Table of Contents


Preface ........................................................................................................................................ vii
Programme Committee ...................................................................................................... ix
Organising Committee ....................................................................................................... x
Welcome from the Organising Committee .................................................................................. xi
CORE - Computing Research & Education ................................................................................... xii
ACSW Conferences and the Australian Computer Science Communications ............................. xiv
ACSW and APCCM 2014 Sponsors ............................................................................................... xvi

Keynote

The Many Faces of Operationalization in Goal-Oriented Requirements Engineering ................. 3
  Fabiano Dalpiaz, Vítor E. Silva Souza, John Mylopoulos

Invited Paper

The Role of Software Models in Developing New Software Systems: A Case Study in Project Management .............................................................................................................. 11
  Koichiro Ochimizu

Contributed Papers

CARE - A Constraint-Based Approach for Re-Establishing Conformance-Relationships ............... 19
  Johannes Schoenboeck, Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Wieland Schwinger, Manuel Wimmer, Martin Wischenbart

Variability in Artifact-Centric Process Modeling: The Hetero-Homogeneous Approach ............... 29
  Christoph Schütz, Michael Schrefl

Conflict Resolution for On-the-fly Change Propagation in Business Processes ............................. 39
  Shamila Mafazi, Wolfgang Mayer, Markus Stumptner

Applying a Test for Atomicity of Method Fragments .................................................................... 49
  Ben Rogers, Brian Henderson-Sellers

Data-driven Requirements Modeling: Some Initial Results with i* .............................................. 55
  Aditya Ghose, Metta Santiputri, Aya Saraswati, Hoa Khanh Dam

Predicting Requirements Changes by Focusing on the Social Relations ..................................... 65
  Takako Nakatani, Toshihiko Tsumaki
Preface

The Asia-Pacific Conference on Conceptual Modelling is celebrating its 10th edition! This volume contains the proceedings of the 10th Asia-Pacific Conference on Conceptual Modelling (APCCM 2014), held at the Auckland University of Technology, New Zealand from January 20 to 23, 2014 again as part of the Australasian Computer Science Week (ACSW 2014).

The APCCM series focuses on disseminating the results of innovative research in conceptual modelling and related areas, and provides an annual forum for experts from all areas of computer science and information systems with a common interest in the subject. The scope of APCCM 2014 includes areas such as:

- Business, enterprise, process and services modelling;
- Concepts, concept theories and ontologies;
- Conceptual modelling and user participation;
- Conceptual modelling for decision support and expert systems; digital libraries; e-business, e-commerce and e-banking systems; health care systems; knowledge management systems; mobile information systems; user interfaces; and Web-based systems;
- Conceptual modelling of semi-structured data and XML;
- Conceptual modelling of spatial, temporal and biological data;
- Conceptual modelling quality;
- Conceptual models for cloud computing applications;
- Conceptual models for supporting requirement engineering;
- Conceptual models in management science;
- Design patterns and object-oriented design;
- Evolution and change in conceptual models;
- Implementations of information systems;
- Information and schema integration;
- Information customisation and user profiles;
- Information recognition and information modelling;
- Information retrieval, analysis, visualisation and prediction;
- Information systems design methodologies;
- Knowledge discovery, knowledge representation and knowledge management;
- Methods for developing, validating and communicating conceptual models;
- Models for the Semantic Web;
- Philosophical, mathematical and linguistic foundations of conceptual models;
- Reuse, reverse engineering and reengineering; and
- Software engineering and tools for information systems development.

The program committee has selected the contributed papers from 17 submissions. All submitted papers have been refereed by at least three international experts, and have been discussed thoroughly. The six best papers judged by the program committee members have been accepted as full papers and two additional short papers are included in this volume.

The organising committee is honoured that Prof. John Mylopoulos and Prof. Koichiro Ochimizu accepted the invitation to give a talk at APCCM 2014.

Prof. John Mylopoulos holds a distinguished professor position (chiara fama) at the University of Trento, and a professor emeritus position at the University of Toronto. He presented the APCCM 2014 keynote on “Goal-Oriented Requirements Engineering”.

Prof. Koichiro Ochimizu is Vice President of Japan Advanced Institute of Science and Technology (JAIST), and Research Professor and Director of Center for Highly Dependable Embedded Systems Technology of JAIST, Japan. The topic of his invited talk was “The Role of Software Models in Developing New Software Systems: A Case Study in Project Management”.

The program committee selected the articles SQL-Sampler: A Tool to Visualize and Consolidate Domain Semantics by Perfect SQL Sample Data by Van Tran Bao Le, Sebastian Link and Flavio Ferrarotti for the APCCM 2014 Best Student Paper, and Data-driven Requirements Modeling: Some Initial Results with i* by Aditya Ghose, Metta Santiputri, Ayu Saraswati, and Hoa Khanh Dam for the APCCM 2014 Best Paper Award. The awards includes a cash prize in the amount of NZD 500 for each individual paper. The Best Student Paper was sponsored by the Computing Research and Education Association of Australasia (CORE), and the Best Paper Award was sponsored by the Department of Computer Science, The University of Auckland, New Zealand, and by the Advanced Computing Research Centre, University of South Australia, Adelaide.

We wish to thank all authors who submitted papers and all the conference participants for the fruitful discussions. We are grateful to the members of the program committee and the additional reviewers for their timely expertise in carefully reviewing the papers. We would like to thank the APCCM 2014 Publicity
Chair Dr Markus Kirchberg, Matt Selway and Andreas Jordan for announcing the event. Finally, we wish to express our appreciation to the local organisers at the Auckland University of Technology for preparing the ACSW 2014 event in New Zealand and would like to thank ACSW 2014 for sponsoring the keynote of Prof. John Mylopoulos.

Georg Grossmann  
University of South Australia  
Motoshi Saeki  
Tokyo Institute of Technology  
APCCM 2014 Programme Chairs  
January 2014
Conference Organisation

Program Committee Chairs

Georg Grossmann, University of South Australia
Motoshi Saeki, Tokyo Institute of Technology

Publicity Chair

Markus Kirchberg, National University of Singapore
Matt Selway, University of South Australia
Andreas Jordan, University of South Australia

Program Committee Members

João Paulo A. Almeida, Federal University of Espírito Santo, Brazil
Boualem Benatallah, University of New South Wales, Australia
Marko Boškovic, Research Studios Austria
Ross Brown, Queensland University of Technology, Australia
Gillian Dobbie, University of Auckland, New Zealand
Flavio Ferrarotti, Victoria University Wellington, New Zealand
Dragan Gašević, Athabasca University, Canada
Sven Hartmann, Clausthal University of Technology, Germany
Brian Henderson-Sellers, University of Technology Sydney, Australia
Marta Indulska, University of Queensland, Australia
Markus Kirchberg, National University Singapore
Hiroyuki Kitagawa, Tsukuba University, Japan
Yasushi Kiyoki, Keio University, Japan
Aneesh Krishna, Curtin University of Technology, Australia
Alberto Laender, University of Minas Gerais, Brazil
Lam-Son Le, University of Wollongong, Australia
Chiang Lee, National Cheng-Kung University, Taiwan
Sebastian Link, University of Auckland, New Zealand
Hui Ma, Victoria University Wellington, New Zealand
Takako Nakatani, Tsukuba University, Japan
Shamkant Navathe, Georgia Institute Of Technology, USA
Martin Necasky, Charles University, Czech Republic
Wilfred Ng, Hong Kong University of Science and Technology
Jolita Ralyte, University of Geneva, Switzerland
Jan Recker, Queensland University of Technology, Australia
Uwe Roehm, University of Sydney, Australia
Michael Rosemann, Queensland University of Technology
Michael Schrefl, Johannes Kepler University Linz, Austria
Nigel Stanger, University of Otago, New Zealand
Markus Stumptner, University of South Australia
Ernest Teniente, Universitat Politècnica de Catalunya, Spain
Bernhard Thalheim, Christian-Albrechts-University Kiel, Germany
Riccardo Torlone, Roma Tre University, Italy
Qing Wang, Australian National University, Australia
Eric Yu, University of Toronto, Canada

