to the same `\chapter` as the pivot. If there is no explicit `\chapter` defined, we consider that the document has one implicit chapter of which every section is a part of.
- *Consistency Zone 3* includes the top-level sections of the remaining chapters (`\chapter`) of the document. If the document does not have chapters, zones two and three are merged and zone four is awarded the consistency specifications originally defined for zone three.
- *Consistency Zone 4* contains the regions that do not belong to any of the previous zones.

6 Evaluation

We conducted a series of tests to experimentally validate our claim that CoopSLA makes a more efficient use of network resources by exploiting the *locality of interest* of users. In particular, we intended to quantify the savings that CoopSLA obtains regarding the overall bandwidth required to propagate the updates generated during an editing session and the access frequency to the network resources. In the following sections we detail the experiments conducted. We first describe the configuration of the simulation environment, and then present and analyse the results obtained.

6.1 Simulation Environment

Clients are simulated by running a predetermined number of parametrized *editing bots*. Editing bots perform text insertions (write or paste), deletions (erase or cut) and browse through the document space. Table 3 shows the decision tree that models the behaviour of the bots used in our experiments.

Each simulation consisted in a five minute run during which bots executed according to their decision tree, propagating the corresponding updates to the server. The server monitored inbound and outbound messages, storing per-client and overall values regarding bandwidth and number of exchanged messages. Unless told otherwise, the results presented were obtained using the consistency requirements described in Table 2. For simplicity, we chose to have only on pivot per-client in the experiments.

Table 3. Bot decision tree

	Add		Remove		Move	
Read	Write	Paste	Erase	Cut	To sides	Up/down
60%	15%	3%	8%	3%	8%	3%

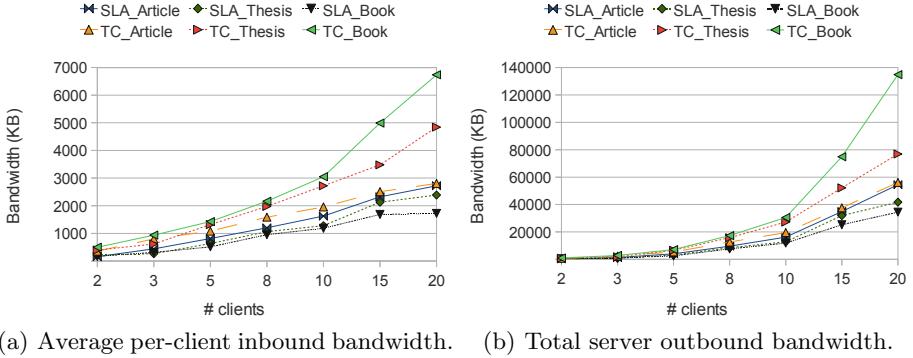


Fig. 2. Used bandwidth

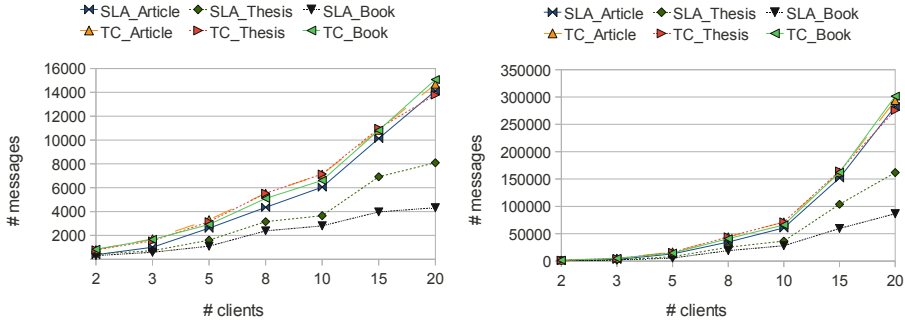
We compared CoopSLA with a baseline TreeDoc implementation that propagates updates to every user as soon as they arrive at the server. Throughout this section we refer to this system as *Total Consistency* (TC), due to its eager propagation approach.

The tests were conducted on Intel Core 2 Quad machines with 8GB Ram running Ubuntu Linux. The server executed on a dedicated machine with no other user-level application running; clients were deployed on up to three machines. The computers were connected through a LAN.

6.2 Evaluation Results

In the first set of experiments, we measured bandwidth and number of accesses to the network at both the clients and the server. In these simulations we varied the number of clients and the type of document space edited. We considered three types of documents that differ mainly in size and structural complexity: an article, a PhD thesis and a book.

Figure 2 presents the results obtained regarding bandwidth. We plotted the results of both CoopSLA and TC for an increasing number of users and the three types of documents we consider (article, thesis and book). The figures show that CoopSLA is able to effectively reduce bandwidth usage both at the client and the server side. Moreover, it shows that as the number of clients and the size of the documents increase, CoopSLA is increasingly more efficient in obtaining bandwidth savings. This behaviour shows the scalability of the system, and is especially relevant considering that as an editing project grows in size, it is more likely that more users will cooperatively access it. The main reason for these results is that in larger documents the average distance between the editing regions of the users is higher; as a result, the probability of postponing and, eventually, merging updates also increases. This fact is particularly evident if we make a pairwise comparison between CoopSLA and TC for each document type. With the article (CoopSLA_Article and TC_Article), the bandwidth savings obtained by CoopSLA are minimal, because the probability that an update



(a) Average number of messages received per-client. (b) Total number of messages sent by the server.

Fig. 3. Messages exchanged

occurs in a client's pivot region (and, consequently, is propagated immediately) is high². If the document grows in size and complexity, the bandwidth savings obtained by CoopSLA increase greatly: approximately 45% less bandwidth with the thesis document (CoopSLA_Thesis and TC_Thesis) and 65% with the book (CoopSLA_Book and TC_Book).

Another important conclusion that can be inferred from Figure 2 is that CoopSLA is able to efficiently minimize one of the main drawbacks of the CRDT approach, the size overhead of path identifiers. When a document is represented as a CRDT, the size of the TreeDoc path identifiers increases as the document grows. Because update messages exchange path identifiers, when a document grows in size, the update messages follow the same pattern. Without CoopSLA, the larger the document is, the larger update messages are; as a result, the bandwidth required to update clients increases. With CoopSLA, on the other hand, we take advantage of the accumulation of postponed messages at the server to merge them before propagating them to the clients.

While the overall traffic generated by the clients is influenced by the specific characteristics of TreeDoc, the number of update messages issued by each client depends only on the editing pattern of the users. By measuring the number of messages exchanged over the network, we are able to clearly isolate and analyse the individual contribution of CoopSLA. Figure 3 shows the results of these measurements for both CoopSLA and TC.

The results further confirm the ones regarding bandwidth. Figure 3 shows that CoopSLA is able to reduce the number of messages received by each client and those savings increase with the size and complexity of the edited document. Again, these results are a direct consequence of CoopSLA's ability to leverage the accumulation of low-priority postponed messages by merging those that cancel each

² Note, however, that we could have obtained better results by configuring the consistency requirements of the pivot region to a less strict setting. We analyse the influence of varying the parameters of CoopSLA later in this section.

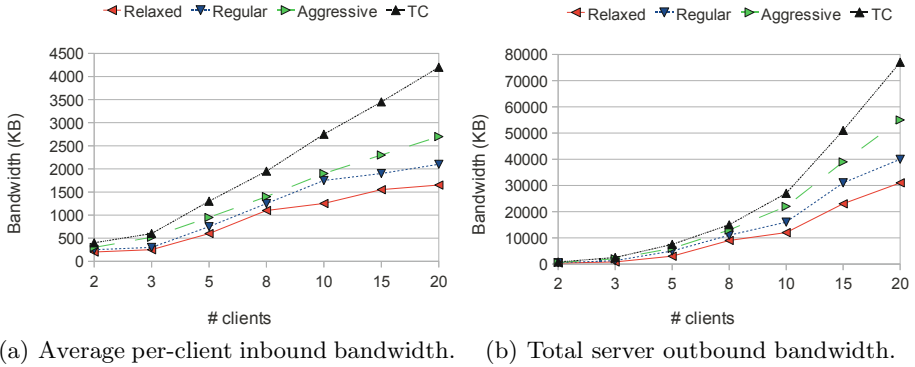


Fig. 4. Bandwidth usage with different consistency requirements

other. As a result, CoopSLA discards unnecessary messages that would have, otherwise, been propagated immediately to each client. This behaviour is desirable not only because it has a direct influence on the reduction of the bandwidth requirements, but also because it means that mobile devices using CoopSLA make a less demanding usage of the network resources, which contributes to reduce battery consumption. Furthermore, these results provide an encouraging indication of how CoopSLA would work with different *operation-transfer* strategies, like OT or other CRDT implementations.

The results presented so far misleadingly indicate that when a document is small, there is no advantage in using CoopSLA. However, CoopSLA allows application programmers to specify consistency requirements arbitrarily, as they see fit for their applications. As long as possible, a programmer should try to relax the consistency requirements, always ensuring the application provides the required levels of interactivity. If necessary, however, the programmer can define more demanding requirements. To show the flexibility of CoopSLA, we measured the bandwidth usage of three CoopSLA consistency specifications that differ in update propagation aggressiveness, as described in Table 4. The *Relaxed* specification provides the weaker guarantees; in particular, it does not require high-priority updates to be propagated immediately. *Aggressive* is the most demanding specification; it requires high-priority updates to be immediately propagated and

Table 4. Consistency Zones

Zone	Relaxed	Regular	Aggressive
1	$\{\theta=5, \sigma=10, \nu=5\}$	$\{\theta=2, \sigma=5, \nu=5\}$	$\{\theta=1, \sigma=1, \nu=1\}$
2	$\{\theta=20, \sigma=30, \nu=10\}$	$\{\theta=10, \sigma=10, \nu=5\}$	$\{\theta=5, \sigma=5, \nu=5\}$
3	$\{\theta=40, \sigma=100, \nu=50\}$	$\{\theta=40, \sigma=100, \nu=30\}$	$\{\theta=15, \sigma=15, \nu=20\}$
4	$\{\theta=120, \sigma=750, \nu=60\}$	$\{\theta=90, \sigma=300, \nu=60\}$	$\{\theta=30, \sigma=50, \nu=50\}$
5	$\{\theta=300, \sigma=1500, \nu=90\}$	$\{\theta=180, \sigma=750, \nu=80\}$	$\{\theta=60, \sigma=150, \nu=50\}$

the remaining zones to be updated frequently. *Regular* is an intermediate specification that provides more relaxed consistency than *Aggressive*, but stricter than *Relaxed*.

Figure 4 shows the results obtained; the bar labelled TC corresponds to the version of the baseline TreeDoc implementation that does not use CoopSLA, while the remaining bars correspond to the three specifications of Table 4. The measurements were made using the Thesis document. As expected, the figure shows that CoopSLA is more efficient when the consistency requirements are more relaxed. This happens because relaxed consistency requirements allow for a larger volume of updates to be retained at the server for longer periods; as a result, the probability that two updates can be merged increases. Conversely, when we increase the aggressiveness of the requirements, we reduce the volume of updates that are retained at the server, thus reducing merge efficiency. However, even considering that CoopSLA is less effective with stronger consistency requirements, the results show that even a fairly strict set of requirements is able to obtain more than 20% bandwidth savings over the version that does not use CoopSLA.

7 Related Work

In this section we discuss prior work on the two topics that are more closely related with our work: divergence bounding and consistency in cooperative editing.

7.1 Divergence Bounding in Optimistic Replication

Designers of replicated systems typically choose between pessimistic and optimistic consistency models[12]. In many cases, however, neither the performance overheads imposed by strong consistency neither the lack of limits for inconsistency are acceptable to applications. An interesting alternative called *divergence bounding* consists in allowing updates to be managed optimistically, but define under which conditions replicas are required to converge and how to enforce that convergence. *Real time guarantees*[1], for example, allows replicas to remain stale for a specified maximum time, before they are required to synchronize. *Order bounding*, another simple solution, limits the number of updates that can be applied to a local replica without synchronization[5].

The TACT[21] framework proposes a multi-dimensional approach to divergence bounding that unifies in a single model three metrics: real-time guarantees, order bounding and a novel metric called *Numerical Error* that bounds the total number of updates, across all replicas, that can proceed before replicas are forced to synchronize. Our work distinguishes from TACT by embodying the notion of locality-awareness into the consistency model. This allows our system to implicitly assign different priorities to different updates that may vary throughout execution.

Vector Field Consistency (VFC) [13,17] is a consistency model for mobile multiplayer games that enables replicas to define their consistency requirements

in a continuous consistency spectrum. The novelty of the VFC model is that it combines multi-dimensional divergence bounding with *locality-awareness* to improve the availability and user experience while effectively reducing bandwidth usage. Consistency between replicas strengthens as the distance between objects decreases. To define these mutable divergence bounds, around pivots there are several concentric ring-shaped *consistency zones* with increasing distance (radius) and decreasing consistency requirements (increasing divergence bounds). Then, in each zone, like in TACT, programmers define a 3-dimensional vector: *time, sequence, value*.

7.2 Consistency in Cooperative Editing

The issue of maintaining replica consistency in cooperative applications has been extensively studied in the last two decades. The most representative solutions fall into the *Operational Transformation* (OT)[2,15,14,7,16,20] category. In OT, each locally generated operation is associated with a timestamp and broadcast to the remaining sites. Then, each remote update received is transformed (e.g., by adjusting its insert/delete index) in order to commute with concurrent operations already applied to the shared document. As a result, transformed operations can be executed without re-ordering previous applied operations.

OT transforms updates in order to make them commute. A recently introduced alternative is to make every operation automatically commutative by representing the document as a *Commutative Replicated Data Type* (CRDT)[9,8,18,11,19]. The CRDT approach considers that a document is composed of a sequence of immutable and uniquely identified elements that can be any non-editable component of a document, like a character or a graphics file. Commutativity is achieved by designing an identifier space that ensures that it is always possible to create a new identifier between two existing ones[9].

To the best of our knowledge, neither approach (OT or CRDTs) considers the dynamically changing interest of the users in the different semantic regions of a document; instead, they propagate every update with the same static priority. Moreover, CoopSLA can use either OT or CRDT as building blocks for update propagation. As described in Section 4.2, in our current implementation we used CRDTs.

8 Conclusion

In this paper we presented a semantic and locality aware consistency model for cooperative editing applications. Our model, named CoopSLA, explores the heterogeneous and dynamic interest of users in different regions of a document space in order to reduce communications between the participants of an editing session. CoopSLA assigns, on a per-user basis, different priorities to different updates, based on the semantic distance between the place in the document where the update is performed and the places in which the user is more interested. Updates with high priority are sent frequently to the client, while low priority

updates are postponed and, when possible, merged. Each priority level is characterized by a multidimensional consistency degree that defines how many and how long updates to a particular object are allowed to be postponed.

We implemented a middleware layer enforcing CoopSLA and extended the popular Tex editor *TexMaker* with cooperative features using it. We conducted a series of tests to experimentally evaluate the performance of CoopSLA. The results presented in this paper support our claim that CoopSLA is very effective in reducing the overhead of replica synchronization without constraining application models and respecting their consistency need.

References

1. Alonso, R., Barbara, D., Garcia-Molina, H.: Data caching issues in an information retrieval system. *ACM Trans. Database Syst.* 15(3), 359–384 (1990), <http://doi.acm.org/10.1145/88636.87848>
2. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: *SIGMOD 1989: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 399–407. ACM, New York (1989)
3. Ellis, C.A., Gibbs, S.J., Rein, G.: Groupware: some issues and experiences. *Commun. ACM* 34, 39–58 (1991), <http://doi.acm.org/10.1145/99977.99987>
4. Imielinski, T., Badrinath, B.R.: Mobile wireless computing: challenges in data management. *Commun. ACM* 37(10), 18–28 (1994)
5. Krishnakumar, N., Bernstein, A.J.: Bounded ignorance: a technique for increasing concurrency in a replicated system. *ACM Trans. Database Syst.* 19(4), 586–625 (1994)
6. Li, D., Anand, M.: Majab: improving resource management for web-based applications on mobile devices. In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, MobiSys 2009*, pp. 95–108. ACM, New York (2009), <http://doi.acm.org/10.1145/1555816.1555827>
7. Li, R., Li, D.: A new operational transformation framework for real-time group editors. *IEEE Transactions on Parallel and Distributed Systems* 18(3), 307–319 (2007)
8. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for p2p collaborative editing. In: *CSCW 2006: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, pp. 259–268. ACM, New York (2006)
9. Preguiça, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: *ICDCS 2009: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pp. 395–403. IEEE Computer Society, Washington, DC (2009)
10. Preguiça, N., Martins, J.L., Domingos, H., Duarte, S.: Data management support for asynchronous groupware. In: *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, CSCW 2000*, pp. 69–78. ACM, New York (2000), <http://doi.acm.org/10.1145/358916.358972>
11. Roh, H.G., Jeon, M., Kim, J.S., Lee, J.: Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel Distrib. Comput.* 71(3), 354–368 (2011), <http://dx.doi.org/10.1016/j.jpdc.2010.12.006>
12. Saito, Y., Shapiro, M.: Optimistic replication. *ACM Comput. Surv.* 37(1), 42–81 (2005)

13. Santos, N., Veiga, L., Brandt, F.: Vector-Field Consistency for Ad-Hoc Gaming. In: Cerqueira, R., Campbell, R.H. (eds.) *Middleware 2007*. LNCS, vol. 4834, pp. 80–100. Springer, Heidelberg (2007)
14. Shao, B., Li, D., Gu, N.: A fast operational transformation algorithm for mobile and asynchronous collaboration. *IEEE Transactions on Parallel and Distributed Systems* 21(12), 1707–1720 (2010)
15. Sun, C., Ellis, C.: Operational transformation in real-time group editors: issues, algorithms, and achievements. In: *CSCW 1998: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pp. 59–68. ACM, New York (1998)
16. Sun, D., Sun, C.: Context-based operational transformation in distributed collaborative editing systems. *IEEE Trans. Parallel Distrib. Syst.* 20(10), 1454–1470 (2009), <http://dx.doi.org/10.1109/TPDS.2008.240>
17. Veiga, L., Negrão, A., Santos, N., Ferreira, P.: Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. *J. Internet Services and Applications* 1(2), 95–115 (2010)
18. Weiss, S., Urso, P., Molli, P.: Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In: *ICDCS 2009: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pp. 404–412. IEEE Computer Society, Washington, DC (2009)
19. Wu, Q., Pu, C., Ferreira, J.: A partial persistent data structure to support consistency in real-time collaborative editing. In: *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pp. 1707–1720 (March 2010)
20. Xia, S., Sun, D., Sun, C., Chen, D., Shen, H.: Leveraging single-user applications for multi-user collaboration: the cword approach. In: *CSCW 2004: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pp. 162–171. ACM, New York (2004)
21. Yu, H., Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.* 20(3), 239–282 (2002), <http://doi.acm.org/10.1145/566340.566342>

Foster an Implicit Community Based on a Newsletter Tracking System

Tiago Lopes Ferreira and Alberto Rodrigues da Silva

Instituto Superior Técnico de Lisboa
Lisbon, Portugal

tiagohenrique@ist.utl.pt, alberto.silva@acm.org

Abstract. Communities have explored the virtual world as a tool to improve their communication. However, when the number of interactions was manageable in its face-to-face manner, the same was not true when the Internet became the main communicator. The number of interactions grows at a pace that is very hard for communities to control. As a consequence connections get lost or forgotten and communities lose the chance to perceive individuals' relations. It is in this gap that the "Newsletter Tracking System" (NTS) comes as an automatic tool that allows communities to capture connections between individuals through their interactions with newsletters. By storing the data on individuals' interactions, NTS discovers implicit connections between individuals and fosters an implicit community. In addition, it uses clustering algorithms to allow communities to better understand how individuals relate to each other and it proposes a "Connection Degree" model (CD) to measure the connections' strength among individuals. NTS and CD were developed and evaluated within Nano-Tera.ch scientific community. At the end, the results showed that implicit communities can be an advantage for real communities to better organize individuals, share knowledge, and promote teamwork.

Keywords: Community, Newsletter Tracking System, Connection Degree, Implicit Connection.

1 Introduction

New technologies have changed the way people interact by providing new approaches to communicate, share, and stay connected to each other. The Internet has revolutionized the computer and communication worlds like nothing before [13], and today it reaches any field and affects the way society builds connections. People can create their own network of contacts and share information with anyone around the world. The more people interact, the more their network of interactions grows [15]. Likewise, communities started to change from groups to networks and to take advantage of the Internet as communication tool [23]. In 1993, the concept of "Virtual Communities" [25] came to live and also the research on the relation between communities and the Internet. Virtually, communities have no restriction on the number of connections and the way they can explore them. When tracking all individuals' connections was

extremely difficult in the physical world it becomes an easier target in the virtual sphere. Technology brought people together as well as their own interests, curiosities, hobbies, professions, and so on. Each time people go online they become exposed and their interactions can be stored as connections with something or someone.

The connections can be defined as explicit connections, if they are clearly expressed by individuals (e.g. individuals' friends or followers), or as implicit connections, every time they are implied (e.g. individuals' interests). Facebook is a good example on explicit and implicit connections. Although friend requests results from the explicit activity of sending a request, the action of clicking a friends' post can result into an explicit relation between the user and the post subject and into an implicit connection with another user who have clicked the same post.

Both explicit and implicit connections are important to understand individuals and communities' network. Explicit connections ensure the knowledge-base on individuals' relations and implicit relations improve that knowledge. An explicit behavior "is controllable, intended, made with awareness, and requires cognitive resources" [6]. Individuals have clear sense of explicit activities and this makes explicit relations extremely important when defining individuals and communities' networks. However, the study on explicit connections can be very limited if individuals do not share behavior and interests. To fill the gap, implicit connections can be used and exploded as long as individuals continue to interact with content and people. Implicit connections are based on individuals' unconscious interactions and can be tracked if they occur virtually. In the presented study an interaction between an individual and the newsletter is stored as an explicit connection between the individual and the subject. Then the explicit connection "individual-subject" is translated into an implicit connection "individual-individual". Implicit connections are important to help a network of connections to be evolved beyond individuals' explicit behavior. The more interactions an individual does, the stronger is the implicit knowledge in the network [21].

The problem arises when capturing and analyzing individuals' implicit connections at the rate they grow. Data management becomes difficult to handle and implicit patterns harder to find. As consequence, "the connections between individuals, groups, and information becomes lost, or forgotten, and individuals and groups become more isolated" [11]. In the presented case-study of Nano-Tera.ch [16], a scientific community at Switzerland, the governing bodies were aware of the complexity on capturing researchers' connections as the community grew. The goal of exploring implicit connections was difficult to achieve due to the increase number of interactions and the high complexity on capturing researchers' interactions. Nano-Tera.ch was interested in understanding how its community was implicit organized but had no way to capture and promote interactions between researches. In fact, Nano-Tera.ch needed an automatic tool able to track the connections among researchers and organize them into results. In addition, there was the goal of finding a way to classify implicit connections in order to measure the connection's strength and thus understand the connections' influence in the discovered universe.

The presented research developed a tool with Nano-Tera.ch to enable communities to capture implicit connections between individuals and also to design a way to measure the connection degree among individuals. The "Newsletter Tracking System" (NTS) is

that developed web tool to discover implicit connections between individuals based on their interactions with newsletters. Each interaction is stored and translated into a relation between the individual and the content. At the end, individuals that have interacted will be related through newsletters' content and therefore implicit related to each other. The "Connection Degree" (CD) model is based on explicit and implicit behavior and it proposes a way to measure the connections' strength among individuals. Individuals are organized into a network of connections and each connection is related to a connection degree value expressing its strength in the network. The higher the connection degree is, the higher the importance of the relationship is for the community.

This paper is structured in 6 sections. The presented introduction as the section 1 and the section 2 as the description of NTS as a tool for communities to explore implicit connections. Section 3 explains the CD model and ends with section 4 where it is described the results according to the practical case study of Nano-Tera.ch community. By the end, section 5 explores some of the related work and section 6 presents several research conclusions.

2 The Newsletter Tracking System

Communities have taken their step into the virtual world and have included the web tools in their habits. Emails, forums and blogs became part of communities' ways to interact. However, when the number of connections between individuals increases, the task of controlling the implicit growing is difficult to monitor.

Through interactions with newsletter the "Newsletter Tracking System" (NTS) proposes a way to capture individuals' interactions, discover implicit connections, and so foster an implicit community. NTS uses electronic mail technology to reach individuals, and newsletters to promote interactions and discover implicit connections among individuals. According to Bellotti Ducheneaut "even colleagues having offices next to each other, or sitting in plain sight of each other, still use e-mail as a principal communication medium" [5]. Also the fact that email is used worldwide and one of the most known tools makes it one of the tools to better reach individuals and capture interactions. The use of newsletters comes as the way to capture individuals' interactions. It gives communities the freedom to design their content and define the newsletter according to individuals' interests. In addition, communities can use their newsletters to extract individuals' interactions while keeping them updated.

In the context of the NTS, a newsletter is defined as a set of news collected into an HTML file. Its content can be defined by several authors and thus result on a collaborative common information space. The periodicity (e.g. monthly) defines the pace at which information reaches individuals and it is defined by the community itself.

2.1 System Overview

The NTS is an online tool that allows communities to foster an implicit community based on individuals' interactions with newsletters. In fact, the NTS supports the relationship between the community and the individuals by providing the tracking tool for the community (Fig. 1). The community itself is managed by a "Community

Manager” that is responsible for triggering the tasks at the NTS. Based on the goal of “Fostering an Implicit Community”, the community manager depends on the NTS to achieve it. This relationship is based on a goal dependency and expressed as a relation “depender-dependee”¹, where the “Community Manager” (the “depender”) depends on the “Newsletter Tracking System” (the “dependee”) to achieve the goal.

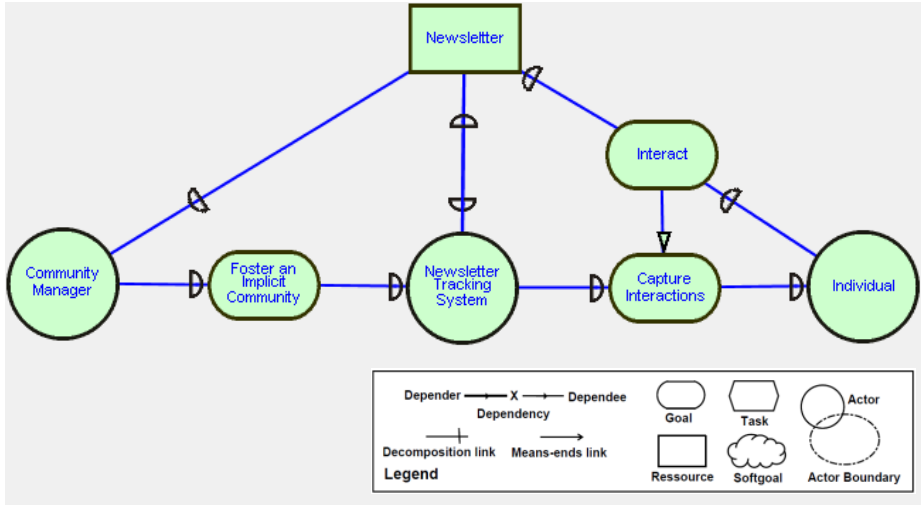


Fig. 1. System Overview

The NTS is responsible for making the decisions that are necessary to achieve the goal and the community manager does not care how the NTS goes about achieving it. On the other hand the NTS has a resource dependency with the community manager. The NTS depends on the community manager to provide the newsletter so it can perform the tasks, satisfy the goals, and also provide the resources. Without all the input elements no further links can be followed and the model stops.

The relationship between the NTS and the individual is also based on goal and resource dependencies. In order to satisfy the goal of “Capturing Interactions”, the NTS depends on the individuals to “Interact” with the “Newsletter” that is a resource dependent on the NTS. The dependency happens in both directions. The NTS needs the individuals to interact with the newsletter, and the individuals need the NTS to provide the newsletter in order to achieve the goal of interact. If some of the elements in the relationships does not do its role as a “dependee”, both parts end up not achieving their goals. The link between the goals “Interact” and “Capture Interactions” represents a “means-ends” link. The mean of interact has an end of capture interactions, which are then used for the NTS.

¹ The used terminology as well as the presented schemas are based on i* framework defined by Yu Eric [24]. It “conceives of software-based information systems as being situated in environments in which social actors relate to each other in terms of goals to be achieved, tasks to be performed, and resources to be furnished” [9].

The presented model is based on a dependency model of goals, where the actors “Community Manager”, “Newsletter Tracking System”, and “Individual” are depended on each other based on goals. The direction of the dependency link defines the way the goal is achieved, i.e. which actor, task, or resource the goal depends on. Also, on the relationships where the “Newsletter” is a resource, the “Community Manager” represents the “dependor” regarding the link with the NTS (the “dependee”), and the “Individual” the “dependee” in the relation with the NTS.

In a deep exploitation of the NTS, this is based on the three main tasks of “Upload Newsletter”, “Send Newsletter”, and “Analyze Data” (Fig. 2). The community manager’s goal of foster an implicit community is decomposed into three different tasks that are trigger by him and performed by the NTS. Although the relationship between the community manager and the NTS is a goal dependency, it can be described as a decomposition of three tasks that need to be performed in order to achieve the goal.

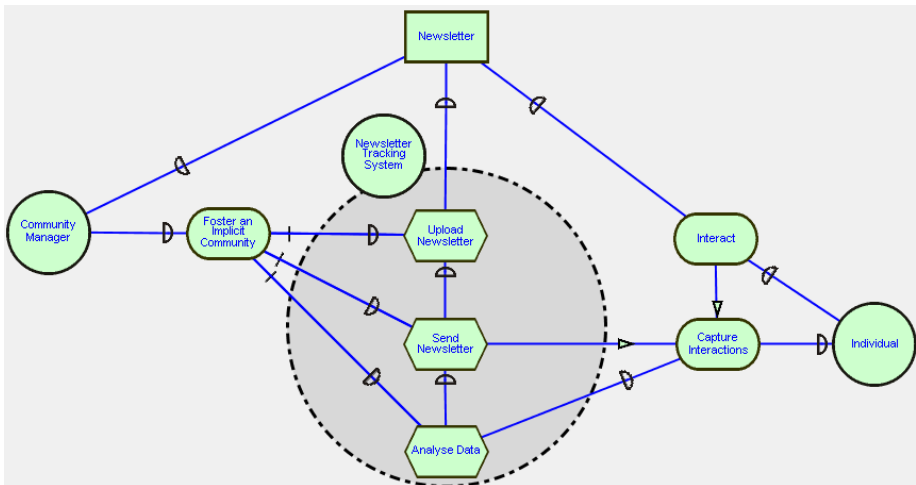


Fig. 2. Newsletter Tacking System Overview

The dependency links show the order in which tasks must be performed. Only the tasks that do not have any dependency links going out can be performed right away the community manager wants to meet his goal. In this case all the tasks are “dependers” and need their “dependees” to run. By following the dependency links, the interpretation is that the “Upload Newsletter” task can be performed as soon as the “Community Manager” provides the “Newsletter”, then the “Send Newsletter” task, and by the end the “Analyse Data” task. Thus, in order to the “Community Manager” achieve the goal of “Foster an Implicit Community” he needs to trigger on the “Newsletter Tracking System” the task “Upload Newsletter” by providing the “Newsletter” as the input, then the “Send Newsletter” task, and finally ask for the system to “Analyse Data”. The task “Send Newsletter” has the end goal of “Capture Interactions” that is needed to perform the last task of “Analyse Data”. Again the “Individual” is responsible for meeting the goal of “Interact” and close the cycle of the dependency links. Once all the dependency links are respected the goals can be reached and the

NTS is able to help communities to fostering an implicit community based on individuals’ interactions with newsletters.

2.2 Upload Process

The process of capture and detect implicit connections among individuals starts with a community uploading a newsletter into the NTS. The upload process is described as the first interaction between a community and the NTS. At this stage, a community reveals its interest on capturing individuals’ interaction and on fosters an implicit community. The process can be represented by Fig. 3 where the “Community Manager” and the “Newsletter Tracking System” are the only actors. The model starts with the dependency goal of “Upload Newsletter” between the community manager and the NTS. In order to the community manager satisfy his goal of uploading the newsletter he needs the NTS to perform the task “Newsletter Uploading”. On the other hands, the NTS needs the “Newsletter” resource given by the community manager. Thus, he should first design the newsletter and then meet the goal of uploading it.

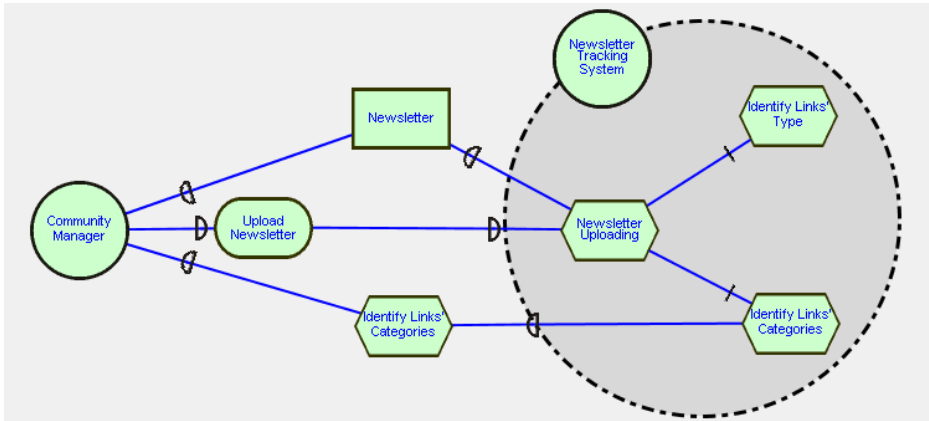


Fig. 3. Upload Process

The newsletter is described as the main resource for the system since it is the element that is shared with individuals. The community manager is responsible for choosing the content, designing the newsletter, and be aware of the newsletter’s importance as a promoter of interactions. The better a newsletter meets the individuals’ needs, the higher the number of interactions. This responsibility is given to communities once they know better what individuals’ desire and expect.

As soon as the newsletter is ready to send, the community manager is able to initiate the upload process by triggering the task “Newsletter Uploading” on the NTS and providing the newsletter as a resource. At this stage the NTS is able to decompose the task into two different tasks - “Identify Links’ Type” and “Identify Links’ Categories”. On the first task the NTS will process the newsletter and identify the type of all the links. The type is defined as the way links can be illustrated and can be identified

The “base-url” represents a common prefix to all the links, namely the path to the server where the NTS is working. The “code” defines one of the possible actions: opening the email, clicking a link, sharing the newsletter, or seeing it online. The “user-id” identifies the individual who triggered the action, the “link-id” which link was clicked, and the “newsletter-id” the newsletter in what the action was performed. The elements are all automatically generated by the NTS and put together in order to build a tracked newsletter for each individual. The newsletters are then the key elements to perform the task of “Send Emails”, where the community manager performs is last interaction with the NTS by proving the “Mailing List” resource, which contains the list of emails to which the newsletter is going to be sent.

The task of sending the emails with a tracked newsletter has the end of “Capture Individuals’ Interaction”. Once the newsletter reaches the “Individual” all the remaining process depends on him, namely on his interactions with the newsletter. The dependency process starts with individuals opening their emails with the tracked newsletter and ends with individuals’ clicking the links.

2.4 Tracking Process

The core component in the NTS boils down to the tracking process, where all individuals’ interactions are capture and stored into the database. Each time an individual clicks a link in the newsletter, the action is stored into the NTS as an interaction between the individual and the newsletter. The follow schema Fig. 5 explains how the tracking process is managed in the NTS and how actors play their roles. “Individual” plays the main role as the tracking process booster by performing the task of “Click a Link” on the “Tracked Newsletter”. The task is then decomposed into the tasks “Track Interaction” and “Get Real Link”.

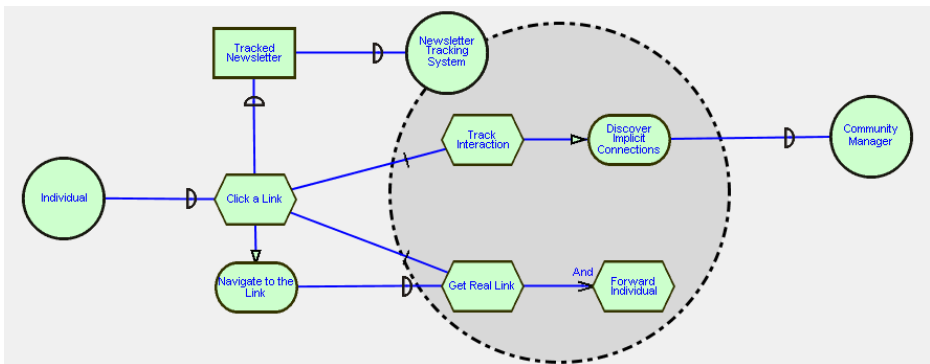


Fig. 5. Tracking Process

Before storing the interaction the NTS performs the task of getting the real link. It converts the tracked link into the original link to which the individual wants to navigate. The task is completed when all the tasks under it are also performed. In this case the contribution link “And” symbolizes that the parent “Get Real Link” is satisfied if

the offspring “Forward Individual” is also satisfied. The NTS should satisfy the goal of “Navigate to the Link” by translating the tracked link into the real link and forward the individual. Once the individual navigates to the link, the NTS loses the individual’s track and no more interactions are stored.

The task “Track Interaction” is based on the data storage of the interaction. At this stage, the NTS stores all the information compiled on the tracked link. The system stores the information about the clicked newsletter, link, individual, and time. The information is stored into the database and used for data analysis. At the end, the task of tracking the interaction has the end of “Discover Implicit Connections” among individuals, where the “Community Manager” appears as the actor responsible for triggering the processes in order to meet the goal.

From the newsletter design to the foster of an implicit community, the NTS is a solution based on the actors “Individual” and “Community Manager” to get the resources, perform the tasks, and achieve the goals. Through dependency links the schemas show how the solution works and how all the pieces fit together.

2.5 Data Analysis

The captured data on individuals’ interactions is the most important asset and the one that allows communities to foster implicit communities. The more data the NTS is able to capture, the stronger will be the results on individuals’ implicit connections and the greater the value of the discovered networks. The NTS is able to expose the captured data by organizing it into visuals and allowing communities to export it.

Although the analysis on a particular newsletter is available right after the newsletter is sent, it is up to the community manager decide when the analysis should be carried out. To help on this decision the NTS provides overall information on the newsletter, such as time passed – total number of days that have passed since the newsletter was sent –, the total number of individuals who interacted with the newsletter, and the total number of clicks. This information is useful to have an overview of the newsletter’s impact and to monitor the results. On a further analysis the NTS divides the presentation of the data into a set of sections:

1. **Links’ Type.** Organizes the links by their type – “Text” or “Image” – and presents the percentage of clicks on both types. The analysis allows communities to understand the best approach to design newsletters. If individuals interact more with image-based or text-based links.
2. **Links’ Categories.** Presents the categories of the newsletter followed by percentage of clicks gather on each. At the end, communities will be able to understand to which categories individuals showed more interest, and who was the category with higher impact on individuals. This categorization is also used to relate individuals with categories and thereby discover implicit connections among them.
3. **Individuals’ Category Clustering.** On clustering, the NTS uses individuals’ interactions to cluster them by categories. A relation “individual-link” is translated into a relation “individual-category” and the individual is added to the category cluster followed by his total number of clicks on the category. The clustering allows communities to discover all the “category-individual” relations and thus followers.

4. **Implicit Connections.** The NTS translates the relations “individual-category” into implicit connections “individual-individual” and assigns them a “Connection Degree” in order to communities have a way of measure the connections. The value is based on individuals’ interactions with categories and on their explicit preferences on the categories. However, in order to visualize the implicit community-based the NTS uses external tools such as Vizster [12] and NodeXL [17].
5. **Data Export.** The process of exportation is what allows communities to export the data in order to use it on external tools. The NTS enables the data to be exported in the file formats of XLS and XML. The goal is to allow communities to exploit the data the way they want and do not limit its exploitation to what NTS offers.

At this stage communities are able to have an overview of individuals’ implicit connections. Once the newsletters reach individuals it all comes to individuals’ interactions. The NTS will automate the process of sending the newsletters to individuals and track each newsletter in order to capture interactions. The system is also responsible for capture every click and translate it from a relation “individual-link” to a relation “individual-category”. At the end, the relations will be used to foster an implicit community based on individuals’ implicit connections.

3 Connection Degree

With communities having the ability to discover implicit connections among individuals, it becomes important to understand the value of each connection in the discovered universe. In a scenario where all individuals click in all the categories, they will be all implicit connected and the community will find it harder to take conclusions. On the other hand, with a way to measure the value of each connection in the network, the community will have the chance to create their own thresholds and filter the implicit connections.

The connection degree model proposes a way to measure the connection strength between every two individuals by calling it “Connection Degree” (CD). The higher the CD, the stronger the relationship between two individuals. The CD model uses both explicit and implicit connections to calculate the explicit and implicit degrees in the CD. While the implicit degree is based individuals’ implicit connections, the explicit degree comes from individuals’ preferences on the newsletters’ categories.

An individual can express his categories interest by explicitly checking a category as preferred. This process is done through the NTS, where individuals can navigate to a “Preferences Page” through their newsletters and check or uncheck their preferences on the categories. Thus, a checked category is understood as a positive preference, an unchecked category as a negative preference, and an unknown preference when no explicit action is performed. The explicit degree will affect the final CD in a “Category Importance” (ci) value, defined by the community and that represents the importance of the individuals’ preferences in the equation.

$$\text{explicit degree (Individual): } ed(I) = \begin{cases} ci, & \text{positive preference} \\ 0, & \text{no preference shown} \\ -ci, & \text{negative preference} \end{cases} \quad 0 \leq ci \leq 1 \quad (1)$$

On the other hand, implicit degree is calculated based on individuals' clicks on newsletters. The action of clicking a link is stored as an implicit connection between the individual and the link, and between the individual and the link's category. The connections are then used to calculate the implicit degree equation on the CD. On the first relations "individual-link", the connections are organized into a matrix $n \times m$ where both rows (n) and columns (m) represent individuals and the values (v_{nm}) the total number of links that both individuals have clicked in common.

$$n \times m = \begin{bmatrix} v_{11} & \cdots & v_{1m} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nm} \end{bmatrix} \tag{2}$$

Regarding the relations "individual-category", individuals are also organized into a matrix where rows represent all pairs of every two individuals (I_i, I_j) and columns the categories (c_l). The implicit value is then calculated based on the following equation.

$$(I_i, I_j) \times c_l = \min(a_{il}, a_{jl}) \times (ed_i + ed_j) \tag{3}$$

Where a_{il} represents the total number of clicks that individual I_i gave in the category c_l and the ed_i the explicit degree (1) for the individual I_i . The implicit value for the two individuals (I_i, I_j) in the category (c_l) is then calculated based on the minimum value of both individuals' total number of clicks in the category times the sum of both explicit degrees. The first part of the equation represents the minimum value on the individuals' categories-based relation and the second part expresses the individuals' explicit interests on the categories. The explicit degrees are added to this equation once it contains the calculation on individuals' implicit connections per category.

The next step translates the relations "individual-category" to the implicit relations "individual-individual". Individuals are organized into a matrix $n \times m$ where rows (n) and columns (m) represent individuals (I) and the values the sum of all (3) equations for all l categories ($\sum_{i=0}^l$ (3)).

$$n \times m = \begin{bmatrix} \sum_{i=0}^l (I_1, I_1) \times c_i & \cdots & \sum_{i=0}^l (I_1, I_m) \times c_i \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^l (I_n, I_1) \times c_i & \cdots & \sum_{i=0}^l (I_n, I_m) \times c_i \end{bmatrix} \tag{4}$$

The final value for the CD between every two individuals is calculated by performing a syntax sum of both resulted matrix from (2) and (4) but with the multiplication operator. At the end only one of the sides of the matrix is taken into account, i.e. a lower or upper triangular matrix, and excluded the diagonal. This will ignore duplicated pair and take only one CD into account.

$$connection\ degree : n \times m = \begin{bmatrix} (2)_{11} \times (4)_{11} & \cdots & (2)_{1m} \times (4)_{1m} \\ \vdots & \ddots & \vdots \\ (2)_{n1} \times (4)_{n1} & \cdots & (2)_{nm} \times (4)_{nm} \end{bmatrix} \tag{5}$$

The multiplication of both values brings together the implicit connection on links and categories, and reveals the final connection degree for every two individuals. The multiplication as the final operator helps to highlight connections where a click can make a difference. Once the results are based on newsletter's interaction, each click should have a significant value so it can positively influence the CD. The CD will allow communities to compare and highlight the most important implicit connections.

The CD model brings to the NTS value on measuring implicit connections and allows communities to have a better overview of the implicit community. The NTS is responsible for capturing and discovering implicit connection and the CD model for assigning every connection a CD value.

4 Evaluation: The Nano-Tera.ch Case Study

The evaluation of this research was done inside the Nano-Tera.ch community. A scientific Swiss federal program with more than 40 projects on the subjects of "Security", "Health", and "Environment" [16]. Nano-Tera.ch diversity goes from different projects to the hundreds of researchers around the world. Thus, in order to capture knowledge on the Nano-Tera.ch community, the management structure at Nano-Tera.ch decided to run the research project "Community Knowledge Development" (CKD) [6]. Within other goals, the CKD was trying to understand how researchers were related and how connections could be explored in order to improve researchers' work and promote collaboration.

The research project CKD supported the development and the evaluation of both NTS and CD model at Nano-Tera.ch community. This project took place during 6 months and the community management designed 6 newsletters to send to the community and to evaluate the system itself. However, the presented results are based on the top 3 of the newsletters regarding interest and reliability. The overall information on the newsletters was more than 3000 of emails sent (1000 per newsletter) and about 1200 of captured clicks. In addition, Nano-Tera.ch tried to maintain the same thousand of individuals (i.e. emails sent) for every newsletter in order to collect data from different sources but for the same individuals.