Additional Reviewers

Michael Karlinger, Johannes Kepler University Linz, Austria
Christoph Schütz, Johannes Kepler University Linz, Austria
Organising Committee

Chairs
Tony Clear and Russel Pears

Venue
Tony Clear

Communications
Russel Pears, Hui Ling Tan, Melanie Curry-Irons and Ryan Butler

Finance
Alison Clear and Eva Ihaia

Sponsorship
Stephen Thorpe

Operations
Adam Winship and Eva Ihaia

Programme, proceedings and booklet
Alison Clear

Catering and registration
AUT Hospitality Services
Welcome from the Organising Committee

On behalf of the Organising Committee, it is our pleasure to welcome you to Auckland and to the 2014 Australasian Computer Science Week (ACSW 2014). Auckland is New Zealand’s largest urban area with a population of nearly one and a half million people. As the centre of commerce and industry, Auckland is the most vibrant, bustling and multicultural city in New Zealand. With the largest Polynesian population in the world, this cultural influence is reflected in many different aspects of city life. ACSW 2014 will be hosted at the City Campus of Auckland University of Technology (AUT), which is situated just up from the Town Hall and the Auckland central business district. ACSW is the premier event for Computer Science researchers in Australasia. ACSW2014 consists of conferences covering a wide range of topics in Computer Science and related areas, including:

- Australasian Computer Science Conference (ACSC) (Chaired by Bruce Thomas and Dave Parry)
- Australasian Computing Education Conference (ACE) (Chaired by Jacqueline Whalley and Daryl D’Souza)
- Australasian Information Security Conference (AISC) (Chaired by Udaya Parampalli and Ian Welch)
- Australasian User Interface Conference (AUIC) (Chaired by Burkhard C. Wünsche and Stefan Marks)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Bahman Javadi and Saurabh Kumar Garg)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by James Warren)
- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Georg Grossmann and Motoshi Saeki)
- Australasian Web Conference (AWC) (Chaired by Andrew Trotman)

This year reflects an increased emphasis for ACSW on community building. Complementing these published technical volumes therefore, ACSW also hosts two doctoral consortia and a number of associated workshops, including those for the Heads and Professors of Computer Science, plus for the first time the ‘Australasian Women in Computing Celebration’. Naturally in addition to the technical program, there are a range of events, which aim to provide the opportunity for interactions among our participants. A welcome reception will be held in the atrium of the award winning newly built Sir Paul Reeves Building, which has integrated the city campus as a hub for student activity and provides a wonderful showcase for this year’s ACSW. The conference banquet will be held on campus in one of the reception rooms in this impressive complex.

Organising a multi-conference event such as ACSW is a challenging process even with many hands helping to distribute the workload, and actively cooperating to bring the events to fruition. This year has been no exception. We would like to share with you our gratitude towards all members of the organising committee for their combined efforts and dedication to the success of ACSW2014. We also thank all conference co-chairs and reviewers, for putting together the conference programs which are the heart of ACSW, and to the organisers of the symposia, workshops, poster sessions and accompanying conferences. Special thanks to Alex Potanin, as the steering committee chair who shared valuable experiences in organising ACSW and to John Grundy as chair of CoRE for his support for the innovations we have introduced this year. We’d also like to thank Hospitality Services from AUT, for their dedication and their efforts in conference registration, venue, catering and event organisation. This year we have secured generous support from several sponsors to help defray the costs of the event and we thank them for their welcome contributions. Last, but not least, we would like to thank all speakers, participants and attendees, and we look forward to several days of stimulating presentations, debates, friendly interactions and thoughtful discussions.

We hope your stay here will be both rewarding and memorable, and encourage you to take the time while in New Zealand to see some more of our beautiful country.

Tony Clear
Russel Pears
School of Computer & Mathematical Sciences
ACSW2014 General Co-Chairs
January, 2014
CORE welcomes all delegates to ACSW2014 in Auckland. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The component conferences of ACSW have changed over time with additions and subtractions ACSC, ACE, AISC, AUIC, AusPDC, HKM, ACDC, APCCM, CATS and AWC have now been joined by the Australasian women in computing celebration (AWIC), two doctoral consortia (ACDC and ACE-DC)and an Australasian Early Career Researchers Workshop (AECRW) which reflect the evolving dimensions of ACSW and build on the diversity of the Australasian computing community.

In 2014, we have again chosen to feature a small number of keynote speakers from across the discipline: Anthony Robins (ACE), John Mylopoulos (APCCM), and Peter Gutmann (AISC). I thank them for their contributions to ACSW2014. The efforts of the conference chairs and their program committees have led to strong programs in all the conferences, thanks very much for all your efforts. Thanks are particularly due to Tony Clear, Russel Pears and their colleagues for organising what promises to be a vibrant event. Below I outline some of COREs activities in 2012/13.

I welcome feedback on these including other activities you think CORE should be active in.

The major sponsor of Australian Computer Science Week:
- The venue for the annual Heads and Professors meeting
- An opportunity for Australian & NZ computing staff and postgrads to network and help develop their research and teaching
- Substantial discounts for attendees from member departments
- A doctoral consortium at which postgrads can seek external expertise for their research
- An Early Career Research forum to provide ECRs input into their development

Sponsor of several research, teaching and service awards:
- Chris Wallace award for Distinguished Research Contribution
- CORE Teaching Award
- Australasian Distinguished Doctoral Dissertation
- John Hughes Distinguished Service Award
- Various Best Student Paper awards at ACSW

Development, maintenance, and publication of the CORE conference and journal rankings. In 2013 this includes a new portal with a range of holistic venue information and a community update of the CORE 2009 conference rankings.

Input into a number of community resources and issues of interest:
- Development of an agreed national curriculum defining Computer Science, Software Engineering, and Information Technology
- A central point for discussion of community issues such as research standards
- Various submissions on behalf of Computer Science Departments and Academics to relevant government and industry bodies, including recently on Australian Workplace ICT Skills development, the Schools Technology Curriculum and the Mathematics decadal plan

Coordination with other sector groups:
- Work with the ACS on curriculum and accreditation
- Work with groups such as ACDict and government on issues such as CS staff performance metrics and appraisal, and recruitment of students into computing
- A member of CRA (Computing Research Association) and Informatics Europe. These organisations are the North American and European equivalents of CORE.
- A member of Science & Technology Australia, which provides eligibility for Science Meets Parliament and opportunity for input into government policy, and involvement with Science Meets Policymakers

A new Executive Committee from 2013 has been looking at a range of activities that CORE can lead or contribute to, including more developmental activities for CORE members. This has also included a revamp of the mailing lists, creation of discussion forums, identification of key issues for commentary and lobbying, and working with other groups to attract high aptitude students into ICT courses and careers. Again, I welcome your active input into the direction of CORE in order to give our community improved visibility and impact.
CORE’s existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2013, and look forward to the continuing shaping and development of CORE in 2014.