4.1 Links' Type

The results focus on trying to understand to what kind of information-exposure individuals interact more - "Text" or "Image". Each newsletter's topic was introduced with an "image" followed by the "text" to which individuals had the chance to interact and reach the same information. This analysis is important for communities to better design newsletters. Individuals at Nano-Tera.ch community have showed their strong interest on information as "Text" with more than 96% of interactions on links linked to text and 4% on links assigned to an image.

4.2 Categories

Nano-Tera.ch decided to define a category for each of the research topics – Health, Environment, and Security – plus a topic related with the community itself – Nano-Tera.ch. The categories were all organized so that all were present in the newsletters.

At the end, the results showed individuals preferences on Health (41%), followed by Environment (27%) and Security (23%), and finally Nano-Tera.ch (9%). In fact, the results on Health can be possibly explained due to the higher number of projects that were born on the Health category and to the Nano-Tera.ch strong research on Health.

4.3 Connection Degree

To illustrate the results on the CD model it was used the external tool NodeXL [17]. Based on the Microsoft Excel, the NodeXL enables the exploration of a community based on every input value. In this case, individuals (nodes) and implicit connections (edges) were organized into a graph and the calculated CDs were used as input values to calculate edges' width and define edges' labels. The presented results are based on the implicit relations with a D higher than 7, in order to highlight the most important connections and have a clearer view of the relevant nodes on the network (Fig. 6).

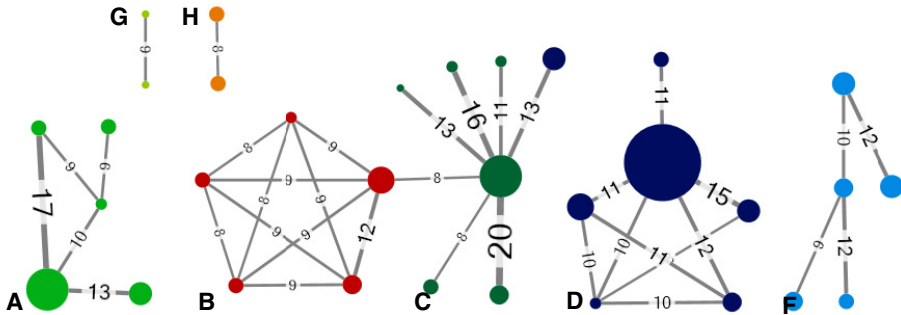


Fig. 6. Connection Degrees

The size of the nodes represents the total number of clicks that the individual have performed and the label on the edges the CD degree calculated to the implicit connection. The colors represent the Girvan-Newman algorithm only applied to the presented connections. By looking at the results the higher CD has a value of 20, with both individuals having more than 6 clicks on the exactly same links on the newsletters. Adding to this, their strong relation is based on the “Security” category, represented by cluster C. The biggest node represents the individual with more than 20 clicks in the newsletters and with a strong interaction on the “Health” category (cluster D). On the other side, individuals at the cluster A (“Nano-Tera”) have also a high CD between them, followed by the cluster B (“Health” and “Security”), the cluster F (“Environment”), and the two isolated clusters G and H with almost all categories. The CDs at the cluster C shows that the “Security” category is the strongest between individuals with a high CD.

4.4 Clusters Detection

The results on clustering detection were performed using the external tool Vizster [12]. Vizster is a social network visualization system that uses Girvan-Newman algorithm to discover clusters in a network of connections.

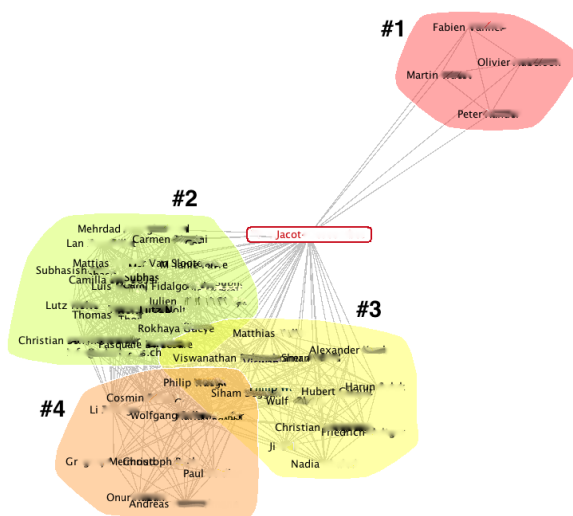


Fig. 7. Clusters Detection

The NTS organizes implicit connections into a network by defining individuals as nodes and connections as edges. Through data exportation, communities are able to use Vizster to visualize NTS discovered network and organize individuals into clusters. The results on the Nano-Tera.ch community showed that individuals are organized into 4 different clusters (Fig. 6). The details show that the Cluster #1 is mainly based on individuals with strong interest on “Nano-Tera” category; the Cluster #2 on individuals with high interest on “Health”; and the Cluster #3 and #4 on individuals highly interested on “Security” and “Environment” respectively.

4.5 Discussion

Once the data is gathered and the results achieved, communities move efforts to better understand the results and extract their value. The presented discussion is based on the Nano-Tera.ch case study and describes some of the conclusions reached at Nano-Tera.ch offices. On links’ type, individuals revealed their text-oriented focus by interacting with a higher number of text-based links than image-based. The results showed that individuals have a strong scientific focus and care about the newsletters’ content. A brief description of a topic will better promote individuals to read it and click it. On the other hand, images had a strong power on balancing newsletter design but a low level of interaction. In fact, “Images” help creating newsletter’s context and “Text” allows individuals to go deeper in the subjects.

On categories, “Health” has proved its high number of interactions and individuals’ preferences on the subject. In fact, the results reflect the higher number of projects on Health and the Nano-Tera.ch main focus on Health issues. The balance between both categories of “Environment” and “Security” can be explained with the approximate number of projects and with the proper division of the topics per newsletters. The fact that Nano-Tera.ch designed all newsletters with fresh news leads individuals to

interact with almost all categories in order to stay updated. On the other hand, “Nano-Tera.ch” category showed that individuals are interested in their own community.

By applying the Girvan-Newman algorithm through Vizster, Nano-Tera.ch can divide individuals into 4 main clusters where relations represent implicit connections. Clusters’ size showed that Cluster #2 is the biggest, followed by the Clusters #3, #4, and #1, with their main categories on “Health”, “Security”, “Environment”, and “Nano-Tera.ch” respectively. The results showed that Nano-Tera.ch community is strongly divided into its main categories and that individuals within a category have a strong connection between them. The overlaps highlight individuals that connect categories and are valuable nodes. In fact, those individuals turn out to be some of the most influential people in the community and with a high number of interactions. In the middle, Nano-Tera.ch has the most valuable individual, implicit connected to all clusters and to a high number of individuals.

The final value on the CD allows communities to place implicit connection at the same level and compare them. The CD model assigns to each connection a CD value that describes how strong two individuals are related. CD is established between any two of the main researchers at Nano-Tera.ch. Although they were connected on their research area, the CDs on the cluster C show that the category has the strongest CD. This fact can be translated to individuals being high related as a team. Moreover, the result can be explained by the low interest on the other categories rather than “Security”. The CD on the Cluster C is also high because individuals at security projects have clicked on security links but ignored the other category links. It is also interesting to note that CDs support the Girvan-Newman clustering algorithm by showing that individuals in a group have a strong CD between them. Even with a sample of all implicit connections, the clustering algorithm continuous to cluster individuals into the four main categories: “Nano-Tera”, “Security”, “Health”, and “Environment”.

Nano-Tera.ch used these results to promote collaboration between researchers and to present to each research the people that showed interest on their work. Nano-Tera.ch also identified the main followers and asked them to promote the newsletters and to help reaching the maximum of researchers. In addition, the results were also used to promote conferences on a topic that researchers showed interest and to answer simple questions such as the number of interested people that may attend.

5 Related Work

The main work on using email as a source to extract information started with Schwatz and Wood using email headers to extract shared interests between people through graph theory [21]. However, that approach is very limited to the emails’ headers and does not take into account the message’s body and the subject, which can be the richest source on individuals’ interests. PeCo (Ogata and Yano, 1998) [18] collected users’ relationship through email headers (“From”, “To”, and “Subject”) but as well as the previous solution does not focus on discovering and explore implicit connections in a network. On the other hand, McArthur and Bruza discovered implicit connections by mining semantic associations from people’s communications [11]. They proposed a model called HALe that automatically creates a dimensional representation of words

based on the email corpus and uses it to discover a network of people implicitly connected. However, that solution does not have any measure connections' value.

Together with the referred works there are several track engines used for marketing purposes, which are also able to send newsletters and track users' interactions. An example is the system developed by Foulger, Chipperfield, Cooper and Storms [7]. The system generates an email template and uses it to track all the receivers' interactions. However, the detection of key user through a connection degree can be harder to achieve or difficult to understand once the model works in a black box.

Barão and Silva proposed an holistic and complex model to define the Relational Capital Value (RCV) of organizations as well as online communities [2, 3]. Explicit and also implicit relational connections (such as these discovered by the NTS) are important for the RCV model application, hence for the determination of online communities relational value.

6 Conclusion

The NTS allows communities to improve the quality of their knowledge on individuals' relationships by introducing a web system able to send newsletters and gather individuals' interactions. Based on explicit and implicit connections NTS is able to bring to light hidden relationships and to measure their CD trough the CD model.

At the end, communities are able to foster an implicit community and to explore individuals' connections based on analysis tools like NTS or exporting the data to external tools for further analysis. The NTS brings value on its ability to capture and expose individuals' implicit relationships through a network. By designing newsletters, communities are able to better understand individuals and to improve the way they explore individuals' implicit connections. We believe that the NTS and the CD model can help communities to have a more valuable overview of their network.

Acknowledgements. This research was supported by the Strategic Executive Committee of Nano-Tera.ch which is a program financed by the Swiss Government. Special thanks to all Community Knowledge Development team Dr. Peter Bradley, Dr. Nitesh Khilwani, and Madhur Agrawal. Also to Prof. Chris Tucci from CSI/EPFL who supported and trigged the project and to Mariana Araújo who boosted the newsletters' design study.

The study was also support by national funds through FCT – Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2011.

References

1. Aggarwal, C.: An introduction to social network data analytics. Springer Science And Business Media, LLC 2011 (2011)
2. Barão, A., Silva, A.: A Model to Evaluate the Relational Capital of Organizations (SNARE-RCO), Conference of Enterprise Information Systems (Centeris'2011), Springer (2011)

3. Barão, A., Silva, A.: How to value and monitor the relational capital of knowledge-intensive organizations?, *Research on Enterprise 2.0: Technological, Social, and Organizational Dimensions*, IGI Global (2012)
4. Bell, G.: *Building Social Web Applications*. O'Reilly Media, Inc. (2009)
5. Bellotti, V., Ducheneaut, N., Howard, M., Smith, I., E. Grinter, R.: Quality Versus Quantity: E-Mail-Centric Task Management and Its Relation With Overload. *Human-Computer Interaction*, vol. 20, pp. 89-138. , Lawrence Erlbaum Associates, Inc. (2005)
6. CKD research project: <http://www.nano-tera.ch/members/263.php>
7. Foulger, M., Chipperfield, T., Cooper, J., Storms, A.: System and method related to generating and tracking an email campaign. IC Planet (2006)
8. Harley, J., Blismas, N.: *An Anatomy of Collaboration Within the Online Environment*. Springer-Verlag Berlin Heidelberg 2010, 14-34 (2010)
9. i* Wiki: <http://istar.rwth-aachen.de>
10. m. boy, d., B. Ellison, N.: Social Network Sites: Definition, History, and Scholarship. In: *Journal of Computer-Mediated Communication*, vol. 13, pp. 210-230 (2008)
11. McArthur, R., Bruza, P.: Discovery of implicit and explicit connections between people using email utterance. In: *Kluwer Academic Publishers*, pp. 21-40 (2003)
12. Heer, J., Boyd, D.: Vizster: Visualizing Online Social Networks. In: *2005 IEEE Symposium on Information Visualization (2005)*
13. M. Leiner, B., G. Cerf, V., D. Clark, D., E. Kahn, R., Kleinrock, L., C. Lynch, D., Postel, J., G. Roberts, L., S. Wolff, S.: The past and future history of the Internet. In: *Communications of the ACM*, vol. 40, pp. 102-108, New York (1997)
14. M Ridings, C., Gefen, D., A0072inze, B.: Some antecedents and effects of trust in virtual communities. In: *The Journal of Strategic Information Systems*, vol. 11, pp. 271-295 (2002)
15. Musser, J., O'Reilly, T.: *Web 2.0 Principles and Best Practices*. O'Reilly Media, Inc. (2006)
16. Nano-Tera.ch: <http://www.nano-tera.ch/topdownbottomup/index.html>
17. NodeXL: <http://nodexl.codeplex.com/>
18. Ogata, H., Yano, Y.: Collecting organizational memory based on social networks in collaborative learning. In: *WebNet*, pp. 822-827 (1998)
19. O'Reilly, T.: What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. In: *O'Reilly Media, Sebastopol (CA) USA*, pp. 17-37 (2007).
20. Papacharissi, Z.: *A Networked Self-Identity, Community and Culture on Social Network Sites*. Routledge (2010)
21. Schwartz, M., Wood, D.: Discovering shared interests among people using graph analysis of global electronic mail traffic. In: *Communication of the ACM* (1993)
22. Swan, K.: Building Learning Communities in Online Courses: the importance of interaction. In: *Education, Communication & Information*, vol. 2 (2002)
23. Wellman, B., Boase, J. Chen, W.: The Networked Nature of Community: Online and Offline. In: *It&Society*, vol. 1, pp. 151-165 (2002)
24. Yu, E.: *Modelling Strategic Relationships for Process Reengineering*. Doctoral Dissertation, University of Toronto (1996)
25. Zaphiris, P., Ang, C.: *Social Computing and Virtual Communities*. Chapman and Hall/CRC, 1^a Edition (2009)

Appendix: Newsletter Tracking System Screenshots

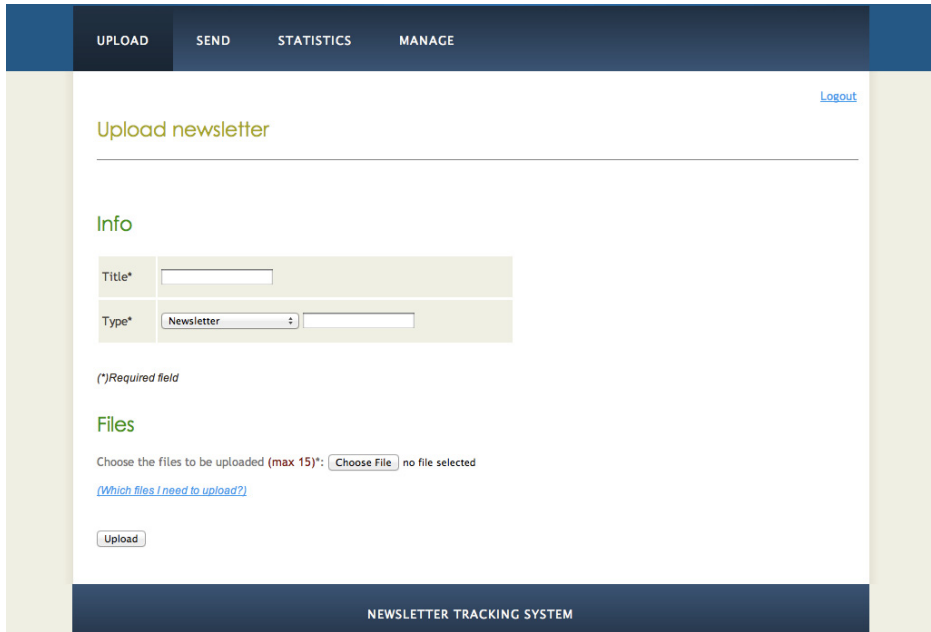


Fig. 8. NTS Upload Page

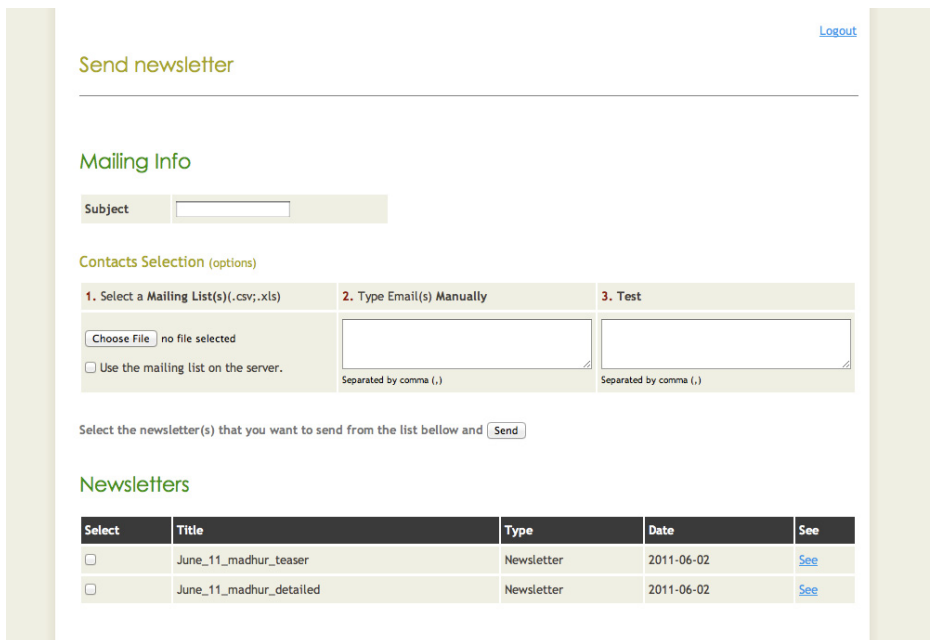


Fig. 9. NTS Send Page

Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA

Uwe Breitenbücher, Tobias Binz, Oliver Kopp,
Frank Leymann, and David Schumm

Institute of Architecture of Application Systems, University of Stuttgart, Germany
Universitätsstraße 38, 70569 Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Abstract. A major difficulty in enterprise computing is the modeling of complex application topologies consisting of numerous individual components and their relationships. Especially in the context of cloud computing, the Topology and Orchestration Specification for Cloud Applications (TOSCA) has been proposed recently for standardization to tackle this issue. However, TOSCA currently lacks a well-defined visual notation enabling effective and efficient communication in order to transport the semantics of the encoded information to human beings. In this paper, we propose a visual notation for TOSCA based on established usability research which provides additional concepts for visual modularization and abstraction of large application topologies.

Keywords: TOSCA, Modeling, Visual Notation, Application Topologies.

1 Introduction

Cloud computing enables significant benefits in terms of cost, flexibility, and scale compared to traditional IT. An important issue is the automation needed to achieve these advantages. The Topology and Orchestration Specification for Cloud Applications (TOSCA [9]) provides a well-defined way to model composite applications and to provide plans for automating their management [4].

Visual notations enable an effective communication because of the powerful and highly parallel human visual system [7]. A well-designed visual notation eases the comprehension of the content structure and enables an easier navigation. In addition, visual notations are generally easier to learn and can be remembered faster than textual syntax. Thus, they are appropriate to complement languages which only provide textual notations such as XML. This leads to a separation of concerns: Visual notations are used for fast and effective communication, the original notation for the actual purpose and more detailed information processing. However, TOSCA does not specify a visual notation to map the language constructs to visual elements. The lack of a visual notation in other specifications resulted in a number of different graphical renderings of the same model. One

example is the Business Process Execution Language (BPEL) [8] for which different approaches to visualize elements regarding their shapes, icons, layout, etc. exist. This becomes a problem when diagrams must be communicated between people using different representations as it might lead to misunderstandings and wrong interpretations. Thus, we advocate using only one common visual notation in addition to the actual notation provided by the original language.

A major problem of existing visual notations is that they currently correspond to what Alexander [1] calls an unselfconscious design culture: The design rationales are not based on explicit design principles. They are based on instinct, imitation, and tradition. As a consequence, many visual notations, like UML, focus on semantics and lack an explicit design process for the visual syntax which results in problems decreasing the usability of the notation. Therefore, we present a Visual Notation for TOSCA (Vino4TOSCA) which is based on an explicit requirements analysis regarding human cognition, usability, ergonomic influences, and evidence-based principles. The remainder of the paper is structured as follows: In Sect. 2 we describe the fundamentals and related work whereon the requirements for the notation in Sect. 3 are based. Section 4 describes the notation and Sect. 5 concludes the paper and gives an outlook on future work.

2 Fundamentals and Related Work

TOSCA [9] is a language to formally describe cloud applications and their management. The structure of an application is captured by a so-called Topology Template, a graph with Node Templates and Relationship Templates, serialized in XML. Node Templates represent the components of an application, for example, an “application server”. Relationship Templates define how a particular node relates to another node, for example, the “application server” node is “hosted on” an “operating system” node. Templates are typed with Node Types and Relationship Types respectively. Types define the meaning of the nodes and relationships by specifying their properties and states of their lifecycle. TOSCA additionally defines policies on nodes, management operations provided by the node, and deployment artifacts implementing the functionality of the node. TOSCA does not define concrete Node Types and Relationship Types as it only provides a way to model them and to compose templates of several individual types into a topology. Therefore, the modeler of an application is able to define new Node Types and new Relationship Types. Node Types can be provided by software vendors as building blocks to simplify the integration of their products into cloud applications. The operational aspects, key for each automated cloud environment, are captured in so-called plans which are workflows capturing the management tasks of an application. The management operations defined by nodes are orchestrated by plans into higher level management functionalities, like deploying or scaling up the application. This enables software developers to model their management knowledge and experience explicitly into these plans which enables operating an application without having all the deep technical knowledge required before.

Moody [7] contributed a design theory, called “The Physics of Notations”, focusing on the physical perceptual properties of notations regarding human capabilities. The principles defined in this design theory are synthesized from theory and empirical evidence. They are based on a theory of how visual notations communicate and provide the basis for the development of Vino4TOSCA.

Existing enterprise architecture modeling languages and notations such as Acme [5] do not provide a visual notation which can be used for TOSCA, because they do not fulfill all requirements we introduce in Sect. 3 and consider as absolutely necessary. In addition, the concept of managing applications by plans differentiates TOSCA fundamentally from other application modeling languages and thus needs special consideration.

3 Requirements Analysis

In this section we present requirements and design principles on the visual notation which are necessary for an effective usage. They have been identified, discussed, and validated by TOSCA users and members of the OASIS TOSCA Technical Committee. In the following sections we use the symbol R_x , with x being the number of the reference, to reference a certain requirement.

The prescriptive component in [7] defines nine principles for designing cognitively effective visual notations to increase speed, ease, and accuracy with which information can be understood by humans: R1 Semiotic Clarity, R2 Perceptual Discriminability, R3 Semantic Transparency, R4 Complexity Management, R5 Cognitive Integration, R6 Visual Expressiveness, R7 Dual Coding, R8 Graphic Economy, and R9 Cognitive Fit.

TOSCA-related requirements address the semantic constructs to obtain a tailored notation: R10 Completeness (all information contained in Topology Templates must be representable), R11 Semantic Correctness (a valid Vino4TOSCA diagram has to be a representation of a valid TOSCA Topology Template), R12 Extensibility (be extensible to show additional information), R13 Compact Representation (support compact visual representation to tackle space problems).

The following requirements shall improve usability and user experience to achieve a broad acceptance and user satisfaction. They are inspired by usability standards (e. g., EN ISO 9241) and [10]. R14 Suitability for the Task (optimization for modeling TOSCA Topology Templates), R15 Self-descriptiveness (diagram and graphical symbols describe their meaning themselves), R16 Simplicity (graphical elements must be easy and fast to draw), R17 User Satisfaction (account for human preferences and enable visually appealing designs).

4 The Notation

Vino4TOSCA covers the modeling of TOSCA Topology Templates by Topology Template Diagrams which mainly consist of Node Templates, Relationship Templates, and Groups (R11, R14). Modeling of plans is not part of the notation as there are already existing languages and notations available (e. g., BPMN).

The notation allows defining profiles which are domain-specific visual languages devised for specific needs, knowledge, and capabilities of users in a certain application domain [6] (R9, R17). The main advantage of profiles is that tailoring enables a strong cohesion to the domain properties and being effective and intuitive for the task to be performed, e. g., a “Whiteboard Profile” defines how to draw a Topology Template Diagram effectively by hand while an “Electronic-Design Profile” can be used for creating a modeling software supporting more details (R14). Profiles are allowed to constrain, but not to structurally modify the notation or change its basic shapes. The notation provides visual variability for profiles as described in Sect. 4.1 (R8), e. g., profiles may forbid the usage of groups or define line colors. Some shapes offer Additional Information Areas which can be used by profiles to add any information which is not natively reflected by the notation (R10, R12, R15). These areas may contain any text or graphic. Nevertheless, a profile has to regard the requirements defined in Sect. 3, too.

To reduce the complexity of large diagrams, the notation must provide mechanisms to group multiple elements visually into one single element. TOSCA Group Templates are not sufficient as they are applied at the TOSCA model level. Thus, grouping would influence the effective application topology. Therefore, the notation provides two additional concepts for visual grouping which are not part of TOSCA itself: Visual Group and Visual Relationship Group. Both may reduce complexity as they enable visual modularization and abstraction as well as reducing the number of symbols by collapsing (R4, R13). The Visual Group may also be used for integrating external diagrams homogeneously (R5).

4.1 Visual Design Rationales

As the basic shapes of the notation must not be changed by profiles (R1) and there is a need for a hand-drawable “Whiteboard Profile”, the notation has to tackle human drawing issues (R16). We decided to use rounded shapes wherever possible because of human drawing skills and preferences [2].

The notation uses the eight elementary visual variables of the Design Space defined by Bertin [3] to visually encode information: Horizontal and vertical position, shape, size, color, brightness, orientation, and texture (R6). They are classified into three categories: (i) Fixed variables defined by the basic notation, (ii) constrained variables defined by profiles, and (iii) free variables. While the first two ones are defined strictly and must be followed when applying the notation (limitations and constraints), free variables can be used for individual modeling of concrete diagrams (points of variability). The fixed visual variables are the retinal variables shape, orientation, and texture of lines: The basic shapes, their surrounding lines, and orientation are defined strictly and must not be changed by profiles or instantiation. All other variables are free if they are not constrained by a profile. Thus, if a profile does not constrain the visual variable color it is also a free variable and different colors can be used wherever the basic notation allows it. A profile is allowed to constrain the retinal variables color, value, texture, and size for enabling a high value of cognitive effectiveness. Constraining variables by profiles enables adding new semantics for different tasks

and/or audiences by creating visual dialects. Generally free visual variables are the horizontal and vertical position of an element.

The basic notation employs icons for describing elements as they have a higher information density and need less space for information presentation than text (R3, R6, R7). Icons are recognized, processed, remembered, and learned more easily and faster than textual information and preferred to abstract shapes by humans [2]. They enable tailoring domain-specific visual languages by using different icons for each domain. All icons are placed on the top most left position of the visual element because humans spend most of their attention to this place.

Text fonts are not defined by the basic notation. This may be done by profiles as especially hand drawn profiles need this variability. To enable fast recognition of textual information, text used to identify semantic elements differs in visual appearance: A name is not decorated, an id is underlined, and the name or id of the corresponding element type is enclosed by two parentheses (R3, R7, R15).

The notation does not define shapes for all elements of TOSCA, e. g., there are no shapes defined for types (e. g., Node Types). This can be modeled as additional information contained in the Additional Information Areas of the template shapes.

4.2 Shapes

This section defines the visual syntax of the notation, i. e., the visual elements and shapes. For each visual element we describe its shape in terms of form, contained information, semantics, variability points, and visual design freedom.

The Node Template Shape shown in Fig.1 represents the Node Template as a rectangle with rounded corners surrounded by a solid line. There are five possibilities to describe the corresponding Node Template: (i) An icon contained in the Icon Area may represent the Node Template or the corresponding Node Type, (ii) using name or (iii) id of the Node Template to identify it, (iv) using name or (v) id of the corresponding Node Type to identify its type. A valid Node Template Shape contains at least one of these five information items. If multiple textual variants are combined, the visual order is given by the vertical ordering in Fig.1 (R7, R13). The Additional Information Area is a rounded rectangle surrounded by a solid line which may be optionally attached below the main shape to provide any additional visual information (variability point). It is positioned behind the main shape so that both upper corners are hidden, as depicted in Fig. 1. The Icon Area is allowed to contain any graphic or symbol. The main shape is allowed to contain any graphic as background image, i. e., behind the Icon Area and the text blocks. Thus, various designs are possible as well as monochrome-colored Node Template Shapes. The surrounding lines are allowed to be colored, but the solid style must not be changed (R1).

The Relationship Template Shape shown in Fig. 2 represents the Relationship Template as a single line with a small shape at each end (shown as question marks in Fig. 2), e. g., an arrow. It visually connects any two *Relational Elements*, which are Node Template Shape, Collapsed Group Template Shape, and Collapsed Visual Group Shape. The semantics of a Relationship Template Shape sourcing

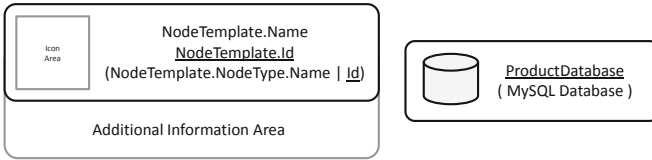


Fig. 1. Node Template Shape and example

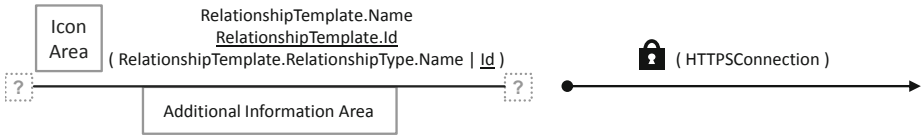


Fig. 2. Relationship Template Shape and HTTPSCollection example

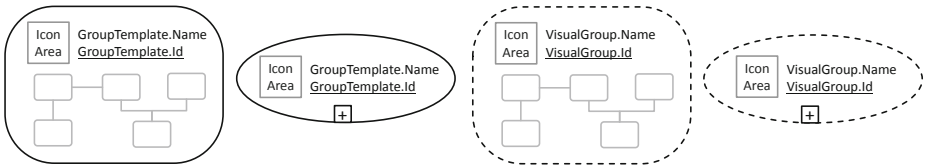


Fig. 3. Expanded / Collapsed Group Template Shapes and Visual Group Shapes



Fig. 4. Expanded and Collapsed Visual Relationship Group Shapes

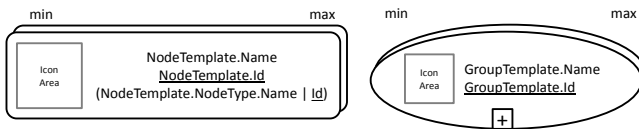


Fig. 5. Multiple instances of Node Template and Group Template

or targeting a Collapsed Visual Group Shape or Collapsed Group Template Shape is that it points to a hidden Relational Element inside the group. The basic notation neither defines a special line style nor shapes which may be used at the line endings. It is only prohibited to use the same dashed line style as the Visual Group Shapes (R2). The two shapes at the line endings may use any free visual variable. There are five possibilities to describe the corresponding

Relationship Template: (i) An icon contained in the Icon Area representing the Relationship Template or the corresponding Relationship Type, (ii) using name or (iii) id of the Relationship Template to identify it, (iv) using name or (v) id of the corresponding Relationship Type to identify its type (R7). The Icon Area is placed on the left of the textual information which is positioned above the line if it is horizontal or at any side if the line is vertical or diagonal. A valid Relationship Template Shape contains at least one of these five information items. The visual order of the elements and an HTTPSCConnection example are shown in Fig. 2. The Additional Information Area is a rectangle surrounded by a solid line which may be attached below touching the line if it is horizontal or sideways otherwise in order to provide additional information about the template.

The Group Template is represented by two different shapes shown in Fig. 3 on the left: The Expanded Group Template Shape is a solid line surrounding elements which are grouped. There is no special shape defined. At the top left position there is the possibility to describe the Group Template Shape by a left-aligned icon or textually by using a name or id. All combinations of them are allowed but at least one has to be used. The collapsed Group Template Shape is an oval surrounded by a solid line. A small square with a plus sign positioned at the bottom center of the shape indicates the collapsed state hiding the contained elements. Inside the oval there must be at least one of the following: An icon, the name, or the id of the Group Template (R7). The free usage of the Icon Area and color of the surrounding line are the only free variables. The background of these two shapes must not be filled with any color or image.

The Visual Group Shapes shown in Fig. 3 on the right are equal to Group Template Shapes with the difference that the surrounding lines are dashed. The semantics is to group elements visually only. The shapes may also be used to represent the integration of other diagrams (R5): Its name or id is used to identify the integrated diagram, especially in collapsed state. The visual variability is equal to Group Template Shapes.

The two Visual Relationship Group Shapes shown in Fig. 4 are used to visually group and hide Relationship Template Shapes connecting Relational Elements. The expanded variant consists of two dashed lines between any two Relational Elements and at least contains two Relationship Template Shapes. The element can be described by a left-aligned icon or textually by using a name or id of the group above the lines. All combinations of them are allowed but at least one has to be used. The Collapsed Visual Relationship Group Shape (right shape in Fig. 4) is a dashed line with a small square containing a plus sign inside positioned at the center of the line connecting any two Relational Elements. The shape's semantics is that it visually groups and hides the Relationship Templates between two Relational Elements. Above the line there must be at least one of the following: An icon, the name, or id of the group (R7). The free usage of the Icon Area and the color of the lines are the only free variables which can be used to design these two shapes. The background must not be filled with any color or image.

TOSCA Node Templates and Group Templates have two attributes representing the number of allowed instances, e. g., multiple instances of a service component: min and max. Multiple instances are represented by drawing a second solid line partly covered by the original shape and writing the min value at the left and the max value at the right above the main shape as shown in Fig. 5.

5 Conclusion and Outlook

We presented a Visual Notation for TOSCA based on a scientific development approach taking the “Physics of Notation” theory and a well-defined requirements analysis into account. The presentation includes a visual model which explicitly defines the elements, relations, and representations. For the time being, an evaluation of the notation was not possible for the following reasons: First, TOSCA has been published quite recently and the users of this language are collecting experience with the language itself just now. Second, to judge the expressiveness of the visual model, one needs to really work with them. Therefore, we implemented the proposed visual notation in an open source web-based TOSCA modeling environment prototype¹ in the CloudCycle² project, which is one early adopter of TOSCA. After a broader usage, we will evaluate the notation using a questionnaire. In addition, we plan to evaluate several new profiles as well as to develop a new diagram type to integrate the proposed notation with process modeling notations such as BPMN. On the official Vino4TOSCA Web page³ we present application examples, profiles, and the meta-model of the notation.

Acknowledgments. This work was partially funded by the BMWi project CloudCycle (01MD11023).

References

1. Alexander, C.: Notes on the Synthesis of Form. Harvard University Press (1964)
2. Bar, M., Neta, M.: Humans prefer curved visual objects. *Psychological Science* 17(8), 645–648 (2006)
3. Bertin, J.: Semiology of graphics. University of Wisconsin Press (1983)
4. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. *IEEE Internet Computing* 16(03), 80–85 (2012)
5. Garlan, D., Monroe, R.T., Wile, D.: Acme: Architectural Description of Component-Based Systems. In: *Foundations of Component-Based Systems*, pp. 47–68. Cambridge University Press (2000)
6. de Lara, J., Vangheluwe, H.: Defining visual notations and their manipulation through meta-modelling and graph transformation. *J. Vis. Lang. Comput.* 15(3-4), 309–330 (2004)
7. Moody, D.L.: The “physics” of notations: a scientific approach to designing visual notations in software engineering. In: *ICSE*, pp. 485–486 (2010)

¹ <http://www.cloudcycle.org/en/valesca/>

² <http://www.cloudcycle.org/en/>

³ <http://www.vino4tosca.org>

8. OASIS: Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)
9. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 Working Draft 07 (June 2012), <http://www.tosca-open.org>
10. Petre, M., de Quincey, E.: A gentle overview of software visualisation. Psychology of Programming Interest Group (PPIG) (September 2006)

BOINC-MR: MapReduce in a Volunteer Environment*

Fernando Costa, Luís Veiga, and Paulo Ferreira

Distributed Systems Group, INESC-ID
Technical University of Lisbon
R. Alves Redol, 9
1000-029 Lisboa, Portugal
fcosta@gsd.inesc-id.pt

Abstract. Volunteer Computing (VC) harnesses computing resources from idle machines around the world to execute independent tasks, following a centralized master/worker model.

We present BOINC-MR, a system able to run MapReduce applications on top of BOINC, the most popular VC middleware in existence. We describe BOINC-MR's architecture and evaluate its performance with a typical MapReduce application. Our results show that BOINC-MR yields a performance increase of 64% in application turnaround time and close to 50% reduction in bandwidth usage in the server side, when compared to the unmodified BOINC system.

Keywords: Volunteer Computing, MapReduce, Adaptive Middleware.

1 Introduction

The use of personal computers' computational power as a tool for science has steadily increased in popularity. To this end, Volunteer Computing (VC) systems have been extremely successful in bringing large numbers of donated compute cycles together to form a large-scale virtual supercomputer. Applications running on this infrastructure tackle problems from a wide range of scientific subjects, from physics to biology, and are tailored for highly parallel number-crunching computations.

BOINC [2] is a VC middleware that currently supports over 40 projects and bolsters a user base of around 450 thousand active machines, making it the most popular system in the world, rivaling the world's supercomputers in computing power. In its current implementation, the network topology is restricted to a strict master/worker scheme, generally with a fixed set of centrally managed project computers distributing and retrieving results from network participants.

* This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113993/2009 and PEst-OE/EEI/LA0021/2011.

Such a centralized architecture is the source of a potential bottleneck in the continuing evolution of Volunteer Computing systems. As projects gain in popularity and their user-bases expand, network and storage requirements can easily become more demanding, thus increasing the load on the server(s). There are worrying signs of stagnation in the number of active users and projects [1], and emerging problems in data distribution and storage [6].

Thus, one must look at alternative computing paradigms that may help Volunteer Computing reach its untapped potential. MapReduce is a widely used computing paradigm, proposed by Google [8], that has obtained considerable support in Cloud Computing communities due to its simplicity, scalability and performance in commodity clusters.

Our goal is to support MapReduce on top an insecure, unreliable VC environment, by taking advantage of the vast improvements in network infrastructure and disk storage in the last mile of the Internet. In this paper, we present BOINC-MR, a BOINC prototype that can run MapReduce jobs, and evaluate its performance in a real-world scenario.

This paper is organized as follows: Section 2 gives background on BOINC and MapReduce; Section 3 discusses the concepts we have just mentioned in more depth; experimental results are presented in Section 4; Section 5 introduces related work; and Section 6 concludes.

2 Background

This section introduces the BOINC system and MapReduce programming paradigm we based our research on.

In order to distribute its work units, each BOINC [2] project has to build its data and executable code as well as setup and maintain their individual servers and databases. Result validation is obtained through the use of task replicas, so that upon task completion, a quorum must be reached by a majority of users before an output can be considered correct.

Most Desktop Grids, such as BOINC or XtremWeb [3], have centralized architectures, in which a server or coordinator is responsible for scheduling task execution. There are exceptions [5], but they are either insignificant in scope or tailored to a different environment.

Such architectures and the limited support for complex applications may have brought on a significant problem: the number of active projects has stagnated. This in turn has led to a 15% decrease in the number of active users [1], a number that is expected to dwindle unless new alternatives are presented that may spark the interest of volunteers.

MapReduce is a software framework for parallel data-intensive computations recently popularized by Google [8]; it is able to represent a wide range of applications, by providing an abstraction for parallel execution ("map") and aggregation of results ("reduce").

MapReduce input is initially split into several chunks, each to be executed by an independent "map" task, assigned to each worker by a centralized master

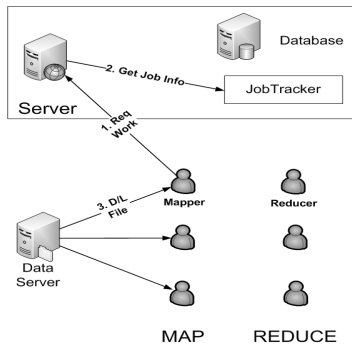


Fig. 1. BOINC-MR Map Phase

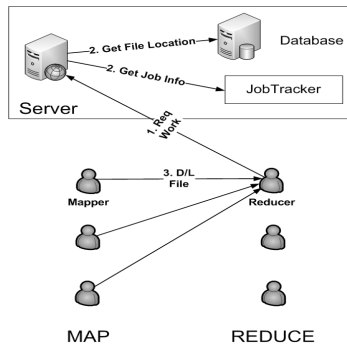


Fig. 2. BOINC-MR Reduce Phase

node. Each worker node processes the map task it was given, and reports its completion to the master. For the "reduce" step, a predetermined number of reduce tasks are created, whose goal is to perform join operations on the map outputs. All reduce inputs are therefore outputs from the previous map task. Throughout the rest of the paper, we will be referring to them as map outputs. Once all map outputs have been downloaded, the reduce task is executed and its final result is saved.

3 BOINC-MR Architecture

BOINC-MR supports the MapReduce paradigm in an unreliable, unsecured Internet environment. One of our main goals was to improve performance when adapting MapReduce to BOINC. In order to achieve this, BOINC-MR decentralizes data distribution and removes unnecessary overhead from the central server by leveraging inter-client transfers. MapReduce is an ideal framework to evaluate the impact of our proposition, since the map stage produces intermediate results that are used as input by reduce tasks.

Map tasks are embarrassingly parallel, with no dependencies or any shared data between them, which allowed us to use the traditional scheduling mechanism when dealing with this step. The BOINC-MR map stage is shown in Fig. 1: (1) A user (mapper) starts by requesting work from the projects central server; (2) The server takes into account the workload of each mapper, as well as its hardware and availability information, and dispatches work units that fit the request; (3) Each mapper downloads input and executable files from the data server, and runs the application. The computation results are then returned to the central server. The server keeps track of which mapper is holding each output file by storing that information in the database.

The reduce stage is depicted in Fig. 2: (1) A user (reducer) requests work from the projects central server; (2) The scheduler appends to each reduce task information the address (IP and port) of mappers holding output for the same job; (3) The reducer then has the possibility of downloading the required input

files directly from the mappers. The server also stores a copy, thus providing a failover mechanism in case of error and guaranteeing data availability. After downloading all required files, each reducer executes its task and returns the final result to the server.

3.1 BOINC-MR Client

A BOINC-MR client requests work by sending the server a message with host characteristics and other information necessary for task scheduling. If there is work available, the server reply includes information on the task to be executed (mentioned in step (2) of Fig. 1 and 2). This task information allows clients to identify which tasks belong to MapReduce jobs.

Once a map task is obtained, the BOINC-MR client acts as mapper and runs the executable to produce the results. Mappers who have finished their task make their output available for reducers to download. We consider mappers to be hosting map outputs for as long as the files are available for download. A BOINC-MR client only accepts incoming requests for its output files, while rejecting messages that do not comply to a predefined file request template. Each mapper will stop accepting connections if one of the following situations occur: the BOINC-MR client is shut down; the MapReduce job has completed successfully; or the mapper has reached a timeout in total hosting time.

If a BOINC-MR client obtains a reduce task, it becomes a reducer. After parsing the task information (sent in step (2) of Fig. 2), the reducer is able to identify which map output files can be downloaded directly from mappers and which files are only available in the server. The BOINC-MR reducer always attempts to download from a mapper before resorting to the server. Each map output file may have multiple mappers hosting it, so the reducer goes through each mapper in the list in order. The mapper address list is ordered randomly at the server side, to prevent the overloading of a single BOINC-MR mapper.

We use a fall back mechanism for failed inter-client downloads. After n failed attempts to download an output file directly from mappers, the reducer resorts to downloading all missing files from the server.

3.2 MapReduce in BOINC Server

The BOINC-MR server must ensure a timely and valid transition between map and reduce steps. It must be able to deal with both BOINC-MR and BOINC clients, and provide information that allows each client to handle each task according to its characteristics. In order to differentiate map tasks from "normal" (non MapReduce) ones, we modify their templates by adding "*<mapreduce>*" tags with additional information such as job id and stage.

The BOINC-MR server uses an additional general configuration file to coordinate between stages and handle task creation. This file is responsible for defining global MapReduce parameters for each job, such as the number of map and reduce tasks. The server uses a dynamic work unit creation mechanism, which is activated as soon as all map tasks have been returned and validated.

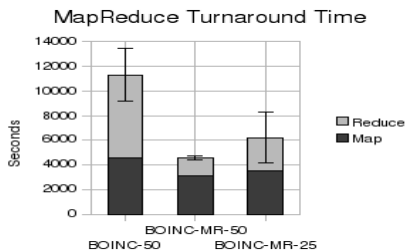


Fig. 3. Weight of Map and Reduce stage in MapReduce job

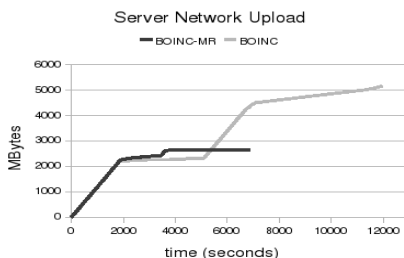


Fig. 4. Network Upload Bandwidth on server with 50 nodes

This mechanism takes all the information provided by the mappers hosting file outputs to edit the necessary templates and insert reduce work units into the server’s database.

Therefore, all reduce tasks sent to BOINC-MR clients (reducers) have the location of the required input data, as IP addresses of mappers and as the projects data server address (URL). It is worth noting that providing the server URL allowed us to guarantee retro-compatibility with unmodified BOINC clients.

4 Experiments

To evaluate our prototype, we use PlanetLab [4], a distributed testbed designed for applications deployed over the Internet. We present the results and implementation details in this section.

We use either 25 or 50 PlanetLab nodes as clients, and one node to act as server. To evaluate our scenarios, we create a BOINC project to run the word count MapReduce application. In word count, each map task receives a text document as input, counts the number of words in it and outputs an intermediate file with “word 1” pairs for each word found. The reduce step collects all the map intermediate outputs and aggregates them into one final output. In our experiments, we use an initial input file of 1 GB, divided into 100 chunks (10MB), each to be handled by a different map task.