John Grundy
President, CORE
January, 2014
The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as Volume 0). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

2015. Volume 37. Host and Venue - University of Western Sydney, NSW.


2013. Volume 35. Host and Venue - University of South Australia, Adelaide, SA.
2012. Volume 34. Host and Venue - RMIT University, Melbourne, VIC.
2011. Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.
2010. Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.
2008. Volume 30. Host and Venue - University of Wollongong, NSW.
2007. Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.
2006. Volume 28. Host and Venue - University of Tasmania, TAS.
1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).
1989. Volume 11. Host and Venue - University of Wollongong, NSW.
1987. Volume 9. Host and Venue - Deakin University, VIC.
1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.
1983. Volume 5. Host and Venue - University of Sydney, NSW.
1982. Volume 4. Host and Venue - University of Western Australia, WA.
1981. Volume 3. Host and Venue - University of Queensland, QLD.
1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.
1979. Volume 1. Host and Venue - University of Tasmania, TAS.
1978. Volume 0. Host and Venue - University of New South Wales, NSW.
### Conference Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Conference Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDC</td>
<td>Australasian Computing Doctoral Consortium</td>
</tr>
<tr>
<td>ACE</td>
<td>Australasian Computing Education Conference</td>
</tr>
<tr>
<td>ACSC</td>
<td>Australasian Computer Science Conference</td>
</tr>
<tr>
<td>ACSW</td>
<td>Australasian Computer Science Week</td>
</tr>
<tr>
<td>ADC</td>
<td>Australasian Database Conference</td>
</tr>
<tr>
<td>AISC</td>
<td>Australasian Information Security Conference</td>
</tr>
<tr>
<td>APCCM</td>
<td>Asia-Pacific Conference on Conceptual Modelling</td>
</tr>
<tr>
<td>AUIC</td>
<td>Australasian User Interface Conference</td>
</tr>
<tr>
<td>AusPDC</td>
<td>Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)</td>
</tr>
<tr>
<td>AWC</td>
<td>Australasian Web Conference</td>
</tr>
<tr>
<td>CATS</td>
<td>Computing: Australasian Theory Symposium</td>
</tr>
<tr>
<td>HIKM</td>
<td>Australasian Workshop on Health Informatics and Knowledge Management</td>
</tr>
</tbody>
</table>

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.
We wish to thank the following sponsors for their contribution towards this conference.

**Host Sponsor**

![AUT Logo](AUT.png)

Auckland University of Technology,
www.aut.ac.nz

**Platinum Sponsor**

![DATACOM Logo](DATACOM.png)

DATACOM,
www.datacom.com.au

**Gold Sponsor**

![University of Auckland Logo](UOA.png)

The University of Auckland,
www.auckland.ac.nz

**Silver Sponsors**

![COLAB Logo](COLAB.png)

COLAB - AUT Design+Creative Technologies,
colab.aut.ac.nz

![IT Professionals Logo](ITP.png)

Institute of IT Professionals, New Zealand,
www.iitp.org.nz

![CORE Logo](CORE.png)

CORE - Computing Research and Education,
www.core.edu.au

**Bronze Sponsor**

![SERL Logo](SERL.png)

SERL - AUT Software Engineering Research Laboratory,
www.serl.aut.ac.nz

**Australian Computer Society**

www.acs.org.au

**Publication Sponsor**

![UWS Logo](UWS.png)

University of Western Sydney,
www.uws.edu.au

**Sponsors of APCCM**

![Department of Computer Science Logo](DeptCS.png)

Department of Computer Science
The University of Auckland,
www.cs.auckland.ac.nz

![Advanced Computing Research Centre Logo](ACRC.png)

Advanced Computing Research Centre,
University of South Australia
acrc.unisa.edu.au
KEYNOTE TALK
The Many Faces of Operationalization in Goal-Oriented Requirements Engineering

Fabiano Dalpiaz\textsuperscript{1}, Vitor E. Silva Souza\textsuperscript{2} and John Mylopoulos\textsuperscript{3}

\textsuperscript{1} Utrecht University, P.O Box 80125, 3508 TC, Utrecht, the Netherlands
\textsuperscript{2} Federal University of Espírito Santo, Av. Fernando Ferrari, 514/CT-7, Goiabeiras, Vitória ES, Brazil
\textsuperscript{3} University of Trento, Via Sommarive 5, I-38123, Povo, Trento TN, Italy

\texttt{f.dalpiaz@uu.nl, vitorsouza@inf.ufes.br, jm@disi.unitn.it}

Abstract. Goal models have been used in Requirements Engineering (RE) to elicit, model and analyse stakeholder requirements. In a goal model, stakeholder requirements are represented as root-level goals that are iteratively refined through AND/OR-refinements to eventually yield a specification consisting of functions the system-to-be needs to implement, as well non-functional requirements and domain assumptions. The association of a function to a goal is called operationalization in the sense that the function specifies how a goal can be made operational. We focus on the concept of operationalization and propose several extensions to account for operationalizations of non-functional and adaptation requirements, as well as behavioural specifications.

Keywords: Goal-oriented requirements engineering, goal model, operationalization.

1 Introduction

Goal orientation in Requirements Engineering (hereafter RE) is founded on the premise that requirements are goals that stakeholders want fulfilled by the system-to-be. Goal orientation was proposed about 20 years ago in (Dardenne et al., 1993) as an improvement over traditional RE techniques that focused on the identification of functions that the system-to-be needs to implement. A goal model explains why are these functions needed and how they contribute to the fulfillment of what the stakeholders want. Moreover, each goal model defines a problem space that includes alternative ways of fulfilling root-level goals. Research on goal-oriented requirements engineering has been conducted in many research groups around the world, for example (Anton and Potts, 1998), (Kaiya et al, 2002), Yu and Mylopoulos, 1998), (Kavakli, 2002). An early survey of research on the topic can be found in (van Lamsweerde, 2001).

These studies extended the software engineering process upstream, so that it starts with stakeholder wants/needs, rather than the functions the system-to-be needs to perform. Thanks to its acknowledged advantages, goal orientation has captured centre stage in RE research, as the dominant technique for requirements elicitation, modeling and analysis.

A goal model is constructed by iteratively refining the goals elicited from stakeholders into simpler goals through AND/OR refinements with obvious semantics: If a goal $G$ is AND/OR-refined into subgoals $G_1$, $G_2$, …, $G_n$, then fulfillment of all/at least one of $G_1$, $G_2$, …, $G_n$ leads to fulfillment of $G$. Eventually, the subgoals obtained through refinements are simple enough (atomic or leaf-level) that they can be fulfilled by a function (aka action/task) that an external agent or the system-to-be can perform. Such functions operationalize leaf-level goals. Operationalizations cross the boundary between problem space (requirements modeled as goals) and the solution space (functional specification).

The main objective of this position paper is to focus on the concept of operationalization and propose extensions that have been found useful in using goal models to capture non-functional requirements (rather than functional ones) (Mylopoulos et al., 1992), also requirements for adaptive systems (Souza et al., 2013a,b) and behavioural specifications of requirements (Dalpiaz et al., 2013).

Our study reviews the fundamental concepts that goal models are founded on (Section 2), then sketches the history of operationalization in Natural and Social Sciences and proposes an extension intended to support the operationalization of non-functional requirements (Section 3). Section 4 proposes another extension to deal with adaptation requirements that introduce the monitor-analyze-plan-execute functionality that characterizes adaptive/autonomic software systems. In Section 5 we note the fundamental distinction between functional and behavioural specifications and introduce a new form of operationalization that maps a goal into a set of behaviours. Finally, Section 6 concludes.

2 Goal Models

Figure 1 shows a simple example consisting of a goal model obtained through refinements and operationalizations for a meeting scheduling system with a single stakeholder goal ScheduleMtg. In the example, fulfillment of the root-level goal can come about by fulfilling three subgoals. In turn, each one of these is OR-refined into two alternatives.

The three OR-refinements define choice points in the design and are labelled $cp_1$, $cp_2$, $cp_3$ respectively. For example, the fulfillment of goal CollectTMtables might be done by assigning the responsibility to a person who manually collects them, or to the system-to-be. In
the former case, the responsible person has to execute function \( P\text{-Collect} \) with the system only providing database support. In the latter case, the system has to carry out \( S\text{-Collect} \), which may involve automatically generated email messages for all anticipated participants, and sending reminders where appropriate.