Our goal is to measure the performance of BOINC-MR when running MapReduce applications, especially in two axis: application turnaround and server bandwidth usage.

4.1 Network Bandwidth and Application Turnaround

We use either 25 or 50 client nodes, while the BOINC-MR server replicates each task twice. Figure 3 shows the application turnaround time results. The BOINC column corresponds to a scenario with 50 unmodified BOINC clients. BOINC-MR-50 and BOINC-MR-25 represent 50 and 25 nodes, respectively, running BOINC-MR. While the difference in the Map stage is not significant, with

BOINC-MR doing slightly better, the Reduce stage shows remarkable improvements. This speedup is due to the fact that BOINC-MR employs inter-client transfers, and because the server spends more time uploading files with BOINC clients. Therefore, it has a higher chance of experiencing higher load due to other images running on its PlanetLab node. With respect to BOINC-MR clients, 25 nodes (BOINC-MR-25) performed worse than 50 nodes (BOINC-MR-50) in the reduce stage. This was due to a smaller number of nodes hosting the map output files. Overall, BOINC-MR takes less than half the time (46%) needed by the unmodified BOINC to complete the MapReduce job.

In order to more accurately evaluate the overhead on the server, we measure its bandwidth usage when running BOINC-MR clients. We do not present the results of network traffic from clients to the server since the server downloads the same amount of data from either BOINC-MR or BOINC clients. In both cases, the server has to download the map and reduce output from each client.

Figure 4 presents the data uploaded by the central data server, when using either BOINC-MR or BOINC clients. As BOINC-MR is faster than BOINC, its experiment ends earlier, after 7000 seconds, while BOINC clients only finish at the 12.000 second mark. We can observe an initial increase in uploaded data in both scenarios, which corresponds to the distribution of map inputs from the server to the clients. Around second 2000, the map tasks seem to have been deployed since we reach a plateau in both scenarios, which is only interrupted when the reduce step begins. The server running with BOINC-MR clients has a slight increase around second 4000 when it starts uploading the reduce task executable file to clients. On the scenario with BOINC clients, however, we can witness a steep slope in upload bandwidth from the central server to clients around second 5000. The server, being the sole owner of reduce input files, must upload all the data to clients. The server in the BOINC-MR scenario ends up with around 2600 MBytes of uploaded data. On the other hand, using BOINC clients forces the server to upload close to 5200 MBytes. This means that the BOINC-MR client can cut the server's upload bandwidth consumption in half.

4.2 Replication Factor

At this point, we evaluate the impact of the replication factor on the map task. Figure 5 shows the results for our experiments with 2 and 3 replicas for the map task. BOINC-2 and BOINC-3 represent the original BOINC client, with a replication factor of 2 and 3, respectively. BOINC-MR2 and BOINC-MR3 are the corresponding BOINC-MR clients. For the unmodified BOINC, using a higher replication factor helped speedup both the map and reduce stage. This can be explained by the lower impact of a slower node. With only 2 replicated tasks, both results are needed to validate the work unit, which means that any node holding a task can slow down the whole computation by not returning it in time. Having 3 nodes makes the slower one redundant.

In the BOINC-MR client scenarios, we can see a slight speedup in the map stage which is attributed to the aforementioned lower impact of slower nodes. However, using 3 map task replicas produced worse results in the reduce step.

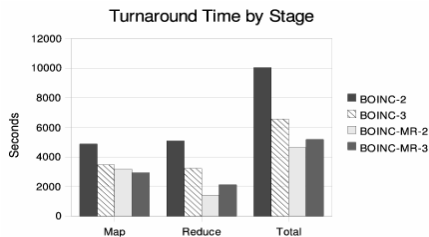


Fig. 5. Turnaround time of BOINC and BOINC-MR clients

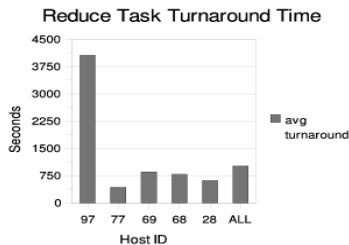


Fig. 6. Reduce task execution time in different hosts

This was unexpected since having 1 more mapper to download map outputs from should improve inter-client transfer speed. There are two explanations for these results. First, the current version of our prototype does not use any heuristic or complex algorithm when choosing which node to download each map output from. Secondly, we can witness a recurring event that is presented in Fig. 6. In cases with low replication such as this (2 reduce task replicas), one reducer's output cannot be discarded as there is no third or fourth reducer running. If a slower reducer is able to obtain several tasks it will reduce the application turnaround time. In Fig. 6, we can see that node 97 is 5 times slower than any other node. This means that, even after all the other reducers have returned their output, the MapReduce job will only end when this node returns its results.

5 Related Work

Combining the concepts of Cloud and Volunteer Computing has already been proposed in [9], in which the authors studied the cost and benefits of using clouds as a substitute for volunteers or servers.

There are two projects that have adapted MapReduce to a desktop grid. MOON (MapReduce On opportunistic eNvironments) [10] is a Hadoop¹ extension that adds adaptive task and data scheduling mechanisms for an enterprise desktop grid.

The work that most closely resembles ours was presented in [11], and, as MOON, introduces MapReduce to desktop grids. XtremWeb has been used in much smaller scale than BOINC, however, and its typical use scenario consists of a federation of research labs.

6 Conclusion

We have presented BOINC-MR, a working prototype that allows MapReduce applications to run on top of a VC system, BOINC. Our results have shown that

¹ Apache Hadoop. <http://hadoop.apache.org/>

we can have a significant improvement in both performance and server bandwidth efficiency if we tailor this paradigm to our wide area environment. We have shown that BOINC-MR takes less than half the time (46%) needed by the unmodified BOINC to complete a word count MapReduce job. Furthermore, using BOINC-MR clients can cut bandwidth consumption in half on the server side, by successfully leveraging client's resources. We have also detected an excessive impact of slower nodes on application turnaround, when clients with limited bandwidth obtain a large number of tasks.

References

1. Anderson, D.P.: Boinc status report. In: The 7th BOINC Workshop (2011)
2. Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID 2004, pp. 4–10. IEEE Computer Society, Washington, DC (2004)
3. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygensky, O.: Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Gener. Comput. Syst.* 21, 417–437 (2005)
4. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.* 33, 3–12 (2003)
5. Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., Mowbray, M.: Labs of the world, unite!!? *Journal of Grid Computing* 4, 225–246 (2006)
6. Costa, F., Kelley, I., Silva, L., Fedak, G.: Optimizing data distribution in desktop grid platforms. *Parallel Processing Letters (PPL)* 18(3), 391–410 (2008)
7. Cunsolo, V.D., Distefano, S., Puliafito, A., Scarpa, M.: Volunteer computing and desktop cloud: The Cloud@Home paradigm. In: Eighth IEEE International Symposium on Network Computing and Applications, pp. 134–139. IEEE (July 2009)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. *Commun. ACM* 51, 107–113 (2008)
9. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.P.: Cost-benefit analysis of cloud computing versus desktop grids. In: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, pp. 1–12. IEEE Computer Society, Washington, DC (2009)
10. Lin, H., Ma, X., Archuleta, J., Feng, W.C., Gardner, M., Zhang, Z.: Moon: Mapreduce on opportunistic environments. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, pp. 95–106. ACM, New York (2010)
11. Tang, B., Moca, M., Chevalier, S., He, H., Fedak, G.: Towards mapreduce for desktop grid computing. In: Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2010, pp. 193–200. IEEE Computer Society, Washington, DC (2010)

Parallel Processing for Business Artifacts with Declarative Lifecycles

Yutian Sun^{1,2,*}, Richard Hull^{2,*}, and Roman Vaculín²

¹ Department of Computer Science, UC Santa Barbara, USA

² IBM T J Watson Research Center, USA

Abstract. The business artifact (a.k.a. business entity) approach to modeling and implementing business operations and processes is based on a holistic marriage of data and process and enables a factoring of business operations based on key business-relevant conceptual entities. The recently introduced Guard-Stage-Milestone (GSM) artifact meta-model provides a hierarchical and declarative basis for specifying artifact lifecycles, and is substantially influencing OMG's emerging Case Management Modeling Notation standard. In previous papers one characterization of the operational semantics for GSM is based on the incremental, strictly serial firing of Event-Condition-Action (ECA) like rules. This paper develops a parallel algorithm equivalent to the sequential one in terms of externally observable characteristics. Optimizations and analysis for the parallel algorithm are discussed. This paper also introduces a simplification of the GSM meta-model that provides more flexibility and makes checking for well-formedness of GSM models simpler and more intuitive than in the preceding works on GSM.

1 Introduction

Business artifacts (a.k.a. business entities with lifecycles) are emerging as an important conceptual basis for modeling and implementing business processes and operations [7,5]. Unlike process-centric approaches, business artifacts enable a holistic marriage of the data- and process-centric perspectives, and permit a factoring of a scope of business that is often robust in the face of changes in the underlying business. A declarative approach to business artifacts, called Guard-Stage-Milestone (GSM) was recently introduced [2,5], and is substantially influencing OMG's emerging Case Management Modeling Notation [1]. Citations [2,5] introduce the operational semantics for GSM and provide three equivalent formulations for it. One of these, which enables a direct implementation, is based on the incremental, serial firing of Event-Condition-Action (ECA) like rules, with each incoming event processed in strict sequence. This paper develops an optimized parallel algorithm for executing GSM processes with improved throughput and response time, and we also introduce a simplified GSM meta-model.

A GSM model typically consists of several artifact types, where each artifact type corresponds to a class of key business-relevant conceptual entities that progress through the business. Each artifact type includes an *information model*, which holds all business-relevant information about an artifact instance as it progresses, and a *lifecycle model*, which represents the ways that the artifact instance might progress. In GSM, *milestones* correspond to business-relevant operational objectives that an artifact instance might achieve, *stages* correspond to meaningful clusters of activity that are intended to

* This author supported in part by NSF grant IIS-0812578.

achieve milestones, and *guards* control when stages can be opened for execution. The stages may be nested, and may be running in parallel. The processing is controlled by declarative expressions, called *sentries*. Each guard is a sentry, and sentries are used to control when stages should open or close and when milestones get achieved or invalidated. The use of sentries provides a declarative basis for GSM, and nesting of stages provides abstraction and modularity. For ease of exposition, in the current paper we focus on the restricted case where there is only one artifact type and one artifact instance. The results presented here naturally generalize to the multi-type/multi-instance context.

The most straightforward approach to operational semantics of GSM, called incremental semantics, is based on the incremental, strictly sequential firing of Event-Condition-Action (ECA) rules. In response to a single incoming event, the GSM system will fire all relevant rules (e.g., for opening/closing of stages or achieving/invalidating of milestones) until no more can be fired; after which the next incoming event can be processed. In practical projects using GSM [9] when many incoming events are occurring in a short time the incremental semantics may create a bottleneck in the system. Such situations often occur in the context of collaborative problem solving [4] which can be naturally well supported by GSM. A typical pattern in collaborative problem solving involves a “shared artifact” [3] which serves as a coordination point of possibly many users and other processes. Shared artifacts are targets of possibly many concurrently incoming events and optimized implementations are needed to avoid bottlenecks.

The major contribution of the present paper is a parallel algorithm for the execution of GSM models. In particular, we (a) use a graph analysis to “target” the set of sentries that might need to be tested, (b) introduce a pipelining technique so that the impact of multiple incoming events on an artifact instance can be processed in parallel, and (c) enable parallel execution of computation steps that stem from a single incoming event. We show that the developed parallel algorithm is equivalent with the sequential algorithm. A second contribution is to introduce a simplified GSM meta-model that is more streamlined than the meta-model of [5,2]. In the simplified model, milestones are by default separated from stages, and the definition of well-formedness is simplified.

Section 2 provides a motivating example. The formal GSM meta-model and the operational semantics are presented in Section 3. Section 4 focuses on an optimized sequential algorithm and a parallel algorithm respectively. Section 5 reviews related work, and Section 6 concludes the paper. Due to space limitations the presentation here is terse; additional details and proofs of correctness are in the full paper [8].

2 A Motivating Example

This section presents an example illustrating the key features of the GSM meta-model and the importance of developing a parallel algorithm for executing GSM processes.

The running example is taken from the domain of *IT Proposal Creation*. The proposal preparation work is highly collaborative, with each proposal involving 10’s of contributing participants and it tends to be bursty, with a flurry of activity in the days leading up to the proposal release. Whenever a worker inputs information into the shared workspace for a proposal, this may trigger several automated steps (e.g., sanity checks on what was just entered, updates to schedules, launching of automated routines such as recomputing component pricing, and/or launching of other human activities). Because of the bursty nature of the work, maximizing throughput is important. Because

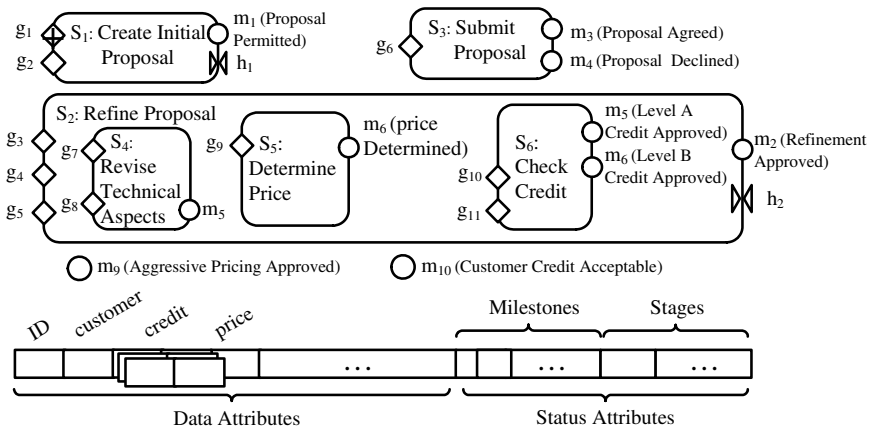


Fig. 1. Part of GSM model for Proposal Creation application

guards	milestones
g3: on +m1	m2 achiever: on refinedProposalReady()
g4: on +m4	m5 achiever: on revisedTechnicalReady()
g5: on resumeRefinement()	m5 invalidator: on +S4
g7: on +S2	m6 achiever: on priceDetermined(price)
g8: on offerManagerRequest()	m6 invalidator: on +S6
g9: on +m5	m7 achiever: on creditAppr(creditLevel) if creditLevel = "A"
g10: on +m6 and if not m7 & not m8	m8 achiever: on creditAppr(creditLevel) if creditLevel = "B"
g11: on +m6 and if price>500K & not m8	m9 achiever: on AggressivePricingExecApproval()
terminators	m9 invalidator: on AggressivePricingRescinded()
h1: on suspendCreation()	m10 achiever: if price <= 500K & (not m7 or not m8)
h2: on suspendRefinement()	m10 achiever: if price > 500K & m8
For each stage S with milestone m, each achiever for m is also a terminator for S	m10 invalidator: on +S6 if price > 500K & not m8

Fig. 2. Selected sentries for Example 2.1

a step by some worker (e.g., giving an approval or submitting some information) may lead to an immediate follow-up step by the same worker or a close collaborator, minimizing response time is also important.

Example 2.1 Figure 1 illustrates a small part of a GSM model that can support Proposal Creation, and Figure 2 shows several sentries for that model. The information model of the business artifact is shown along the bottom of Figure 1, and top layers of the lifecycle model are shown above. The three top-level phases of activity, depicted as GSM stages with rounded corner boxes, are *Create Initial Proposal* (S1), *Refine Proposal* (S2), and *Submit Proposal* (S3). Stage *Refine Proposal* is typically executed several times. Stages can be arranged hierarchically, as illustrated in *Refine Proposal*. (Although not illustrated in this example, distinct stages may execute in parallel.)

GSM milestones are shown as circles. Several milestones are naturally associated with the completion of stages, e.g., *Refine Proposal* ends when an executive approves the refinement. Other milestones are free-standing. For example, *Customer Credit*

Acceptable will become true if the proposal price is \leq \$500K and the client has a Level A credit rating, and if the proposal price is $>$ \$500K and the client has a Level B credit rating. Milestones can be tested as Boolean values by conditions anywhere in the GSM model.

Guards, which control when stages can be opened for execution, are shown as diamonds. Guard g_1 on *Create Initial Proposal* is a “bootstrapping” guard; when it is triggered a new artifact instance is created. In some cases it is convenient to have a *terminator* for a stage, shown using a bowtie. For example h_2 on *Refine Proposal* is a terminator that can be triggered if a worker indicates that this stage should be suspended (e.g., because the client has stated that they are withdrawing their request). Unlike milestones, guards and terminators cannot be tested as Booleans. ■

We now briefly describe a representative scenario where fast response is important.

Example 2.2 Three representative substages are shown in *Refine Proposal*, namely *Revise Technical Aspects*, *Determine Price*, and *Check Credit*. In practice, *Revise Technical Aspects* would have a number of substages dealing with hardware, software, workforce, logistics, etc. *Determine Price* might involve several substages, and for this example we assume that they are all automated (e.g., for determining costs associated with various portions of the technical aspects, for computing shipping costs and taxes, for adding everything up). Only a subset of these substages would be relevant for a given execution of *Determine Price*. There may be other stages analogous to *Determine Price*, e.g., for determining risk, impact on branding, or impact on competitive positioning. The *Check Credit* stage might be triggered if a newly computed price is higher than the credit level already approved for the customer.

After a management-level worker approves a changed part of the *Revise Technical Aspects* stage, there will be processing to recompute the price, which will then lead to a feedback that either confirms that the client credit is still acceptable, or indicates that a higher-level Credit Check must be performed. It is desirable that this feedback be given to the worker within just a couple of seconds. ■

3 Guard-Stage-Milestone Meta-model

This section presents the syntax and operational semantics of the GSM meta-model. The focus is one GSM model that involves a single artifact type. Due to space limitations, the presentation is largely informal; formal definitions may be found in [8].

As illustrated in Section 2, a *Guard-Stage-Milestone model* (or *GSM model*) includes both an information model and a lifecycle model. This is typically denoted as a 5-tuple $\Gamma = (Att, EType, Stg, Mst, Lcyc)$, where the components are, respectively, the set of *attributes* (partitioned into the set Att_{data} of *data attributes* and Att_{status} of *status attributes*; the set of (*incoming*) *event types*; the set of *stages*; the set of *milestones*; and the *lifecycle model* (which is defined below).

The *domain* of each data attribute A , denoted $Dom(A)$, is assumed to include the undefined value \perp . For each milestone m , there is a Boolean *milestone status* attribute also denoted by m in Att_{status} ; this is *true* if the milestone has been achieved and not since invalidated, and *false* otherwise. For each stage S there is a Boolean *stage status* attribute also denoted by S ; this is *true* if the stage is open and *false* otherwise.

Artifact instances of GSM model Γ interact with the external environment by sending and receiving *typed external events* with event types defined in $EType$. There are two

types of external events: *incoming events* that are received from the external environment and *outgoing events* that are sent to the external environment. An *event instance* (or simply, *event*) consists in an event type and a *payload*, specified as a family of attribute/value pairs. As a notational convenience, the attributes here are drawn from the set of data attributes of Γ . When an artifact instance incorporates an incoming event, the attributes mentioned in the payload are updated according to the values

The actual “work” of a GSM model is performed by *tasks*, which are contained in atomic stages. Tasks are invoked through message sending. The two types of task relevant for this paper can: (a) generate 1-way messages (when invoked they wait for a “handshake” indicating success or failure); (b) generate 2-way service calls (when invoked they wait for the service call return from the called service or a time-out message). (Upon completion of a 2-way service call generated by a task, an incoming message is received by the artifact instance, and the atomic stage associated with the task is closed.) In GSM, computational tasks (including assignments of attribute values) are modeled using 2-way service calls.

A *sentry* for GSM model Γ is an expression of form ‘**on** \langle event expression \rangle **if** \langle condition \rangle ’; or ‘**on** \langle event expression \rangle ’; or ‘**if** \langle condition \rangle ’. The event expression may have one of the following forms: *Incoming event expression*: E , where $E \in EType$ (intuitively, it gets satisfied if an event of type E occurs); *Internal event expression*: For each milestone m this includes $+m$ and $-m$; and for each stage S this includes $+S$ and $-S$. Intuitively, $+m$ is triggered when m is achieved, $-m$ when m is invalidated, $+S$ when S is opened, and $-S$ when S is closed. As illustrated in Section 2, the guards and terminators associated with stages are sentries. Also, milestones have associated achieving sentries and invalidating sentries.

The *lifecycle model* of a GSM model $\Gamma = (Att, EType, Stg, Mst, Lcyc)$, is the tuple $Lcyc = (Substages, Task, Submilestones, Guards, Terminators, Ach, Inv)$ where the components are, respectively a function that maps each stage to its family of substages (where the substage relationship forms a forest); a function that maps each atomic stage to the unique task that it contains; a function that maps a subset of the milestones to stages that they are contained in; a function that associates guards to stages; a function that associates terminators to stages; a function that associates achieving sentries to milestones; and a function that associates invalidating sentries to milestones.

A *snapshot* of a GSM model Γ is an assignment Σ that maps each attribute of Γ to a value in its domain (which includes \perp for data attributes). Snapshots are required to satisfy the *GSM invariant*, namely that if a stage S is closed (i.e., if status attribute S is assigned the value false) then all of its child stages are also closed. In the following we also need the notion of *pre-snapshot*; this is an assignment Σ for Γ that might not satisfy the GSM invariant.

Citation [2] introduces three equivalent formulations of the operational semantics of GSM, called *incremental*, *fixpoint*, and *closed-form* (which is expressed in first-order logic); in the current paper we focus exclusively on the incremental formulation. This is based on responding to an incoming external event by repeated application of Event-Condition-Action (ECA) like rules until no further rules can be fired. The ECA-like rules are formed from the sentries of the GSM model. Because negation is present in the ECA-like rules, some restrictions are placed on GSM models, and on the order of rule application. These restrictions, described below, are analogous to those found in stratified datalog and logic programming.

A fundamental notion in GSM is that of *Business Step* (*B-step*). These are conceptual atomic units of business-relevant processing, and correspond to the effect of incorporating one incoming event into a snapshot of GSM model Γ . B-steps have the form of 4-tuples $(\Sigma, e, \Sigma', Gen)$, where Σ, Σ' are snapshots, e is an incoming external event, and Gen is a set of outgoing external events. Under the incremental formulation, this 4-tuple is a B-step if there is a sequence of pre-snapshots $\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(e, \Sigma), \Sigma_2, \dots, \Sigma_n = \Sigma'$ where, speaking intuitively, Σ_1 corresponds to the direct incorporation (called “immediate effect”, denoted as *ImmEffect*) of event e into Σ , and Σ_{i+1} corresponds to the application of a sentry to Σ_i for $i \in [2..n]$. (For simplicity of exposition below, we also permit Σ_{i+1} to be identical to Σ_i for some i .) Further, Gen is the set of outgoing events caused by the tasks that are launched during the sequence. In the formal model, computation of this sequence of pre-snapshots is assumed to happen in a single instant of time, and the set Gen of events is transmitted to the external environment in one batch immediately after Σ' is computed. We sometimes write a B-step as a triple (Σ, e, Σ') if Gen is understood from the context.