The fulfillment of goal \( \text{FindFreeRm} \) also has two alternatives. The first one is a domain assumption: we can fulfill \( \text{FindFreeRm} \) by simply assuming that there will always be free rooms for a requested meeting. Alternatively, we can fulfill the goal \( \text{GetRm} \) that locates a free room. Domain assumptions simplify the design problem by assuming that some of the sub-problems will be solved by actors in the system’s environment, so the designer doesn’t have to worry about them.

Goal refinement (AND/OR) transforms a goal into one or more simpler ones. Eventually, these goals need to be operationalized (“made operational”) either through the assignment of a function or the assignment of a domain assumption. In the former case, the designer is taking a proactive stance: “use this function to fulfill that goal”. In the latter, the stance is opportunistic: “something will happen to fulfill the goal, so we need not worry about it in our design”. Operationalizations relate functions/domain assumptions to the goals they operationalize. We call such operationalizations \textit{functional} to distinguish them from what will be proposed in latter sections.

Another kind of link in Figure 1 is a conflict link (marked by “\( X \)”), which says that two goals/tasks/domain assumptions are in conflict with each other, so they can’t be together part of a single solution. For instance, having a person collect timetables is in conflict with having the system choose a schedule for the meeting because not all collected timetables can be assumed to be in machine-readable form.

The goal model of Figure 1 suggests 6 possible solutions to the problem of scheduling meetings. The prefix of each element of the solution indicates whether it is a function (F) or a domain assumption (DA): \{F: P-Collect, DA:RoomsAv, F:P-Schedule\}, \{F: P-Collect, F:GetRm, F:P-Schedule\}, \{F: S-Collect, DA:RoomsAv, F:P-Schedule\}, \{F: S-Collect, DA:RoomsAv, F:S-Schedule\}, \{F: S-Collect, DA:RoomsAv, F:S-Schedule\}, \{F: S-Collect, DA:RoomsAv, F:P-Schedule\}. Such solutions are known as functional specifications.

A critical feature of goal models is that given a problem, e.g., fulfilling the goal \( \text{ScheduleMtg} \), they define a space of alternative solutions, rather than a single solution. In this respect goal models (and feature models used to specify product families) are unique among models used in Software Engineering.

### 3 Qualitative Operationalization

The practice of operationalization (operationalism) was first proposed in Physics by Percy Williams Bridgman in 1927 (Bridgman, 1927). In short, operationalism calls for scientists to define their concepts, however abstract and intangible, in measurable terms. For example, “mass” might be operationalized in a gravity-oriented way as affinity to gravity measured by a weight scale. It can also be operationalized as resistance to force. Operationalism was subsequently adopted by the Life and Social Sciences where it provides measurable definitions for concepts such as “health” and “free and fair judiciary”. For instance, “health” might be operationalized into a combined function of blood pressure, sugar level and number of drinks per day. “Free and fair judiciary” might be operationalized, on the other hand, in terms of the number of times there is government interference to judiciary functions, how often are members of the judiciary convicted of crime, etc.

For our purposes, operationalization of non-functional requirements amounts to adopting a precise measure by which an ill-defined non-functional requirement (“softgoal”) can be measured as to the degree of its satisfaction. This is consistent with RE practice, where non-functional requirements are supposed to be “metricized” in terms of a metric. For example, a performance non-functional requirement might be metricized as “System shall process 1,000 transactions per second”, and a usability requirement as “Users will be able to use the system after 3 hours of instruction”.

Figure 1: a simple goal model for the meeting scheduler
Figure 2 shows an extended version of the meeting scheduler goal model with two softgoals representing non-functional requirements LowCostSch and GoodQuaMtg. These are operationalized into quality constraints (“metrics”) ‘Each meeting scheduling will cost ≤ €20’ and ‘Each meeting will have > 70% participation’. We call this type of operationalization qualitative, to distinguish it from its functional cousin.

4 Operationalizing Adaptation Requirements

The increasing complexity of software-intensive systems, combined with the uncertainty of the environments wherein they operate has made adaptive software systems a popular topic for researchers and practitioners alike (De Lemos et al., 2013).

To design adaptive systems, we need not only vanilla functional and non-functional requirements, but also adaptation requirements, such as “Meeting scheduling should not fail more than 2% during any one month period”, or “If the collection of timetables fails because some participants did not respond, go ahead and schedule the meeting with the timetables you have”. Such requirements are operationalized by a feedback loop that monitors the performance of the system and takes action when the requirement is not being met, i.e., meeting scheduling fails more than 2% of the time, or timetables haven’t been collected from all participants.

Adaptation requirements come in two flavours. Awareness requirements (Souza et al., 2013a) impose constraints on the states (succeeded, failed, cancelled, etc.) of other requirements (i.e., goal model elements). For example, suppose that for the meeting scheduler three elements were found to be critical: ChooseSched, DA:RoomsAv and QC:70%Part. Further, each requirement has a different level of importance: ChooseSched should never fail, whereas we can tolerate one failure per week for DA:RoomsAv and would like a 75% success rate for QC:70%Part. These requirements for the monitoring component of the feedback loop are represented in Figure 3.

At runtime, the meeting scheduler should log changes of state of the instances of its goal model, e.g., “\text{T:S-Sched} has started”, “\text{T:S-Sched} has succeeded/failed”, etc. The feedback controller reads from this log, propagating the information up the model (following Boolean semantics of operationalization links), which may cause other elements to also change their state. These changes may eventually cause awareness requirements to fail, thereby triggering the system’s adaptation mechanism.

Evolution requirements (Souza et al., 2013b) constitute another kind of adaptation requirement. Such requirements specify changes to other requirements when certain conditions apply. Evolution requirements are defined as Event-Condition-Action rules, taking relevant events from the monitoring component of the feedback loop and applying adaptation actions to the managed system, depending on certain conditions.

Figure 3 shows three evolution requirements for the meeting scheduler example. The first one uses failures of AR5 as the triggering event and is associated with two possible actions: (a) have the system retry the function that caused the failure; or (b) reconfigure the system. The condition for (a) is that it can only be applied once, whereas the condition for (b) is that (a) has been attempted but has failed to solve the problem.

The above example illustrates two kinds of adaptation action: evolution and reconfiguration. Evolution is used when stakeholders know exactly what the system should do in order to adapt. In this case, adaptation is defined as a series of modifications to the goal model, evolving it to represent a new problem space. Such modifications can take place at the instance level, which changes the system for a single user session (e.g., the retry case illustrated for ChooseSched), or at the requirement/class level, which changes the system from that point on (e.g., if requirement R fails more than we can tolerate, replace it with a less strict version R’-).

Reconfiguration, on the other hand, can be applied when stakeholders do not have a specific solution to the problem, but would like the system itself to search in its solution space for alternative solutions (or specifications, as explained in Section 2). The three choice points of
Figure 1 (cp1, cp2, cp3) allow the feedback controller to try different ways of satisfying system goals, therefore adapting to specific situations.

Control variables can also be identified as means for system reconfiguration. Figure 3 shows variable FhM connected to goal CollectTMtables. It prescribes from how many participants one should collect timetables before moving on to schedule a meeting. It should be clear that changing the value of this variable could affect the satisfaction not only of goal CollectTMtables, but also of other requirements (e.g., the less timetables you collect, the higher the chance of poor quality meetings as participants find they can’t attend a meeting they are supposed to). The relation between changes in these parameters (i.e., choice points and control variables) and the effect in satisfaction of requirements should also be elicited in order for the adaptation component of the feedback loop to be able to use this information properly.

Different reconfiguration algorithms have been proposed in the literature, each requiring different information to be included in a requirements model. (Dalpiaz et al., 2012) and (Souza et al., 2012) are two such examples from our own work.

5 Behavioural operationalization

Specifying a function through which a goal is operationalized is one way to move from the problem to the solution space, but there are also others. Behavioural specifications constitute one such alternative. Behavioural operationalizations define the possible behaviours of a system as the set of allowable sequences of executions of its functions (Dalpiaz et al., 2013).

Behavioural and functional operationalizations are complementary. Functional operationalization is applied to every leaf goal of a goal model and it defines the function through it can be fulfilled. Behavioural operationalization, on the other hand, applies to non-leaf goals and specifies in what order subgoals are to be fulfilled. For example, if goal G is AND-refined into subgoals G1, G2, a possible behavioural operationalization is ‘G1 ; G2’, meaning that G1 must be fulfilled first, followed by G2. Alternatively, we may specify ‘G1 | G2’, exactly one of the two subgoals needs to be fulfilled. More generally, we can use regular expressions of subgoals for behavioural operationalization. For instance, we may want to say that in order to schedule a meeting, we need to collect timetables one or more times until all timetables have been collected, then proceed with the scheduling:

CollectTMtables+ ; Schedule

Here ‘+’ stands for Kleene closure of regular expressions, while ‘;’ indicates temporal ordering.

Behavioural expressions are actually more than regular expressions since we sometimes want to specify that two subgoals need to be fulfilled concurrently. This is indicated by the shuffle operator ‘#’. For example in Figure 4, meeting scheduling (G1) is operationalized with ‘G2; (G3 # G4)’, indicating that CollectTMtables must be fulfilled first, followed by the interleaved fulfillment of G3 and G4. Likewise, the goal for collecting timetables (G2) is annotated with ‘(G4 | G5)#’, indicating that 1 or more versions of G4 and G5 can be fulfilled concurrently.

Functional operationalization tells us how to fulfill a leaf goal in terms of a function. On the other hand, behavioural operationalization tells us how to fulfill a non-leaf goal by using together solutions for its subgoals.

6 Discussion

Operationalization is about making something operational, either by providing a function that defines its operation, or by making it measurable. For goal models, these forms of operationalization cover the two basic types of requirements: functional and non-functional. In this paper we have examined other forms of operationalization that account for adaptation requirements and non-leaf goals.
These extensions reflect different stances that a designer can take to the fulfillment of a requirement. A proactive stance amounts to providing the means to fulfillment through a function. An opportunistic stance assumes the problem away. A scientific stance delivers the means of measuring its degree of fulfillment. Finally, a reactive stance amounts to offering a mechanism (feedback loop) to cope with failures.

In conclusion, operationalization is a rich concept that has been in use in the Sciences for almost a century. It manifests itself in different ways for different classes of requirements. More importantly perhaps, it reflects multiple strategies to problem solving and design that go far beyond what has been explored and deployed in Requirements Engineering.

Acknowledgements
This work was supported in part by ERC advanced grant 267856, titled “Lucretius: Foundations for Software Evolution”, http://www.lucretius.eu.

References


INVITED PAPER
The Role of Software Models in Developing New Software Systems;  
A Case Study in Project Management

Koichiro Ochimizu  
Vice President of  
Japan Advanced Institute of Science and Technology (JAIST)  
Research Professor and Director of  
Center for Highly Dependable Embedded Systems Technology, JAIST  
1-1 Asahidai, Nomi, Ishikawa, Japan  
ochimizu@jaist.ac.jp

Abstract

The collaboration process in a joint project with a Japanese company is introduced. Topics included are the role of software models at each phase of the joint project and the skill set model of the collaboration.

We have developed a software project scheduling tool suited to the Japanese work environment.

Through the joint project, models play a key role as an effective communication media between Academia and Industry. They are useful for:
- Problem-discovery and problem-setting;
- Abstraction of the real world to extract necessary information, organizing this data into a useful information structure;
- Construction of a mathematical model to assure the generalizability of the theory and the tool;
- Visualization of the organizational resource usage state and definitions of several metrics.

Keywords: Joint Project with Company; Project Scheduling Tool; Load-Capacity Model; Resource Binding; Integer Linear Programming (ILP).

1 Introduction

Many activities for incremental technological innovations are now performed in Japanese companies. For the success of a new technological innovation, collaboration between industry and academia is very important. In this paper, I will address the role of models as communication media between academia and industrial people in the joint project mentioned below.

1.1 Technical Results

Embedded software for digital electrical appliances development projects in recent years are planned based on appropriate release time.

However, most projects require long-term development. Therefore, two or more projects are implemented concurrently, as shown in Fig. 1.

Fig.1 Multiple Projects Problem

To evaluate the feasibility of a new software development project in a situation in which two or more projects are being implemented concurrently, it is necessary to consider not only software development workload but also residual capacity of an organization.

We have developed the Load-Capacity Model for verifying feasibility at an early stage of the software development project. We have developed a method for simultaneous project scheduling, and a resource-binding based on the Load-Capacity Model. This method is formulated as Integer Linear Programming, using Load-graph and Capacity-graph, which represent workload and organizational capacity, respectively, as inputs. Outputs of this method are Gantt Charts and Load-Capacity Diagrams. The load-capacity diagram shows the state of an organization after workload has been assigned. Experimental results on realistic examples show that the method provided useful solutions in a practical time period. (Saito, Kusanagi, Ochimizu 2011), (Kusanagi, Saito, Ochimizu 2011), (Saito, Kusanagi, Ochimizu 2012).

1.2 Skill Set Model of Collaboration

One effective method to promote research on software engineering is joint work with industry, which enables us to recognize and solve problems in the real world. Through several case studies we performed, including this case, four types of key persons are needed for the success of collaborations.

(1) Key person 1 (from industry)
A person who is highly motivated for problem-discovery and problem-solving, and knows the real world very well.

(2) Key person 2 (from academia)
A person who has expertise in software engineering, and who knows the various kinds of research outcomes in software engineering. Key person 2 should try to discover the real problem together with key person 1 from superficial problem statements. Key person 2 is also responsible for proposing several candidates for a solution, based on his/her knowledge and experience. Key person 1 should be able to evaluate the effectiveness of the proposed solutions.

(3) Key person 3 (expert on mathematics)
A person who can propose and formulate mathematical models for the solution. Mathematical model is important to assure generalizability of the problem and the solution.

(4) Key person 4
A person who can develop a prototype tool based on the conceptual models and the mathematical models adopted. A tool is very important for field testing and then for familiarizing the new result in industry.

(5) Key person 1 again
Key person 1 should have authority to perform several field tests using a tool, with the permission and help of people in the field.

2 Problem-Discovery and Problem-Setting
Both key person 1 and key person 2 should discover a real problem to be solved, from among the superficial problem statements in the field.

2.1 Original Proposal from Industry
Fig.2 shows the original proposal from key person 1 which represents the structure of the organization with problem statements.

Fig. 2 From Single Project to Matrix Structure
New product development starts from a simple project, as shown at the top of Fig. 2. The more products have been sold, the more derivatives appear. And a new project is initiated for each derivative. So, the structure of the organization changes from the single project structure to the matrix structure, as shown at the bottom of Fig.2.

There are several functional teams which supply experts in operating systems, drivers, middle-ware and GUI to several projects. When we start to develop a new product, a project leader needs to manage a simple project. In a matrix structure, each project needs its own project leader.

A matrix structure is just a direct abstraction of an organizational structure, and it is an informal model. The informal model is useful for problem-discovery if it is a good abstraction and reflection of the real world. We could observe and discuss several problematic situations based on this structure.

Time must be spent at this stage because problem-discovery and problem-setting are the most important tasks for the success. We must consider what the problem is very carefully. There are several candidates as the problem. The following information shown in Table 1 was given from key person 1 to key person 2, to share knowledge of the current status related to concurrent execution of multiple projects. There are three types of stakeholders. Those are a senior manager, a project leader, and a functional team leader.