Example 3.1 In Fig. 1, suppose now S_1, S_3, S_5 and S_6 are closed and S_2 and S_4 are open. A new B-step can be triggered by receiving event *revisedTechnicalReady*. In this B-step, m_5 will be achieved and so stage S_4 will close. Since g_9 is $+m_5$, the B-step will also open stage S_5 . After that no further sentries are applicable and the B-step ends. ■

B-steps must satisfy two properties, called *Inertial* and *Toggle-Once*. Speaking intuitively, inertial means that if Σ and Σ' differ on some status attribute, then there must be some sentry that justifies the change. Toggle-Once means that in a sequence as given above, a status attribute can change value at most once. This corresponds to the intuition that everything business-relevant about a B-step should be observable by looking at the snapshots before and after the B-step.

We now describe how the sentries of Γ are used to create the ECA-like rules, called here ‘Prerequisite-Antecedent-Consequent’ (PAC) rules. The “prerequisite” component of these rules helps to enforce the Toggle Once property. (In most naturally arising GSM schemas, the “prerequisite” component is not needed.)

Definition: A *Prerequisite-Antecedent-Consequent* (PAC) rule ρ , for GSM model Γ is a tuple (π, α, γ) , where: (Prerequisite) π is a formula on attributes in Att ; (Antecedent) α is a sentry based on attributes in Att , internal events over Att_{status} , and external event types $EType$; and (Consequent) γ is an internal event $\odot\sigma$, where $\odot \in \{+, -\}$ and $\sigma \in Att_{status}$.

Returning to the incremental semantics, suppose that a partial sequence $\Sigma = \Sigma_0, \Sigma_1 = ImmEffect(e, \Sigma), \Sigma_2, \dots, \Sigma_i$ has already been constructed. A PAC rule (π, α, γ) is *applicable* to (Σ, Σ_i) if $\Sigma \models \pi$ and $\Sigma_i \models \alpha$. In this case, Σ_{i+1} may be formed by modifying Σ_i according to the status change called for in γ .

A GSM model Γ is *well-formed* if the graph defined next is acyclic. This graph is also central to the parallelization developed in Section 4.

Definition: The *extended polarized dependency graph* (EPDG) of GSM model Γ , denoted $EPDG(\Gamma)$ is a graph where the node set \mathcal{V} includes: for each event type E , nodes $+E$; for each milestone m , nodes $+m$ and $-m$; and for each stage S , nodes $+S$ and $-S$. The edge set \mathcal{E} is defined as follows. (Here “ \odot, \odot' ” are *polarities*, and range over $\{+, -\}$; σ, σ' are not necessarily distinct status attributes; and (π, α, γ) as a PAC rule in Γ_{PAC} .)

	Basis	Prerequisite	Antecedent	Consequent
Explicit rules				
PAC-1	Guard: if on ξ if φ is a guard of S . (Include term S' if S' is parent of S .)	$\neg S$	on ξ if $\varphi \wedge S'$	$+S$
PAC-2	Terminator: if on ξ if φ is a terminator of S .	S	on ξ if φ	$-S$
PAC-3	Milestone achiever: if m is a milestone and on ξ if φ is an achieving sentry for m . (Include term S' if S' is parent of m .)	$\neg m$	on ξ if $\varphi \wedge S'$	$+m$
PAC-4	Milestone invalidator: if m is a milestone and on ξ if φ is an invalidating sentry for m .	m	on ξ if φ	$-m$
Invariant preserving rule				
PAC-5	If S is a child stage of S'	S	on $\neg S'$	$-S$

Fig. 3. Prerequisite-Antecedent-Consequent Rules of a GSM Model

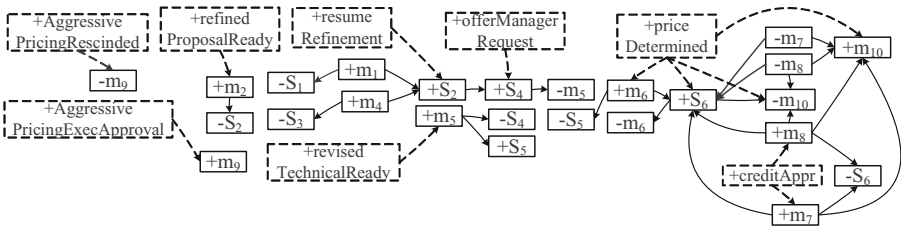


Fig. 4. (Extended) Polarized Dependency Graph

- If α includes in its triggering event ξ the internal event expression $\odot'\sigma'$, and γ is $\odot\sigma$, then include edge $(\odot'\sigma', \odot\sigma)$.
- If α includes in its condition the expression σ' and γ is $\odot\sigma$, then include edges $(+\sigma', \odot\sigma)$ and $(-\sigma', \odot\sigma)$.
- If a guard g (or a terminator h) of a stage S contains a triggering event of type E , or a data attribute from the payload of E , then include edge $(+E, +S)$ (or $(+E, -S)$).
- If an achieving (or invalidating) sentry of milestone m contains a triggering event of type E , or a data attribute from the payload of E , then include edge $(+E, +m)$ (or $(+E, -m)$).

Intuitively, an edge $(\odot\sigma, \odot'\sigma')$ is included in the EPDG if possible impacts on $\odot\sigma$ should be finalized before the PAC rules impacting $\odot'\sigma'$ are considered. We may also use the notion of “PDG” [8] (EPDG without $+E$ nodes, where E is an event type) in the remainder of this paper.

Fig. 4 shows to part of the EPDG for the GSM model of Example 2.1 (the solid nodes and edges form the “PDG”). Here, there is an edge from $+m_5$ to $+S_5$, because the guard g_9 on S_5 includes $+m_5$. Also, both $+m_8$ and $-m_8$ point to m_{10} in the EPDG, because m_8 is mentioned in the **if** part of an achiever of m_{10} .

When constructing a B-step using the incremental construction, the PAC rules are considered in an order based on a topological sort of the PDG. A key result of [2] states that if Γ is well-formed and e is applicable to Σ , then there is exactly one snapshot Σ' (and one set Gen) such that $(\Sigma, e, \Sigma', Gen)$ is a B-step. In particular, the construction of Σ' is independent of the topological sort used for rule application.

As discussed in [8], in some corner cases after completion of a B-step $(\Sigma, e, \Sigma', Gen)$ some PAC rule might be applicable to Σ' . In this paper we consider only GSM models Γ for which this does not arise: such GSM models are called “*orphan-free*”.

4 Parallelized Business Steps: Algorithm with Parallelism

The reason to propose the parallel algorithm for GSM is to solve the bottleneck that may occur when events are evaluated in a strictly sequential manner. The proposed parallel algorithm uses the following techniques. (1) *Targeting*: When processing a single B-step, it is sufficient to traverse only a subset of the PDG graph instead of the entire graph (introduced in this section); (2) *Pipelining*: B-steps can be evaluated in parallel. (3) *Parallelism within a single B-step*: Even within a B-step, some evaluations can be done in parallel.

We start by introducing the targeting algorithm which serves as a basis for the parallel algorithm. This approach reduces the number of PDG nodes that need to be evaluated in every B-step. We show the equivalence between the targeting algorithm and the sequential one.

Assuming an orphan-free and well-formed GSM model, the following “*targeting algorithm*” performs one B-step for an incoming event e of type E :

- For each reachable node v from $(+E)$ in topological order
 - If v is $+E$, then apply immediate effect with e .
 - Otherwise, apply each PAC rule associated with v .

Theorem 4.1 (Informal) The targeting algorithm and the incremental algorithm share the same operational semantics.

The proof of Theorem 4.1 is in [8]. If the multi-threaded mechanism is used in the GSM engine, one optimization using the targeting algorithm can be obtained by evaluating two nodes that have no partial relationship in parallel. We use this idea in the parallel algorithm to handle multiple events.

Example 4.2 In Fig. 4, if event “priceDetermined” comes, instead of traversing the whole PDG, only nodes $+m_6$, $+S_6$, $-S_5$, $-m_6$, $+m_{10}$, and $-m_{10}$ need to be visited in topological order. Furthermore, since $-S_5$, $+m_{10}$, and $+S_6$ share no partial relationship, these three nodes can be evaluated in parallel. ■

The targeting algorithm provides the basis for parallelizing B-steps. Suppose two events of different types arrive at about the same time and the EPDG subgraphs of the two event types have no intersection in terms of nodes. Naturally, it is possible to traverse the two subgraphs and apply the corresponding PAC rules in parallel. On the other hand, if there are some nodes shared by the two subgraphs, the parallel algorithm needs to make sure that PAC rules associated with the shared nodes for the event that arrived earlier are applied before those associated with the later event. We assume that incoming events and the corresponding B-steps are labeled by increasing numbers (or logical timestamps) that correspond to the order in which events occurred. The parallel algorithm will use these numbers to make sure that during the concurrent B-steps, the ordering of events is not violated.

In the algorithms, we use the following notation. Given a GSM model Γ and its EPDG $(\mathcal{V}, \mathcal{E})$, each node $v \in \mathcal{V}$ is associated with two sets denoted as $will_visit(v)$ and $has_visited(v)$, whose elements are positive integers. Given a status attribute σ , the *complement* of node $+\sigma$ (or $-\sigma$) is node $-\sigma$ (or $+\sigma$), denoted as $co(-\sigma)$ (or $co(+\sigma)$).

For each incoming event, there are three high-level steps of how B-steps are processed in parallel: *labeling*, *evaluation*, and *removal*.

1. Labeling: In order to understand which node is available for immediate processing, it is necessary to label them first. Suppose the k^{th} event arrives which is of type E , then add k to $will_visit(v)$ of each EPDG node v that is reachable from $+E$. This “*labeling policy*” is based on the result of the targeting algorithm.

2. Evaluation: Once the numbers are labeled for the k^{th} event, the B-step k can start to evaluate its corresponding reachable nodes similar to targeting algorithm (by applying the PAC rules or the immediate effect). A B-step k can evaluate a node v , if v satisfies the following *evaluation policy*: (1) k is the smallest number in $will_visit(v)$; and (2) k is smaller than each number in $will_visit(u)$ and $will_visit(co(u))$, where u is a node and (u, v) is an edge; and (3) k is no greater than each number in $will_visit(co(v))$.

3. Removal: Once B-step k applies the PAC rules or the immediate effect on a node v , k is moved from $will_visit(v)$ to $has_visited(v)$ which indicates that the B-step k has visited v already. This removes the ordering restriction and will unblock the later B-steps, so that they can apply PAC rules on the shared nodes.

Theorem 4.3 (Equivalence theorem, informal) .The parallel algorithm and the targeting algorithm are equivalent in terms of the observable behavior.

The formal statement and the proof of Theorem 4.3 is in [8]. Theorem 4.3 guarantees that the parallel algorithm can correctly handle multiple incoming events in parallel and generate the same snapshots, outgoing events, and query answers as if all incoming events and queries are processed sequentially.

In [8], an optimized algorithm is provided. Based on the optimized version, it can be shown that the complexity in the worst case is no worse than the targeting algorithm; while much better than targeting algorithm in terms of order in the best case.

5 Related Works

The GSM paradigm used here is based on the business artifact model originally introduced in [7,6], but using a declarative basis [2,5].

There is a strong relationship between the GSM model and Case Management [13,12]; both approaches focus on conceptual entities that evolve over time, and support *ad hoc* styles of managing activities. The GSM framework provides a formal operational semantics [2,5]. The core GSM constructs are being incorporated into the emerging OMG Case Management Modeling Notation standard [1], and there is ongoing work to adapt the GSM semantics to that context.

DecSerFlow [11] is inherently more declarative than GSM. GSM can be viewed as a reactive system that permits the use of a rich rules-based paradigm for determining, at any moment in time, what activities should be performed next.

There is a loose correspondence between the artifact approach and proclerts [10]; both approaches factor a BPM application into “bite-size” pieces that interact through time.

Procllets do not emphasize the data aspect, and support only message-based procllet interaction. In addition to supporting messages, GSM permits interaction of artifact instances through condition testing and internal event triggering.

Citation [3] presents a framework for supporting web-based collaborative business processes. They use a construct called *task artifact* to hold the complete collaboration state, and to help manage the response to incoming events. An interesting research question is to explore whether GSM could provide a useful approach for specifying possible lifecycles of task artifacts.

6 Conclusions

Business artifacts with Guard-Stage-Milestone (GSM) lifecycle models offer a flexible, declarative, and modular way to support collaborative business processes. The core constructs of GSM are being incorporated into the emerging OMG Case Management Modeling Notation standard. This paper presents an algorithm to support parallel execution of GSM processes, which can be especially useful in the context of highly collaborative and/or web-scale business processes. The paper also introduces a GSM meta-model that simplifies the previously published one.

An important next step is to implement and benchmark the parallel algorithm presented here, on both synthetic and “real” processes, to determine the practical gains in throughput and response time yielded. More broadly, it will be useful to examine existing and future application areas for GSM and Case Management to identify other ways to optimize overall performance.

References

1. BizAgi, Cordys, IBM, Oracle, SAP AG, Singularity (OMG Submitters) and Agile Enterprise Design, Stiftelsen SINTEF, TIBCO, Trisotech (Co-Authors). Proposal for: Case Management Modeling and Notation (CMMN) Specification 1.0, Document bmi/12-02-09, Object Management Group (February 2012)
2. 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)
3. Dorn, C., Taylor, R.N., Dustdar, S.: Flexible social workflows: Collaborations as human architecture. *IEEE Internet Computing* 16(2), 72–77 (2012)
4. Dustdar, S.: Caramba - a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases* 15(1), 45–66 (2004)
5. Hull, R., et al.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: Proc. 5th ACM Intl. Conf. on Distributed Event-based Systems, DEBS 2011, pp. 51–62. ACM, New York (2011)
6. Kumaran, S., Nandi, P., Heath, T., Bhaskaran, K., Das, R.: Adoc-oriented programming. In: SAINT, pp. 334–343 (2003)
7. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Syst. J.* 42, 428–445 (2003)
8. Sun, Y., Hull, R., Vaculín, R.: Parallel processing for business artifacts with declarative lifecycles (full version). IBM internal technical report, available on request (2012)
9. Vaculín, R., et al.: Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: Proc. Intl. Conf. on Enterprise Distributed Objects Conference (EDOC), pp. 151–160 (2011)

10. van der Aalst, W.M.P., et al.: Procllets: A framework for lightweight interacting workflow processes. *Int. J. Cooperative Inf. Syst.*, 443–481 (2001)
11. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: *The Role of Business Processes in Service Oriented Architectures 2006* (2006)
12. van der Aalst, W.M.P., Weske, M.: Case handling: a new paradigm for business process support. *Data Knowl. Eng.* 53, 129–162 (2005)
13. Zhu, W.-D., et al.: Advanced Case Management with IBM Case Manager,
<http://www.redbooks.ibm.com/redpieces/abstracts/sg247929.html?Open>

Automatically Generating and Updating User Interface Components in Process-Aware Information Systems

Jens Kolb, Paul Hübner, and Manfred Reichert

Institute of Databases and Information Systems
Ulm University, Germany
{jens.kolb,paul.huebner,manfred.reichert}@uni-ulm.de
<http://www.uni-ulm.de/dbis>

Abstract. The increasing adoption of process-aware information systems (PAISs) has resulted in a large number of implemented business processes. To react on changing needs, companies need to be able to quickly adapt these process implementations. Current PAISs only provide mechanisms to evolve the schema of a process, but do not allow for support the automated creation and adaptation of user interfaces (UIs). The latter may have a complex logic and comprise conditional elements or database queries. Creating and evolving UIs manually is a tedious and error-prone task. This paper introduces a set of patterns for transforming fragments of a business process, whose activities are performed by the same user role, to UIs of the PAIS. In particular, UI logic can be expressed using the same notation as for process modeling. Furthermore, a transformation method is introduced, which applies these patterns to automatically derive UIs by establishing a bidirectional mapping between process model and UI. This mapping allows propagating UI changes to the process model and vice versa. Overall, our approach enables process designers to rapidly develop and update complex UIs in PAISs.

1 Introduction

Process-aware information systems (PAISs) separate process execution from application code. Hence, a separation of concerns is realized based on explicit *process models*. When initially capturing business processes in process models, focus is put on business aspects, while technical aspects concerning process execution are excluded. Usually, respective process models cover the users' activities at a fine-grained level (cf. Fig. 1a). Hence, before deploying such a process model in a PAIS, it must be revised and customized. For example, several human tasks, forming a process fragment in the process model, may be combined into one activity in the executable process model (cf. Fig. 1b). This activity is then implemented by a *user interface (UI) component* in the PAIS, e.g., a user form whose logic corresponds to the one of the initial process fragment (cf. Fig. 1c). Based on this logic, for example, form elements may be disabled when selecting a certain check box, or web services may be called in the background. Overall,

both the implementation and maintenance of the UI components in a PAIS is a cumbersome and costly task. This hinders quick adaptations of process implementations [1].

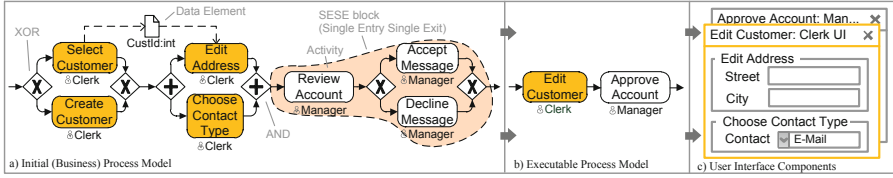


Fig. 1. Deriving UI Components from a Business Process Model

The process evolution of the processes implemented in a PAIS is a critical success factor for any company. Such an evolution requires changes of the process models and their associated UI components. Process model evolution is a well-understood feature in modern PAISs [2,3]. There exist editors for defining simple UI components of the PAIS (e.g., moving or renaming input fields in a UI). However, complex changes of the logic of UIs can not be done by users, but require process implementers. Moreover, the automatic propagation of changes made in the UI components to the process model and vice versa is not supported. We address these issues through the automatic generation of UI components out of process fragments, and present patterns for transforming process fragments to UI components. While *elementary transformation patterns (ETP)* transform single activities to simple UI elements, *complex transformation patterns (CTP)* enable the mapping of entire process fragments and their logic to UI components, showing the same behaviour as the process fragment. Next, we provide an advanced transformation method that allows generating UI components out of a process model based on the user roles assigned to activities. This method allows propagating changes of UI components to the process model and vice versa. Our transformation method decreases the effort for evolving PAIS to changing needs. The paper is structured as follows: Section 2 introduces basic notions. Section 3 describes common patterns for transforming process model fragments to UI components. Section 4 presents a method for transforming process models to UI components, which is based on transformation patterns and role-based process views. Section 5 discusses related work and Section 6 summarizes the paper.

2 Basic Notions

A process model is described in terms of a directed graph whose node set comprises *activities*, *gateways*, and *data elements*. An activity either corresponds to a *human task* and thus requires user interactions, or to a *service* representing an *automated task*. In turn, gateways can be categorized into *AND*, *XOR* and *Loop* and are used for modeling parallel/conditional branchings and loops. Edges between activities and/or gateways represent precedence relations, i.e., the *control*

flow of the process model (cf. Fig. 1a). Furthermore, *data elements* comprise *primitive* data elements and *complex* ones. Primitive data elements cover elementary data values of the process model and have one of the following types: *integer*, *float*, *boolean*, *string*, *date*, or *URI*. Based on this, the data flow is defined by a set of directed edges connecting data elements and activities. *Writing* a data element is expressed through an edge pointing from an activity to the data element. In turn, *reading* a data element is expressed by an edge from this data element to the activity. We presume that process models are *well-structured* [4], i.e., sequences, branchings (of different semantics), and loops are specified as blocks with well-defined start and end nodes having the same gateway type. These blocks, also known as *SESE* (*single-entry-single-exit*) blocks (cf. Fig. 1a), may be nested, but are not allowed to overlap.

3 User Interface Transformation Patterns

The goal of our research is to *identify and apply a general set of UI transformation patterns to map process fragments to UI components*. To achieve this goal, we first describe a three-step method, which we apply for identifying UI transformation patterns (cf. Fig. 2). *Step 1* analyzes and evaluates PAIS projects in which we were involved. More precisely, we analyze the process models from these projects as well as their technical implementation. *Step 2* analyzes existing PAISs and their support for UI generation. *Step 3* scans related literature. The empirical results are used to specify general UI transformation patterns. Based on these patterns, *Step 4* develops a transformation method for the automatic generation of role-specific UI components (cf. Section 4).

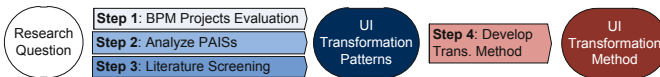



Fig. 2. Transformation Pattern Identification Method

3.1 Elementary Transformation Patterns

Elementary transformation patterns (ETP) enable the transformation of single process model elements to simple UI elements. For example, an activity may be transformed into a simple user form. Thereby, the respective ETP considers activity input/output data elements and maps them to form elements.

ETP1 (Human Activity Transformation) transforms a single activity to a *Form Group Element (FGE)* (cf. Table 1); i.e., for each human activity of a process model, an FGE is generated. In modern PAIS, usually, such an FGE is represented by a dialog window.

Table 1. ETP1: Human Activity Transformation

ETP1: Human Activity Transformation	
Description:	A <i>human activity</i> of a business process model (i.e., an activity to be performed by a human resource) is transformed into a <i>Form Group Element (FGE)</i> . An FGE corresponds to a UI element that contains UI elements for displaying or editing data.
Example:	<p>A clerk must perform an activity, in which customer data is edited.</p> 
Problem:	To perform a human activity within a PAIS, a user interaction is required.
Implementation:	An FGE can be implemented in terms of a dialog window. In the context of CTPs, an FGE constitutes a grouping element of the UI.

ETP2 (Service Activity Transformation)¹ creates application stubs for automated tasks not performed by a user (e.g., fetching data from a database). ETP2 is needed to generate complete UI components enabling interactions with both users and backend systems (cf. Section 3.2).

ETP3 (Data Flow Transformation)¹ transforms data elements and data flow edges to UI elements. *ETP3* as well as related patterns *ETP3.1-ETP3.3* generate *Field Elements (FE)* within an FGE; i.e., when generating the FGE (cf. ETP1), the data elements and edges of a process activity are transformed to input/output FEs of a UI component. In this context, sub-patterns *ETP3.1* and *ETP3.2* are applied to indicate whether the FE is read-only or editable. Finally, *ETP3.3* transforms the type of a data element to a specific FE; e.g., a boolean data element is transformed to a radio button element with two choices.