Table 1 Role and Problem Statement of each Stakeholder (Kusanagi 2012)

<table>
<thead>
<tr>
<th>Role</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senior Manager</td>
<td>- should manage all of the products</td>
</tr>
<tr>
<td></td>
<td>- should prioritize both projects and functional tasks to release all of</td>
</tr>
<tr>
<td></td>
<td>the derivatives by due date</td>
</tr>
<tr>
<td>Project Leader</td>
<td>- should achieve QCD goals of his last project</td>
</tr>
<tr>
<td></td>
<td>- worry about whether each functional team can supply sufficient human</td>
</tr>
<tr>
<td></td>
<td>resource to his project to meet his QCD goals</td>
</tr>
<tr>
<td>Functional</td>
<td>- is responsible to supply proper human resources to multiple projects</td>
</tr>
<tr>
<td>Team Leader</td>
<td>- should allow proper time for team members to improve their skills</td>
</tr>
<tr>
<td></td>
<td>- should accumulate know-how</td>
</tr>
</tbody>
</table>

There are several superficial problems; process-related ones, product-related ones, and human-resource-related ones. In our collaboration, we agreed the human-resource-related problem is the most influential one.

2.2 Our Solution
Through discussion, we reached the following conclusion, as shown in Fig.3.

A project scheduling method should take into account both work load of a project and the residual capacity of an organization.
Now we need to develop a theory and models which can deal with resource management of functional teams for a given workload. We proposed the Load-Capacity Model, which contains necessary information to evaluate feasibility of a new project.

2.3 Abstraction of the real world to extract necessary information and to organize this data into a useful information structure

Next we tried to extract necessary information for both work load and capacity. Fig. 4 shows the extracted information.

![Load-Capacity Model](image)

**Fig. 4 Load-Capacity Model (Kusanagi 2012)**

The Load-Capacity Model shown in Fig. 4 consists of five components: Architecture Type, WBS Type, WBS Instance, Project Team and Organization (Regular Organization and Outsource Organization). Architecture Type, WBS Type and WBS Instance are basic components of the load structure. Regular Organization and Outsource Organization are basic components of the capacity structure. Teams of a new project need to satisfy the requirements of both the load structure and the capacity structure (Saito, Kusanagi, and Ochimizu 2011, 2012).

1) **Architecture Type**: “Architecture type” means the structural aspects of a system architecture, typically it consists of GUI, application, middleware, driver, and operating system. The attributes of an architecture type are dependency relations and communications. In general, functional teams are organized according to the architecture of the system. Relationships among components of the architecture require communications between the developers. These communications are one of the load factors.

2) **WBS Type**: WBS (Work Breakdown Structure) type is a set of basic work units, named Work Packages (WPs) usually organized as a tree structure. There are two WBS types, WBS type for new developments and WBS type for derivative developments. WBS types are usually defined by the Software Engineering Process Group (SEPG) of each organization. The attributes of a WBS type are the load unit of each WP (WP load unit) and the process. WP load unit of each WP becomes the standard of the estimation. The process is a set of one or more WPs, and WBS type includes processes as subsets.

3) **WBS Instance**: The WBS instance is a set of WPs instantiated and tailored from the WBS type, and it corresponds to all WPs that are actually executed by the project. The attributes of a WBS instance are the total load, the standard estimated man-hours (SEH), the required skills, and the precedence constraints. In the Load-Capacity Model, the total load is the sum of the SEH of each WP and the communication overhead. However, the communication overhead does not appear easily in the WBS instance. SEH is derived from WP load unit. There are precedence constraints between WPs, which are derived from the dependency relations and the processes.

The skill given to each WP as an attribute is called the required skill, and a skill of a resource is called “own” skill. We use θ as the symbol of skill. Examples of skills are business knowledge in requirement definition, design methodology in system design, and programming language in software implementation. “skill” in this model is different from ability. In our model, we temporarily deal with skill as a classification of skill. We can assign WP to the resource if the required skill corresponds to “own” skill.

4) **Project Team**: A project team is a set of people involved in product development. The attributes of a project team are the consumption of resources, the distribution of the load, and the communication overhead. Resource consumption is the assignment of the load (WP) to the resources based on the relation between the delivery date and the skill.

5) **Organization**: The organization is a set of developers. The organization consists of a regular organization and outsource organizations. Each organization has two or more subsets, and we call each subset a functional team. An organization offers members of functional teams to each project as resources to accomplish the project. The attributes of a regular organization are resources, capacity and skill. The attributes of an outsource organization are cost, quality and skill. In the Load-Capacity Model, the resources are classified according to the number of skills. In our model, a resource which has two or more skills is defined as a highly-skilled resource.

3 Formulation of a mathematical model to assure the generalizability of the theory and the tool

Mathematical model of the solution is necessary to assure the generalizability of the problem and the solution.
We formulated our resource-binding method as Integer Linear Programming. Fig. 5 shows an outline of our model.

3.1 Load Graph
There are two inputs of our algorithm. One is the Load Graph, which is a graph representation of the upper part of Fig. 4.

A WBS instance showing all precedence constraints is called a load-graph. An example is shown in Fig. 6 (Saito, Kusanagi, and Ochimizu 2011).

A Load Graph is a directed acyclic graph. Each vertex represents a WP, and the directed edge represents the sequencing relation between WPs. It is assumed that SEH (number of time slots) and a required skill are given to each WP as attributes.

3.2 Capacity Graph
Another input is a Capacity Graph, which is a graph representation of the lower part of Fig. 4. The capacity-graph is derived from the capacity structure of the Load-Capacity Model. Figure 7 shows an example of a capacity-graph (Saito, Kusanagi, and Ochimizu 2011).

The capacity-graph shows the hierarchical structure of the organization. In a capacity graph, a vertex represents a resource (human) and a role, and edges represent relations in the organization. A vertex (resource) in the capacity-graph has one or more “own” skills. The residual capacity of resource \( r \) at time \( t \) is shown by a capacity function \( f(r, t) \in \{0, 1\} \) where value 1 means WP is assigned. This team’s “own” skill is shown in the table in Fig. 7.

3.3 Load-Capacity Diagram
The load-capacity diagram shows the state of an organization after workload has been assigned. The idea is to embed the load-graph into the capacity-graph satisfying the constraints (Saito, Kusanagi, and Ochimizu 2011).

Each row of the Load-Capacity Graph corresponds to a member of a project, representing who has time resources, represented by a sequence of time slots. In this example, we assume a project consists of only middleware team members.

3.4 Gantt chart
Gantt chart is added as one of the outputs, because key person 1 advised us that it is necessary to prepare user-familiar output. Fig. 9 shows an example of a Gantt chart produced by our system.

3.5 Resource-Binding algorithm
We adopted Integer Linear Programming as a resource binding algorithm. The reasons why we adopted the ILP are mentioned below.

(1) Various conditions of the project can be formulated as the constraints of ILP.
(2) Two or more schedules can easily be generated by changing the objective function and the constraints.
The minimum resources or the minimum development period are obtained by an optimal solution under the given constraints.

The solution is obtained faster than in the past, by improvement of the performance of the ILP-solver and calculator.

Table 2 shows examples of constraint representation, and Table 3 shows an example of an objective function which represents the resource-allocation policy.

**Table 2 Decision Variables and Constraints (Saito 2011)**

- Binary decision variable: $z_{ij} \in \{0,1\}$
  - Only when WP is bound to resource $r$ and starts in slot $i$
- Constraints:
  1. Each WP must be implemented at least once
     \[ \sum_{i} x_{ij} = 1 \]
  2. The sequencing relation must be satisfied
     \[ \sum_{j} \frac{1}{c_{ij}} \geq \sum_{j} \frac{1}{c_{ij} + h} \]
  3. WPs bound to the same resource must not be concurrent
     \[ \sum_{i} x_{ij} \leq 1 \]
  4. Remaining capacity of each resource must be considered
     \[ \sum_{j} x_{ij} \geq \sum_{j} \frac{1}{c_{ij} + h} \]

**Table 3 Objective Function (Saito 2011)**

- We can consider various objective functions, which express the policies adopted in the project.
- Example: Minimize the development period and the number of WPs to be assigned to Team Leader and Member2

\[
\min \left( \sum_{i} \sum_{j} z_{ij}^{\text{ALLOC}} + \sum_{i} \sum_{j} z_{ij}^{\text{CTOR}} \right)
\]

4 **Visualization of resource allocation state by “pieces of ice in a cup” metaphor**

We adopted “pieces of ice in a cup” metaphor to visualize the resource-binding result, and to develop metrics.

If we compare the capacity of an organization to a cup, and the load of a project to a piece of ice, scheduling can be compared to putting several pieces of ice into the cup. The shape of a piece of ice depends on the project management policy (Fig. 10).

Fig.11, Fig.12, and Fig.13 show examples of the scheduling result of a third project, based on different work-load-allocation policies. All of the results used real data collected in field testing.

**Fig.11 “Smoothing of work load” policy (Kusanagi 2012)**

**Fig.12 “Use only skilled people” policy (Kusanagi 2012)**

**Fig.13 “Uneven distribution of work load” policy (Kusanagi 2012)**

5 **Developing the Metrics**

Fig. 14 helps to visualize the resource allocation. We can define several metrics based on the presentation of
Fig. 14. Table 4 shows the requirement for the metrics and the metrics defined.

Table 4 Usage of Metrics (Kusanagi 2012)

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senior Manager</td>
<td>Manager both multiple projects and all of the products</td>
<td>Judge feasibility of all projects. Understand the status of resources of each functional team</td>
</tr>
<tr>
<td>Project Manager (Leader)</td>
<td>Achieve QCD goals for a specific project</td>
<td>Judge feasibility of his/her project at the early stage of project planning.</td>
</tr>
<tr>
<td>Functional Team Leader</td>
<td>Is responsible to supply proper human resources for multiple projects simultaneously - Should allow proper time for team members to improve skills</td>
<td>- Balancing of several policies - Judge the feasibility of a new project from the viewpoint of resource supplier - Prepare for the next project - Packing Rate = Load of the new project at the present period - Residual capacity at the present period - Margin for next project = residual capacity at the extended period = total organizational capacity at the extended period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliance rate to the Deadline - Estimated Time - Given time resources</td>
</tr>
</tbody>
</table>

7 Conclusion

We created many informal diagrams through the joint project. All of them played an important role at each phase of the collaboration.

1. The abstract depiction is a trigger of discussion, and was useful for sharing knowledge about situations in the field.

2. The abstract depiction is also useful for extracting the necessary information from the field.

3. Mathematical model is necessary to develop the tool

4. Visualization of computing results is also necessary for practical usage of the method and the tool in the field.

Without the informal models created through our collaboration, we could not communicate with each other to achieve valuable consensus and agreement.

8 Acknowledgement

I will express my best thanks to my partners in this joint project, Dr. Takumi Kusanagi who works for Toshiba played a very important role in this joint project as key person 1. Dr. Akinori Saito, who was a PhD candidate during this joint project, and played a very important role too as key person 3 and key person 4.

9 References


CONTRIBUTED PAPERS
CARE – A Constraint-Based Approach for Re-Establishing Conformance-Relationships

Johannes Schoenboeck 1
Elisabeth Kapsammer 2
Wieland Schwinger 2
Juergen Etzlstorfer 2
Martin Wischenbart 3

1 University of Applied Sciences Upper Austria, Campus Hagenberg, Austria
2 Johannes Kepler University Linz, Austria
3 Vienna University of Technology, Austria

Abstract

In Model-Driven Engineering, models have to conform to their associated linguistic and ontological metamodels. While linguistic metamodels are usually not subject to frequent changes, ontological metamodels are. Thus, ontological conformance may be easily corrupted by changes on the metamodel or by the incantuous modification of models. Current approaches for re-establishing conformance relationships are often (i) deeply woven into specific tools to record changes and to derive resolutions, or (ii) require extensive user effort to guide the resolution process, and (iii) the output of these approaches usually is one single solution, whereas alternative solutions remain unexplored. To allow for exploring a broader solution space independent of specific tools and to avoid extensive user involvement by utilizing predefined repair actions, we propose a logic programming approach called CARE, for accomplishing multiple solutions. In particular, CARE is based on a formalization of the ontological conformance relationship as constraints, accompanying repair actions for counteracting constraint violations, as well as quality criteria for ranking the solutions. This paper reports on the realization of CARE based on Answer Set Programming and summarizes lessons learned from applying the approach in several experiments.

1 Introduction

Model-Driven Engineering (MDE) [27] proposes a continuous use of models to conduct the different phases of software development. Models have to conform to their associated linguistic and ontological metamodels that are prevalent in a certain domain and which define concepts, their relationships, as well as constraints among each other. Consequently, the conformance between models and their associated metamodels can be classified into ontological conformance, based on the meaning (e.g., an object Mickey is an ontological instance of a class Mouse), and linguistic conformance, regarding their syntactical form (e.g.,

Mickey is an linguistic instance of Object), as shown in Figure 1. In this context, ontological conformance is concerned with the domain definition, while linguistic conformance considers language definition [2].

While linguistic metamodels, such as Ecore [1], are often standardized and changed seldomly, ontological metamodels, representing concepts within a certain domain, are frequently subject to change [13]. This refers to the typical case of metamodel evolution, which entails the co-evolution of dependent artifacts in order to retain conformance [6, 8, 29]. Further, in data integration scenarios, conformance of existing models to a new metamodel may be disrupted [3], and, thus, has to be re-established. Moreover, the incantuous modifications of models may also violate ontological conformance, while keeping the models syntactically correct, thereby maintaining linguistic conformance. In either case, model processing is obstructed in current tools, until conformance between models and their ontological metamodels is re-established, which is the focus of this paper.

Since the manual re-establishment of conformance is tedious and error-prone, dedicated (semi-) automatic approaches exist, which are, however, (i) often tightly coupled to specific tools to record changes and to derive repair actions, or (ii) require extensive user effort in the resolution process, e.g., by demanding to specify the resolution steps beforehand, or by guiding the tool step-by-step. If changes cannot be tracked or derived due to the absence of the prior metamodel version, it is of special interest to automatically re-establish conformance without relying on manually provided input. Finally, representatives of both kinds of approaches mostly (iii) provide a single solution, only, disregard alternative solutions with respect to their qualitative properties.


1Ecore is the realization of MOF [23] in the Eclipse Modeling Framework (EMF) http://www.eclipse.org/modeling/emf
To overcome these shortcomings, we propose a framework for Constraint-based REmapping of ontological conformance relationships (CARE), which is (i) independent of a specific tool by means of a stand-alone framework instead of being deeply woven into an existing modeling environment, (ii) utilizes predefined repair actions, and, thus, requires little user involvement for specification as well as guidance of the resolution process, and is (iii) capable of generating multiple, ranked solutions, which fit a set of quality criteria that should be naturally fulfilled by repair solutions, such as the preservation of information capacity.