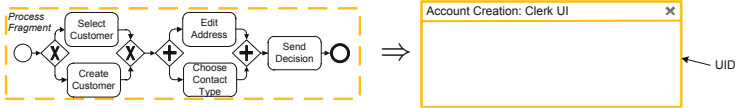
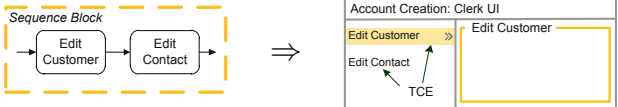
3.2 Complex Transformation Patterns

Complex Transformation Patterns (CTPs) allow transforming entire fragments of a process model, whose activities shall be performed by the same user, to UI components. When creating such a UI component both the control and data flow of the process fragment are considered. Hence, each generated UI component covers parts of the overall process logic. By combining *role-specific* activities in the same UI component, unnecessary UI context switches can be avoided. To structure such a UI component, tab elements—called *Tab Container Elements (TCE)*—are used. A CTP interconnects TCEs according to the control flow of the process fragment to which it is applied. Single activities and data elements related to the process fragment are transformed using ETPs.

CTP1 (Process Model Transformation). generates a *User Interface Dialog (UID)* for process fragments whose activities are processed by the same user role (cf. Table 2). A UID is a toplevel container, which is represented by a dialog window in the PAIS containing UI elements representing activities and data elements.

¹ A more detailed description of all ETPs can be found in [5].

Table 2. Pattern Descriptions for Patterns CTP1 and CTP2

CTP1: Process Model Transformation	
Description:	For a particular process fragment, a surrounding <i>User Interface Dialog (UID)</i> , i.e., a toplevel container window, is generated. Following this, all other UI elements related to activities of this fragment are generated based on ETPs and CTPs, and are then embedded in the UID.
Example:	All interactions with a clerk shall be done using the same UI component. 
Problem:	The UI elements related to the activities and data elements of a particular process fragment need to be mapped to a toplevel container window. The UI flow logic (e.g., the ordering in which field elements may be displayed or written) corresponds to the control flow of the given process fragment.
Implementation:	For each process fragment, a UID element (dialog window) is generated.
CTP2: Sequence Block Transformation	
Description:	A sequence of activities (and SESE blocks) is transformed into a sequence of <i>Tab Container Elements (TCE)</i> to be processed in the same sequential order.
Example:	A clerk first edits the customer data and then the corresponding contact data. 
Problem:	Human activities, performed in sequence by the same user (role), shall be accomplished using the same UI component, instead of using separate UI components (e.g., dialog windows) for each activity.
Implementation:	For each activity (or SESE block), a TCE element is created. The order in which these TCEs are processed relates to the one of the respective activities.

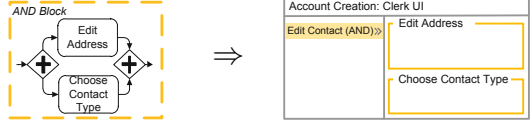

CTP2 (Sequence Block Transformation) deals with the transformation of a sequence of activities (and SESE blocks respectively) to TCEs (cf. Table 2). For each activity (or SESE block) of the sequence, a TCE element is generated and linked to other TCEs according to the given activity sequence.

CTP3 (Parallel Block Transformation) transforms parallel activities (or SESE blocks) of a process fragment to UI elements within the same UID. These elements may then be accessed concurrently (cf. Table 3). The UI component is similar to the one of a single activity (cf. ETP1). However, CTP3 not only covers the transformation of activities arranged in parallel, but enables the concurrent processing of SESE blocks arranged in parallel to the respective UI elements.

CTP4 (XOR Block Transformation)² transforms an XOR branching of a process fragment to a UI component. CTP4 generates independent TCEs for each branch of the XOR branching. The decision, which branch and hence which TCE shall be selected, is made during run-time; e.g., whether the TCE element for creating a new customer or the one for editing an existing customer shall

² A more detailed description of this pattern can be found in [5].

Table 3. Pattern Descriptions for Patterns CTP3 and CTP6

CTP3: Parallel Block Transformation	
Description:	A parallel block and its activities are mapped to a single TCE for their processing. This TCE allows for their concurrent execution.
Example:	While editing the address of a customer, the contact type the customer wants to use for communication can be entered in parallel. 
Problem:	Activities (or SESE blocks) of a process fragment, which are performed in parallel by the same user (role), shall be mapped to the same UI component; UI elements then must be displayable/editable concurrently.
Implementation:	When applying CTP3, for each parallel branch, FGEs are added to the TCE.
CTP6: Background Activity Transformation	
Description:	While human activities are performed by human resources, service activities may automatically fetch or save data concurrently in the background.
Example:	A user selects a customer name in order to edit respective customer data. After selecting the name, in the background, all available customer information is retrieved from the database and displayed to the user. 
Problem:	A service activity needs to be executed concurrently to human activities performed by the same user (role). This requires the concurrent fetching/storing of data from a database as well as the automated and dynamic displaying of new form elements.
Implementation:	Dynamic forms, which contain background activities, need a change listener mechanism to detect user inputs and to react on them.

be displayed. In particular, run-time data for deciding which branch of an XOR branching shall be executed is required.

CTP5 (Loop Block Transformation)² transforms a loop block to elements of a UI component. For each loop, CTP5 generates a TCE and corresponding UI elements for nested activities or SESE blocks (cf. CTP1). Additionally, a decision element is required to decide whether to exit the loop after completion of a particular iteration or trigger the next loop iteration either based on data elements processed during loop execution (e.g., evaluating data for validity) or external criteria (e.g., calling someone until getting an answer).

CTP6 (Background Activity Transformation) reflects the need for dynamically loading data elements by a service activity (cf. Table 3). More precisely, data has to be fetched from or stored to a backend system, while the user concurrently works on human activities.

4 Transforming Process Models to User Interfaces

Section 4.1 shows how the presented patterns are used to transform process fragments to UI components of the PAIS. Further, we discuss how process fragments can be adapted through changes of the UI components (cf. Section 4.2).

4.1 User Interface Transformation Method

To transform a process model, consisting of several process fragments, to multiple UI components, we introduce a five step method (cf. Fig. 3). Thereby, the number of generated UI components depends on the number of different user roles involved in the process.

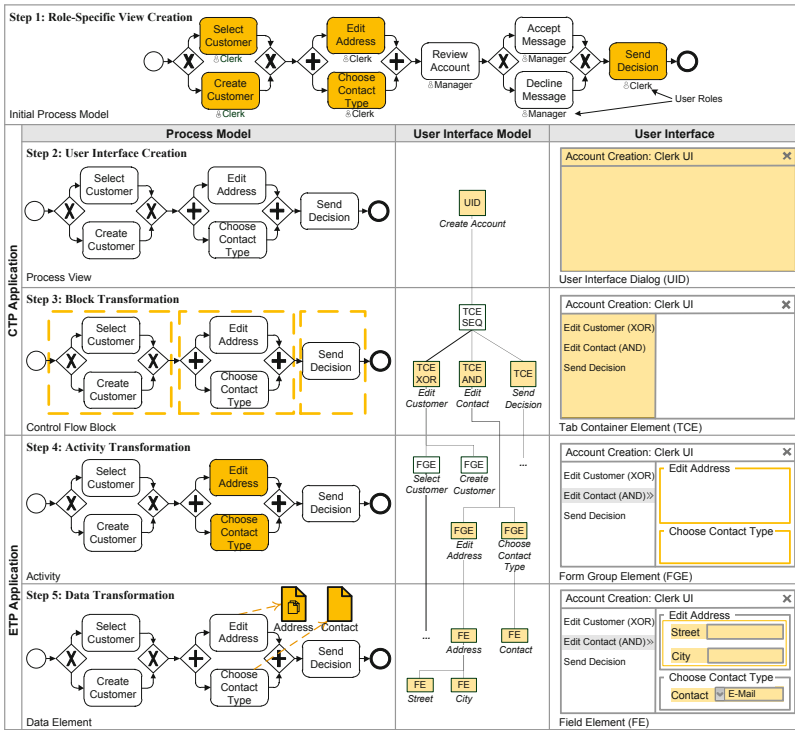


Fig. 3. Transformation Method

Step 1. *Role-specific process views* [6,7,8] are created for the given process model. A process view abstracts from certain aspects of the process model (e.g., it only contains activities of a particular user role). In our context, a role-specific process view constitutes the basis for creating a role-specific UI component.

Step 2. For each process view, a *User Interface Dialog (UID)* is created. A UID acts as a toplevel container including all UI elements required for processing the activities of a process view. For this purpose, CTP1 is applied (cf. Table 2).

Step 3. CTP2-6 are applied to transform complete process fragments to UI elements. For each CTP applied, a *Tab Container Element (TCE)* is generated. Each TCE is represented in the tab bar area (cf. Fig. 3, Step 3). When clicking such an item, the corresponding UI elements are displayed. If there are nested SESE blocks, they are displayed in a hierarchical tree in the tab bar area.

Step 4. Single activities (ETP1+2) are transformed into *Form Group Elements (FGE)*. Basically, each FGE represents one activity in the process model. In case of a parallel branching, multiple activities are displayed on a TCE element in the UI (cf. Fig. 3, Step 4).

Step 5. Data elements of the process view are transformed to *Field Elements (FE)* and positioned within an FGE. There exist different kinds of FEs depending on the data type of the respective data element (cf. pattern ETP3.3 in [5]).

The internal structure of the resulting UI component is represented through a *User Interface Model (UIM)* (cf. Fig. 4b). This tree-based schema describes the UI structure generated by our transformation method.

4.2 Synchronizing Process Model and UI Changes

After generating complex UI components for a process model through process views and deploying them in the PAIS, users may want to modify the UI. Basically, two categories of UI changes can be distinguished. *Local changes* are changes not affecting the associated process view. For example, assume that a user re-positions the FGE *Edit Address* within the TCE *Edit Contact (AND)* in Fig. 4a. Such a change would not affect the execution order of the activities in the process view, i.e., it only affects the visual representation of the UI component. Hence, the change needs not be propagated to the view. *Global changes* modify the logic of the UI and the control flow of associated process views as well (e.g., adding FGE *Edit Phone Number* and respective FE *Phone*, cf. Fig. 4a). The correct position of the change within the UIM can be determined by the hierarchical structure of the UI (cf. Fig. 4b). The changes of the UIM are then propagated to the process view; note that the latter is represented by the UIM. Finally, the change of the process view has to be propagated to the basis process model on which the view is created. For this propagation the concepts developed in the *proView*³ project can be applied [9,10].

5 Related Work

Task Models describe the actions to be performed by a user when interacting with an information system. Different variants of task models exist [11]. These approaches describe the goals, steps and operations of a UI. *Concurrent Task Trees (CTT)*, in turn, provide a hierarchical model supporting various types of tasks (e.g., automatic vs. manual task) and relationships between tasks (e.g., sequential vs. parallel execution) [12].

³ <http://www.dbis.info/proView>

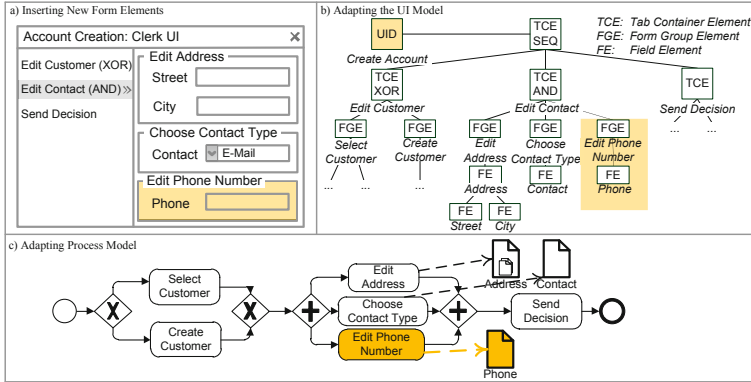


Fig. 4. Adapting Process Models through Changes in the UI Component

Model-Driven UI Development applies the principles of *Model-Driven Development* to UI development. Although a lot of competing approaches exist, an accepted standard is missing [13,14]. FlowiXML [15], for example, provides a methodology to develop UIs for business processes, taking the organizational structure as well as the process model into account. However, it does not allow for the automated generation of UIs. Based on FlowiXML, [16] describes user tasks through task models (i.e., CTT) within a process model. Based on these models, an abstract UI description is generated and transformed into a UI component at run-time. This approach allows for changes based on UIs and discusses how to manually align them with process models. However, automatic propagation is not supported. [17] transforms a process model into a human interaction perspective, which allows specifying data elements, user roles, tasks, and UI layout. After manually refining them, corresponding UIs are generated during run-time. Furthermore, data-centered process management offers a different (i.e., data-centered) view on processes. In particular, state transitions of process-related data elements are described. Based on this, UIs can be generated as well [18].

UI Generation in Existing PAIS is able to create UIs automatically (e.g., IBM Lombardi [19]). Single activities of a process model can be transformed into simple UIs, taking associated data elements into account (cf. ETPs). More complex scenarios are not covered. None of the approaches allows for the automatic generation of complex UIs based on process models. The adaptation of process models based on changes of the UI is only considered rudimentarily.

6 Conclusion

In this paper, we showed how UI components can be automatically created from entire process fragments and process models respectively. For this purpose, elementary and complex transformation patterns were identified and described.

Furthermore, a transformation method, which applies these patterns to create complex UIs based on process views, was introduced. Our approach further enables the propagation of UI changes (e.g., adding new input fields) to the associated process model and vice versa. Finally, we implemented our UI generation approach in a powerful proof-of-concept prototype [5]. In summary, our approach will contribute to reduce costs for PAIS development and maintenance.

Future research will address the execution aspects of process models and associated UIs as well. In this context, features such as jumping back to an already edited UI element will be supported by adapting the process instance.

References

1. Pradeep, H.: Process-User Interface Alignment: New Value From a New Level of Alignment. *Align Journal* (October 3, 2007)
2. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring Large Process Model Repositories. *Computers in Industry* 62(5), 467–486 (2011)
3. Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Inf. Sys.* 10(2), 93–129 (1998)
4. La Rosa, M., Wohed, P., Mendling, J., ter Hofstede, A.H.M., Reijers, H.A., van der Aalst, W.M.P.: Managing Process Model Complexity Via Abstract Syntax Modifications. *IEEE Transactions on Industrial Informatics* 7(4), 614–629 (2011)
5. Kolb, J., Hübner, P., Reichert, M.: Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems. Technical report, UIB 2012-04, Ulm University (2012)
6. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: *Proc. ACM SAC 2012, Riva del Garda (Trento), Italy* (2012)
7. Kolb, J., Reichert, M., Weber, B.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: Oppl, S., Fleischmann, A. (eds.) *S-BPM ONE 2012. CCIS, vol. 284*, pp. 237–251. Springer, Heidelberg (2012)
8. 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)
9. Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-centered Adaption of Large Process Models. In: *Proc. Intl. Conf. on Service Oriented Computing (ICSOC 2012), Shanghai, China* (to appear, 2012)
10. Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: *Proc. of the Business Process Management 2012 Demonstration Track, Tallinn, Estonia* (to appear, 2012)
11. Limbourg, Q., Vanderdonckt, J.: Comparing Task Models for User Interface Design. *The Handbook of Task Analysis for Human-Computer Interaction* 6 (2004)
12. Paternò, F., Mancini, C., Meniconi, S., Maria, V.S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: *Proc. IFIP TC13 Int'l Conf. on Human-Computer Interaction*, pp. 362–369 (1997)
13. Traetteberg, H., Molina, P.J.: Making Model-Based UI Design Practical: Usable and Open Methods and Tools. In: *Proc. IUI 2004*, pp. 376–377 (2004)
14. Lu, X.: Model Driven Development of Complex User Interface. In: *Proc. MoDELS 2007, Workshop on Model Driven Development of Advanced User Interfaces* (2007)

15. Garcia, J.G., Vanderdonckt, J., Calleros, J.M.G.: FlowiXML: A Step Towards Designing Workflow Management Systems. *Int'l Journal of Web Engineering and Technology* 4(2), 163–182 (2008)
16. Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: User Interface Derivation from Business Processes: A Model-Driven Approach for Organizational Engineering. In: *Proc. ACM SAC 2008*, pp. 553–560 (2008)
17. Sukaviriya, N., Sinha, V., Ramachandra, T., Mani, S., Stolze, M.: User-Centered Design and Business Process Modeling: Cross Road in Rapid Prototyping Tools. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) *INTERACT 2007*. LNCS, vol. 4662, pp. 165–178. Springer, Heidelberg (2007)
18. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards A Framework for Object-Aware Process Management. *Journal Software Maintenance and Evolution: Research & Practice* 23(4), 205–244 (2011)
19. Yang, S., Sun, Y., Waterhouse, J., Lau, D., Al-Hamwy, T.: Modeling and Implementing a Business Process Using WebSphere Lombardi Edition 7.1. In: *Proc. CASCON 2010*, pp. 374–375 (2010)

Embedding ‘Break the Glass’ into Business Process Models

Silvia von Stackelberg, Klemens Böhm, and Matthias Bracht

Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

Abstract. Break the Glass (BTG) is an important feature for authorization infrastructures, as it provides flexible access control in exceptional cases. Current realizations have two drawbacks: They neglect the need to manage authorization steps, and they do not take immediate process context into account. Our approach in turn embeds BTG functionality into business processes (BPs): The steps to perform BTG and the obligations compensating a BTG access for data are parts of the BPs. To support process designers in embedding BTG steps and obligations, we introduce an expressive annotation language for specifying BTG tasks for BP models. In particular, our language allows process designers to take BP context into account and to specify security constraints for role holders performing BTG tasks. Using our approach, one can efficiently specify and use context-aware BTG functionality for BPs.

Keywords: Security, process model annotation language, immediate context.

1 Introduction

Problem Statement. Security mechanisms are important for Business Process Management (BPM). For instance, authorization constraints specify which roles may perform a task or access certain data. However, such mechanisms sometimes are too rigid, and more flexibility is needed. To illustrate, emergencies (e.g., in E-health) and disaster management necessitate rights to access data in exceptional situations. Thus, a trade-off between security on the one hand and flexibility on the other hand needs to be facilitated.

The so-called *Break the Glass (BTG)* principle provides flexibility by allowing users to overcome access denials in exceptional cases [1]. The designer specifies in advance who, in particular situations, will have access rights he normally does not have. In line with [5], the prerequisites to “break the glass” from the application perspective are: (1) regular access is denied, (2) BTG access is foreseen for the exceptional case, (3) a user explicitly asks for access, (4) optionally, another user has to agree to this access. We call the sequence of steps when users ask for exceptional access *BTG steps* in the following. Next, *obligations* typically are part of BTG, i.e., operations that compensate¹ for the security violations. Obligations can be triggered immediately after breaking the glass (synchronously) or later (asynchronously).

¹ We use the term *compensation* for the execution of obligations. As a data access cannot be undone, it is at least mitigated by compensating actions.

Example 1 (E-Health). In the regular case, only dedicated persons, such as the family doctor of a patient, are authorized to access health-record data of patients. We assume that this data is stored externally and policies (e.g., sticky policies) specify authorizations for data access. In a life-threatening situation, other members of the medical staff might need access to the data. By *Breaking the Glass*, physicians who are not authorized in the regular case access the record in a controlled way. An exceptional access results in many obligations, such as auditing the data access, informing the family doctor, among others. We exemplarily focus on **O1**: At the end of the treatment process, the physician has to send a report to the family doctor. Here, the point of time when sending the report depends on the BP context, i.e., when the treatment of the patient has been finished. This obligation is asynchronous because it refers to a later point in time.

We envision integrating BTG functionality into BPMS. This is new and challenging, because existing approaches providing authorization infrastructures for BTG (e.g., [1], [5], and [9]) do not cover the following aspects: (1) Modelling BTG steps and obligations as part of the BP and executing them. (2) Considering BP-specific features, BP context in particular.

Regarding (1), related work leaves the execution of BTG steps and obligations to the application and views them as black boxes. However, a BTG access typically consists of several steps. The same holds for obligations. This asks for mechanisms to embed BTG steps and obligations into the BP, since the modelling of such steps and their execution is exactly the purpose of BPMSs.

The development of context-aware systems is a challenging research area. Most approaches take environmental context into account. *Immediate process context* in turn is information that characterizes the process itself. It refers to the execution state of a process instance, such as the state of tasks, associated actors, or objects to be accessed [12]. Regarding (2), combining immediate BP context with BTG functionality has several advantages, as we will explain in Section 2. But existing work does not take immediate BP context for BTG realizations into account.

Goals and Challenges: Our overall goal is to integrate BTG functionality into the BP and to have it executed by a BPMS. By doing so, we take BP context into account. To accomplish this, this paper focuses on the following goals:

- *Facilitating the embedding of BTG steps and obligations into BPs.* Without any support for the embedding, process designers have to model both the process logic and the security constraints for BTG functionality by hand. This requires profound security knowledge and thus is error prone; and it is time-consuming. Thus, there should be support at the process-modelling level. By using annotations for process models (e.g., [7], [11], and [14]), designers can rely on the modelling primitives they are used to. We develop an annotation language for BP models representing BTG functionality.
- *Context-aware annotation language.* As BP context is important for BTG functionality, the annotation language has to provide support for the coupling of contextual information with BTG tasks.

We leave the design and realization of an infrastructure supporting context-aware BTG to a future publication.

The goals lined out above are challenging, for the following reasons:

- As the embedding of BTG functionality into BP is new, the design of an expressive annotation language asks for a systematic requirements analysis.
- This results in the specification of a comprehensive set of expressions to represent BTG functionality (e.g., which tasks have to be performed, who is authorized).
- Current systems do not feature the coupling of immediate BP context with BTG functionality. To support this, we develop a representation of BP context process designers can easily use.

Contributions: We have developed new concepts to embed BTG functionality at the BP modelling layer. “Embedding” means that potential BTG steps and obligations are integrated into a BP, and BP context is taken into account. In particular, we make the following contributions:

- *Motivation.* We list advantages of using immediate context information for BTG functionality. As context can be any information, we provide expressions for the specification of context relevant for an application.
- *Specification of an annotation language allowing to represent BTG steps and obligations.* In particular, it allows to specify actors involved in BTG steps and BP-context-specific constraints for BTG options. By using these constructs, process designers can smoothly embed BTG support into BPMS. We specify a generic process fragment the BPMS has to execute in order to fulfill the specifications contained in annotations.

Paper structure: We motivate context-aware BTG functionality in Section 2. Section 3 lists requirements. Section 4 describes the annotation terms for BP models. Section 5 discusses related work, and Section 6 concludes.

2 Motivation for Contextual BTG Functionality

Immediate BP context relevant for BTG can be information on the core BP regarding the functional, behavioral, organizational, operational, and data aspects of process instances. We borrow these categories from [12]. – Coupling BTG functionality with BP context has the following advantages:

(1) *BTG steps may comprise BP-context-specific constraints:* BP schemas allow to impose control-flow constraints on BTG steps. For example, in the E-health scenario, there might be a task to determine the urgency of a treatment. A physician might be allowed for BTG only if this task has been performed. Such a constraint can be expressed by referring to the *execution state* of a BP instance. However, existing BTG approaches do not allow to specify this.

(2) *Specification of constraints for obligations:* Constraints for the execution of obligations can be specified in the same way as for BTG steps. In our scenario,

there might be one or several physicians involved in an treatment. An example of a constraint is to send an email only when several physicians have been involved.

(3) *Specification of BP context for obligation parameters:* In general, obligations are parameterized. For example, an obligation might say that an individual who accesses a data object in parallel to a BTG access on this data must be informed about the respective BTG action. In our example, the system must pass the email address of that individual to the application executing the obligation. By using BP-context information on *associated actors*, the system might determine the receivers of the email automatically.

(4) *Triggering asynchronous obligations:* Synchronous obligations are triggered immediately, together with the BTG action. Asynchronous obligations are triggered at an absolute or a relative point in time. In our example, the *execution state of a BP* determines when an obligation takes place (i.e., send the patient report when the last task of the treatment is finished). Being able to refer to execution states of tasks gives way to asynchronous obligations.

As these advantages are essential, it is important to combine BTG functionality with BP context.

3 Requirements for Annotation Language

To identify requirements on a language allowing to specify BTG functionality, we have analyzed BTG use cases in two different real-world scenarios, an E-employment and an E-health application. Following these analyses, a BTG-annotation-language must support the following aspects:

R1: *Security constraints for BTG users:* BTG functionality has to be provided in a controlled way, i.e., it must be specified at design time who shall obtain the BTG rights, namely to break the glass, to access data, and to repair the glass. Thus, the BTG vocabulary must distinguish different types of users involved in a BTG action. The vocabulary must allow to specify authorization and authentication constraints for these users.

R2: *BP-context-specific constraints:* It must be possible to specify the start time for BTG steps or obligations, i.e., *when* the tasks have to be performed, by taking the BP context into account. In particular, asynchronous obligations that rely on BP-context-specific conditions must be possible. Example 1 has motivated this. Further, it should be possible to represent conditions for the execution of BTG steps and obligations, i.e., *whether* tasks have to be performed. To illustrate, one might specify that an obligation is needed only if an external physician has worked on the emergency treatment. The specification of BP context must be user-friendly [6]. This means that process designers should not have to deal with the BP-engine-internal representation of BP context, but should be able to specify BP context at the abstraction level of BP models.

R3: *Parameters for obligations:* It must be possible to specify BP-context-specific parameters of obligations (e.g., associated actors), cf. Example 1.

4 Design

In this section we describe how we embed options for breaking the glass into BP models. We first describe the different BTG roles and motivate annotating process models with BTG functionality. We then say how we represent BP context. Finally, we describe the annotation language in brief.

4.1 BTG Roles

We introduce roles having BTG rights in the following. In line with [2], we distinguish three types of users involved in a BTG option: the first type are users who have the right to break the glass, i.e., users who activate a BTG case (e.g., patients who decide). We call the corresponding role *BTG Activator Role*. Second, the *BTG Access Role* are users who have the right to access a resource if BTG has happened (physicians in our example). In practice, it is possible that process participants have both rights. Third, the *BTG Compensator Role* are users who are allowed to perform obligations in the BTG case.

4.2 Embedding BTG Functionality into Business Processes

Our idea is to integrate BTG steps into the application process. Thus, the BP Engine controls their execution.

The question now is how to realize this embedding. We see several alternatives, with two extremes: to represent them within the process model or to dynamically adapt process instances at runtime if needed (ad-hoc adaptation).

With the first extreme, process designers embed any options for breaking the glass in the BP model, using conventional modelling primitives. Process events and gateways can represent these options for exception handling. This means that a process instance can perform any BTG case or not. This is likely to lead to very complex BP models, because a single BTG case already consists of a sequence of tasks and might have many corresponding obligations. However, BTG functionality is only needed in exceptional cases, and whether it is needed is known only at runtime. This observation leads to the second extreme, namely to enable ad-hoc changes at runtime, meaning that process instances deviate from the specified process model. This can be interpreted as a case of exception handling, and it affects only single process instances. Process designers have to specify allowed deviations in advance. This approach requires a BP Engine that is capable to deal with ad-hoc changes at runtime. Currently, there is only little support in BP Engines for this (e.g., by the AristaFlow BPM Suite [3]). If platform-independence is an issue, a solution currently cannot rely on these features.

Design Decision: Our approach is a middle ground. We let process designers specify BTG options in a BP model² with specific annotation vocabulary. The

² In line with our security-annotation language, we represent BTG annotations in BPMN process models. But our BTG approach is sufficiently general to be applied to other process-model-languages.

BPMS transforms these annotations by extending the BP schema with canned process fragments and executes them as part of the BP. By means of annotation terms, process designers specify who will have the various BTG rights for data and the constraints for enabling BTG.

4.3 Formalizing BP Context

It is the task of the Engine to manage BP context. As BP context is important for BTG, we need a way to represent it in the BP model. One way is to specify constraints for BP context relevant for BTG by using the internal representation of the BP Engine. But this is error-prone and time-consuming, since process designers typically are not familiar with the internal representations.

Design Decision: We propose a vocabulary to represent BP context in BTG annotations. We formalize BP context on the abstraction level of BP models by introducing functions for tasks and data that return values representing the BP context. These functions enable the specification of associated actors, tasks, and data objects to be used for the representation of temporal and causal BP-context constraints in the annotations. The BPMS transforms these specifications into representations the BP Engine can handle. This addresses R2.

We define the syntax and semantics of BP constraints in [13]. In brief, a BP-context constraint is a Boolean expression, using at least one of the following functions:

- The functions `performer(task)`, `data-user(object)`, `owner(object)` return subjects related to a BP instance, namely the actor performing a task instance, the actor accessing a data object, and the data owner respectively.
- The functions `start-time-exec(task)` and `end-time-exec(task)` return the start and end times of the execution of a task.
- The functions `start-time-access(object)` and `end-time-access(object)` return the start time and end time of access to a data object.
- `data-access(task)` returns the set of data objects accessed by a task.

This set of functions is sufficient for the applications we have studied. Using these functions, process designers can represent a comprehensive set of BP-context-constraints within annotations for BTG steps and obligations.

Example 2 (BP Context Constraints). To express that BTG is only allowed for adults, we set `exec=performer(activity-ID).age ≥ 18`. We specify an asynchronous start time depending on the execution time of a task by `start = end-time-exec(activity-ID)`. To say that an obligation has to be executed if several performers are involved we formulate `exec = performer(activity-ID-1) ≠ performer(activity-ID-2)`.

4.4 Specification of BTG Steps for BP Models

Our annotation language features the specification of security aspects and of BP-context constraints as follows: To represent authorizations for BTG steps, we

need two out of three BTG roles, namely *BTG Activator Role* and *BTG Access Role*. This accounts for R1. These role holders have the right to perform particular BTG tasks. The specifications for *BTGActivator* and *BTGAccessor* can have optional authentication refinements. Further, one can specify constraints on the start or the execution of BTG steps by means of parameters *start* and *exec*. The first one states when the BTG steps have to be executed, and which constraint must hold at this point in time. In contrast, *exec* specifies constraints that must hold for executing BTG steps in general. One BTG action can have many obligations, and obligation specifications can be complex. Thus, annotations contain a list of obligation-IDs which have to be executed when the glass has been broken. We annotate each obligation separately for the BP model.

A BTG annotation, starting with “<<BTG:”, contains a set of assignments for a specified vocabulary, and ends with “>>”. The parameter *right* specifies the nature of the access to a data *object* for which the glass can be broken. The assignments for *object* and *right* are obligatory. [13] gives a complete definition of the syntax and the semantics.

Example 3 (E-health (cont.)). Figure 1 displays the excerpt of a process model representing the visit of a patient to a physician [13]. To enable BTG functionality for health records of patients, we make use of the BTG-annotation term for activity “Check health-record availability”. Figure 1 graphs the annotation.

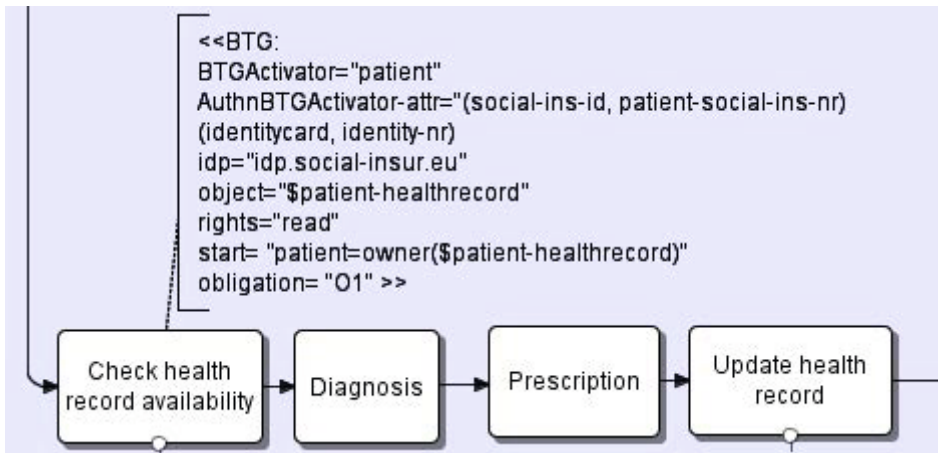


Fig. 1. Annotation of Activity “Check health-record availability”

The meaning of the annotation is as follows: To provide the BTG option, a process designer assigns patients whose health record might be accessed to *BTG Activator*. In our case, the patients have to agree to break the glass. As this is a security-relevant task, the process designer asks for authentication for the *BTG Activator*. In other words, the patient now has to authenticate himself, and this needs to be modelled. The authentication specification says that the IdP must

authenticate a patient by the AuthnBTGActivator attributes social insurance number and identity-card number. The data to be accessed are health records of the patients. We set "read" access rights for the BTG case. The condition `start` specifies that the glass can only be broken if the patient is the owner of the data object. The parameter obligation specifies that O1 must be executed.

A BTG annotation means that the BPMS has to provide options for BTG steps, as described in Section 4.2. To embed these steps into the BP, we rely on the fundamental technique of process fragments, enabling to re-use parts of process structures. Using our approach, executable BPs require specifications for BP-context constraints as well as for security (data access, authorizations and authentications for role holders). The BTG annotations contain these specifications, and our secure BPMS transforms them to the extended process model. By doing so, process fragments are generic and can be used in any BP model.

Figure 2 shows the process fragment for BTG steps. It represents the execution order for BTG tasks as well as conditions on BP-context constraints. The annotated role assignments specify authorizations for role holders.

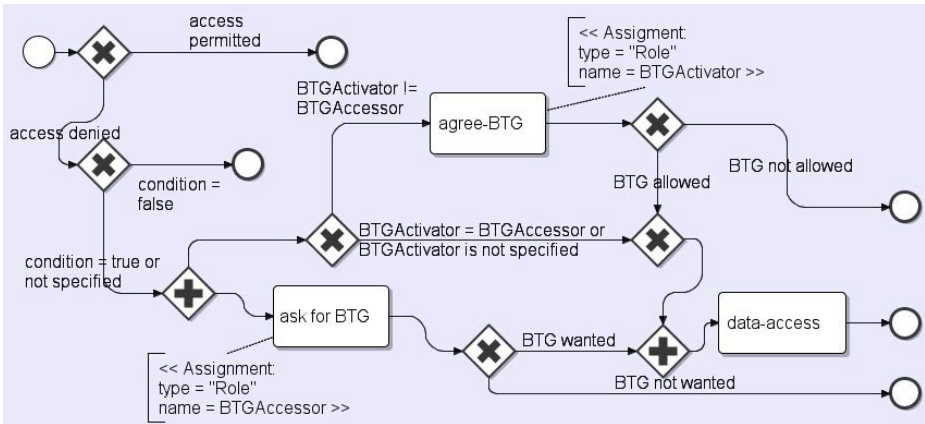


Fig. 2. Process Fragment for BTG steps

In line with annotations for BTG steps, we provide a vocabulary to describe obligation and represent them as language primitives. [13] gives a complete definition of the syntax and the complete semantics. The specification fulfills R3. Regarding the transformation, the system substitutes each BTG annotation with a BTG-process fragment, specifies the extended BP model, and generates the access-control policy.

5 Related Work

Regarding the integration of BTG into business processes, work from three research threads is of particular relevance: access-control policies for BTG, context-aware, security-related research for BPs, and security-annotation languages.

BTG access control policies: Several approaches realize BTG by implementing access control policies ([1], [8], [5] and [9]). But they do not feature support for BP context, due to their generic nature, and do not address the management of BTG steps and obligations from the perspective of the application, as we do. Our approach in turn does not focus on a BTG-enabled access-control-policy language, but on an infrastructure for embedding BTG functionality into business processes. We manage BTG authorization functionality to some degree by tasks being part of the BP. To illustrate, the BPMS executes process branches with BTG options, instead of offering BTG options by the authorization component, as in [5]. Our approach makes access control rules for BTG easier.

Context-aware security support: According to the classification in [12], our work focuses on immediate context. We thereby also consider security and privacy aspects. The activity and object context in [10] is similar to our understanding of immediate context. Most recent work on context-aware BP (e.g., [9]) is confined to the context of the *environment*, which we do not address. We employ BP context for security aspects. [10] summarizes well-known approaches on BP-context-aware access control methods. To bind access rights to the execution time of tasks (strict least privilege), the approaches use immediate BP context (e.g., [10]). Further, the realization of Binding and Separation of Duties [4] requires immediate context information at runtime. Our purpose differs from the discussed approaches. We address context-aware BTG functionality.

Security modelling languages: [7], [11], [14], among others, propose annotation languages to represent security constraints in BP models, but lack in two aspects: None of them takes BTG into account; None of them provides features to represent BP context as part of the annotation language. Our language is the first to cover this.

To our knowledge, our approach is unique in that we embed BP-context-constraints into the BTG-annotation-language, and use this contextual information for the embedding of process fragments by taking authorization rules into account.

6 Conclusions

The “Break the Glass” concept facilitates controlled access to data in exceptional situations. To our knowledge, this article has been first to provide BTG functionality for business processes. As breaking the glass and compensating a BTG action require several tasks, BTG steps and obligations should be embedded in processes. We have shown that using BP context for BTG tasks is essential.

To disburden the process designer from modelling BTG steps and obligations by hand, we have proposed a vocabulary for annotating the process model with BTG functionality. In particular, we take BP context into account. This reduces the design effort significantly.

Acknowledgements. This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement n° 216287 (TAS³ - Trusted Architecture for Securely Shared Services) as well as by the European Social Fund and by the Ministry Of Science, Research and the Arts Baden-Württemberg.

References

1. Brucker, A.D., Petritsch, H.: Extending Access Control Models with Break-glass. In: SACMAT (2009)
2. Chadwick, D. (ed.): Design of Identity Management, Authentication and Authorization Infrastructure, TAS3 Deliverable 7.1, Version 3.0.1 (2010)
3. Dadam, P., Reichert, M., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L.T., Jurisch, M., Kreher, U., Göser, K.: From ADEPT to AristaFlow BPM Suite: A Research Vision Has Become Reality. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBP, vol. 43, pp. 529–531. Springer, Heidelberg (2010)
4. Bertino, E., Martino, L., Paci, F., Squicciarini, A.: Security for Web Services and Service-Oriented Architectures. Springer (2010)
5. Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R., Antunes, L.: How to securely break into RBAC: The BTG-RBAC model. In: ACSAC, pp. 23–31 (2009)
6. Hallerbach, A., Bauer, T., Reichert, M.: Context-based configuration of process variants. In: TCoB, pp. 31–40 (2008)
7. Mülle, J., von Stackelberg, S., Böhm, K.: Modelling and Transforming Security Constraints in Privacy-Aware Business Processes. In: SOCA, pp. 1–4 (2011)
8. Alqatawna, J., Rissanen, E., Sadighi, B.: Overriding of Access Control in XACML. In: POLICY, pp. 87–95 (2007)
9. Marinovic, S., Craven, R., Ma, J., Dulay, N.: Rumpole: A Flexible Break-glass Access Control Model. In: SACMAT, pp. 73–82 (2011)
10. Park, S.H., Eom, J.H., Chung, T.M.: A Study on Access Control Model for Context-Aware Workflow. In: INC, IMS and IDC, pp. 1526–1531 (2009)
11. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN extension for the modeling of security requirements in business processes. Trans. Inf. Syst. – IE-ICE E90-D, 745–752 (2007)
12. Rosemann, M., Recker, J.C., Flender, C.: Contextualisation of business processes. Int. Journ. of Business Process Integration and Management 3(1), 47–60 (2008)
13. von Stackelberg, S., Böhm, K., Bracht, M.: Embedding BTG into Business Processes (2012), <http://dbis.ipd.kit.edu/1860.php>
14. Wolter, C., Schaad, A.: Modeling of Task-Based Authorization Constraints in BPMN. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 64–79. Springer, Heidelberg (2007)

Author Index

- Abdullah, Haris II-797
Abushnagh, Yousef II-871
Adams, Michael I-212
Afsarmanesh, Hamideh II-825
Aït-Ameur, Yamine II-879
Akbarinia, Reza II-825
Almendros-Jiménez, Jesús M. II-915
Álvarez, Hernán II-763
Amyot, Daniel II-700
Angajala, Prabhu K. I-323
Ayat, Naser II-825
- Baldoni, Roberto II-492
Baron, Mickaël II-897
Barone, Daniele II-700
Bazhar, Youness II-879
Bellatreche, Ladjel II-879, II-897
Bereta, Konstantina II-932
Bergamaschi, Sonia II-645, II-736
Binz, Tobias I-416
Biswas, Debmalya II-511
Böhm, Klemens I-455
Borgida, Alex II-700
Bouzeghoub, Mokrane I-128
Bracht, Matthias I-455
Breitenbücher, Uwe I-416
Breitman, Karin K. II-646
Brook, Matthew II-871
Buccafurri, Francesco II-855
Buijs, Joos C.A.M. I-305
- Cabanillas, Cristina I-56
Casanova, Marco A. II-646
Chakroun, Chedlia II-879
Chowdhury, Nafisa Afrin II-664
Comi, Antonello II-855
Conforti, Raffaele I-212
Cortes-Cornax, Mario I-110
Costa, Fernando I-425
Costa, João I-380
Cruz, Isabel II-645
Cuzzocrea, Alfredo II-527
- Dadam, Peter I-1
Dalpiaz, Fabiano I-232
- Damova, Mariana II-807
da Silva, Alberto Rodrigues I-398
Datcu, Mihai II-932
De Giacomo, Giuseppe I-194
de Macêdo, José A.F. II-646
de Oliveira, Anderson Santana II-470
De Virgilio, Roberto II-780
Di Ciccio, Claudio I-194
Di Martino, Beniamino II-628
Dorn, Christoph I-362
Dou, Dejing II-664
Dumitru, Corneliu Octavian II-932
Dupuy-Chessa, Sophie I-110
- Edwards, George I-362
Englund, Cristofer II-752
Espinoza-Molina, Daniela II-932
- Farinella, Tania II-736
Felli, Paolo I-194
Ferreira, Paulo I-380, I-425
Ferreira, Tiago Lopes I-398
Ferscha, Alois II-469, II-602
Ficco, Massimo II-628
Fleischhacker, Daniel II-718
Fokou, Géraud II-897
Folino, Francesco I-287
Furtado, Antonio L. II-646
- Garbis, George II-932
Gater, Ahmed I-128
Ge, Mouzhi II-682
Giorgini, Paolo I-232
Grigori, Daniela I-128
Grönvall, John-Fredrik II-752
Grubert, Jens II-863
Guarascio, Massimo I-287
- Hepp, Martin II-682
Hitzler, Pascal II-807
Hölzl, Gerold II-602
Horkoff, Jennifer II-700
Hu, Yuxiao I-194
Hübner, Paul I-444
Hull, Richard I-433

- Iordache, Raluca I-182
- Jain, Prateek II-807
- Jean, Stéphane II-879, II-897
- Jiang, Lei II-700
- Jiménez, Claudia II-763
- Joshi, Amit Krishna II-807
- Kaczmarek, Krzysztof II-843
- Kantere, Verena I-146
- Karpathiotakis, Manos II-932
- Khouri, Selma II-897
- Kolb, Jens I-444
- Kopp, Oliver I-416
- Koubarakis, Manolis II-932
- Kovaceva, Jordanka II-752
- Kurz, Marc II-602
- Kyzirakos, Kostis II-932
- Lamanna, D. Davide II-492
- Lapouchnian, Alexei I-342
- La Rosa, Marcello I-212
- Lax, Gianluca II-855
- Letier, Emmanuel I-110
- Leymann, Frank I-416
- Lincoln, Maya I-74
- Linderman, Mark I-323
- Lindman, Magdalena II-752
- Lodi, Giorgia II-492
- Lohrmann, Matthias I-38
- Lotz, Volkmar II-470
- Madria, Sanjay K. I-323
- Maggi, Fabrizio Maria I-20, I-250
- Marrella, Andrea I-268
- Matei, Alexandru I-110
- Mecella, Massimo I-194, I-268
- Medvidovic, Nenad I-362
- Molch, Katrin II-932
- Moldoveanu, Florica I-182
- Monakova, Ganna I-92
- Morgan, Graham II-871
- Mowbray, Miranda II-475
- Muñoz-Escóí, F.D. II-549
- Mylopoulos, John I-342, II-700
- Negrão, André Pessoa I-380
- Nikolaou, Charalampos II-932
- Nixon, Lyndon II-863
- Nuutila, Esko II-797
- Pallardó-Lozoya, M.R. II-549
- Papanikolaou, Nick II-620
- Paulo, J. II-584
- Pearson, Siani II-469, II-475
- Pereira, J. II-584
- Pinheiro, Ângela M.A. II-646
- Po, Laura II-736
- Pontieri, Luigi I-287
- Reichert, Manfred I-2, I-38, I-444
- Reis, P. II-584
- Reitmayr, Gerhard II-863
- Resinas, Manuel I-56
- Rieu, Dominique I-110
- Rinderle-Ma, Stefanie I-1
- Rinne, Mikko II-797
- Rodríguez-Castro, Bene II-682
- Rosaci, Domenico II-855
- Ruiz-Cortés, Antonio I-56
- Ruiz-Fuertes, M.I. II-549
- Russo, Alessandro I-268
- Saccá, Domenico II-527
- Sacramento, Eveline R. II-646
- Salnitri, Mattia I-232
- Schumm, David I-416
- Schunselaar, Dennis M.M. I-20
- Schwarz, Gottfried II-932
- Scicluna, James II-863
- Sharp, Craig II-871
- Sheth, Amit P. II-807
- Sidorova, Natalia I-20
- Simão, José II-566
- Sioutis, Michael II-932
- Sousa, A. II-584
- Souza, Vítor E. Silva I-342
- Srinivasa, Srinath II-789
- Stuckenschmidt, Heiner II-718
- Sun, Yutian I-433
- ter Hofstede, Arthur H.M. I-212
- Tineo, Leonid II-763
- Törmä, Seppo II-797
- Tran, Huy I-164
- Ushaw, Gary II-871
- Vaculín, Roman I-433
- Valduriez, Patrick II-825
- van der Aalst, Wil M.P. I-20, I-305

- van Dongen, Boudewijn F. I-305
Vassos, Stavros II-932
Veiga, Luís I-380, I-425, II-566
Venticinque, Salvatore II-628
Verma, Kunal II-807
Vidal, Vânia M.P. II-646
Vidyasankar, Krishnamurthy II-511
Völker, Johanna II-718
von Stackelberg, Silvia I-455
Wasser, Avi I-74
Westergaard, Michael I-250
Yeh, Peter Z. II-807
Yu, Eric II-700
Zdun, Uwe I-164
Zhou, Xiaofang I-1