Operating in three phases, CARE is able to re-establish ontological conformance, independent of whether changes in the model or in the metamodel caused the violation. Therefore, CARE is based on the core concepts of Ecore, having classes, attributes, and references with respective objects, values, and links. In phase 1, constraint violations are detected based on a formalization of the ontological conformance relationship in the form of dedicated constraints in logic programming, in phase 2, constraint violations are repaired by means of repair actions, generating multiple ontological conforming solutions, and in phase 3, ranking and selection of the best solutions is achieved by applying quality criteria, which comprise structural and semantic knowledge about the potential solution space. Furthermore, prove-conformance information and statistics are collected during all phases and reported to the user for inspecting the solutions.

Before the three phases of the CARE approach are presented in detail in Section 3, Section 2 introduces a motivating example. Section 4 demonstrates a proof-of-concept prototype based on Answer Set Programming, while lessons learned from several experiments are presented in Section 5. Finally, related work is discussed in Section 6 before Section 7 concludes.

2 Motivation

This section introduces a motivating example and thereupon outlines different ways to deal with the challenges to automatically re-establish ontological conformance based on different kinds of knowledge. Figure 2 shows a simple example, basing on UML state machines, with a model \( M \) comprising two State objects, whose conformance gets violated due to metamodel evolution comprising three steps: extract subclass for State resulting in SimpleState and FinalState as well as make class abstract for State\(^2\). These steps, however, do not need to be known for re-establishing conformance of \( M \) to the evolved metamodel \( M' \), since CARE is capable of re-establishing conformance regardless of the type of changes and whether changes in the metamodel or in the model caused the violation. Nonetheless, linguistic conformance must not be disrupted, i.e., both \( M \) and \( M' \) must be linguistic instances of Objects and Classes, respectively.

**Solution Space.** To re-establish conformance in the example, non-conforming objects (cf. \( s_1 \) and \( s_2 \) in Fig. 2) may either be retyped (reclassified as instances of the concrete classes SimpleState or FinalState) or deleted. By applying these options, a total of nine solutions with respect to model-metamodel constraints arises, so far disregarding any constraints on the metamodel. Thus, the potential solution space for retyping or deleting non-conforming elements contains \((c + 1)^3\) solutions (with \( c \) = number of candidate classes + 1 for deletion, \( o \) = number of non-conforming objects).

**Selection.** These solutions, however, need not to be valid, since metamodel-specific constraints may be violated. Consequently, for selecting valid (i.e., conforming) solutions from the solution space, metamodel-specific constraints, e.g., specified in the Object Constraint Language (OCL), and user-provided constraints, such as mapping instructions, must be validated. Thus, given the OCL constraint in

\[ \text{OCL constraint on } MM' \]

\[ \text{context } \text{FinalState: } 
\text{inv: } \text{self.successorCount} = 0 \]

Note, that the classification of evolution steps is based on [12] and [15].
Figure 3: Classification of Available Knowledge in Correlation to Solution Space

Figure 2, the solutions $M', M', M'$, and $M'$ are invalidated.

**Ranking.** The remaining set of valid solutions may further be ranked, to provide the user with ordered solutions pushing forward those solutions fulfilling dedicated qualitative criteria. For this, heuristics incorporating structural and semantic knowledge may be employed. Regarding structural knowledge, the structural similarity between the objects in $M$ and $M'$ as well as between $M$ and $MM'$ may be employed, by using methods which are similar to existing schema and ontology matching techniques [7, 26]. In contrast, matching has to be done across meta-modeling layers. Furthermore, in the context of ranking, the prevention of information loss (indicated by shades in Fig. 2) may be considered essential. Exploiting this structural knowledge, $M'1$ scores best, preserving all attributes, and $M'9$ is ranked last, since all model elements are deleted. Additionally, semantic knowledge obtained from an external knowledge base (e.g., an ontology) or provided by the user in terms of domain knowledge, may be used for ranking. Given the knowledge that the value “final” of attribute label rather corresponds to FinalState than to SimpleState, retyping of $s2$ to FinalState is preferred, thus, favoring $M'2$ over $M'1$, as indicated in Figure 2. Consequently, the ranking of valid solutions depends on a particular configuration, which may favor different quality criteria. Although CARE provides a default configuration, e.g., trying to prevent information loss, users may overrule it by providing a custom configuration.

To summarize, Figure 3 gives a classification of the sources of knowledge, comprising constraints for selection of valid solutions, and heuristics to be utilized for ranking of solutions. Given these sources of knowledge, the CARE framework may be divided into a generic core part, comprising conformance constraints as well as structural knowledge and an optional extensions part, which is domain specific. The extensions part may be divided into constraints, comprising metamodel-specific ones (e.g., specified in OCL) and user-provided ones (e.g., mapping instructions), and semantic knowledge provided by users or external knowledge bases, as well as the user configuration. Consequently, although the CARE framework may provide multiple valid solutions by relying on the generic core part, only, the more information is available for the extensions part, the better the solution space may be narrowed, as indicated in Figure 3. To achieve the goal of re-establishing conformance, CARE operates in three phases, which are described in detail in the next section, together with an overview of the approach.

**3 CARE Process: Detection, Repairing, Ranking**

The main idea of CARE is to re-establish conformance between non-conforming models and their metamodels by utilizing ontological conformance relationships, formalized as constraints between a model and its metamodel. Based on corresponding repair actions, CARE generates multiple ranked solutions facilitated by Answer Set Programming (ASP) [18]. To achieve this goal, CARE operates in three phases as illustrated in Figure 4.

As a prerequisite, the non-conforming model, as well as the metamodel together with its constraints are transformed into ASP. For that, CARE is based on the core concepts of metamodeling, consisting of classes, attributes, and references, i.e., a representative subset of Ecore focusing on structure, not considering operations 3. Ecore, however, may be replaced by other modeling languages, by implementing the appropriate transformations and by modifying the conformance constraints, repair actions and ranking rules.

In the first phase, violations are detected (cf. 1 in Fig. 4) by means of ASP rules which employ built-in conformance constraints (e.g., objects have to be instances of non-abstract classes), as well as optional metamodel-provided and user-provided constraints. Second, during model repairing (cf. 2 in Fig. 4), all possible solutions that re-establish ontological conformance are calculated by means of corresponding repair rules. In order to rank the resulting models in a third phase (cf. 3 in Fig. 4), built-in heuristics (e.g., based on structural similarity) together with external semantic knowledge (e.g., for semantic matching) may be employed in so-called ranking rules. This means that for repairing and ranking a “guess, check & optimize” methodology [10] is applied, which allows non-optimal solutions to be discarded early during their computation, and thereby allows to speed up the whole process. Since all three phases operate on the same knowledge base of logic programming facts and rules, CARE enriches solutions with meta information by using intermediate results to generate provenance information and statistics, which, ultimately, support the user in inspecting differences between solutions after the fulfilled transformation back
3.1 Phase 1: Detection of Violations

For the detection of conformance violations between a model and its metamodel, CARE is based on formalized constraints, i.e., checking conditions for conformance on all model elements. In general, conformance is accomplished, if each model element can be syntactically regarded as a valid instance of a type in the metamodel [17]. Applied to the core concepts of Ecore, this means that model elements, namely objects with features (values and links), have to conform to respective classes with class features (attributes and references) in a metamodel. Therefore, we derived a more specific definition, formulated as a set of object, value, and link constraints, as listed in Table 1. Thus, object constraints are denoted with ocX, value constraints with vcX and link constraints with lcX, respectively, whereby the X represents an ID. For instance, for the type of an object a corresponding class must exist (cf. oc1), which must not be abstract (cf. oc2, depending on oc1). Besides such generic constraints provided by the CARE core, constraints on the metamodel in terms of OCL, as well as user-provided ones in terms of mapping instructions may be considered to validate conformance of model elements (cf. CARE extensions, which are currently specified manually). An example for an OCL constraint was shown in the introductory state machine example (cf. Fig. 2), defining that a FinalState must not have successors, i.e., successorCount must be 0.

To check whether all conformance constraints are fulfilled, they are formulated as separate interdependent logic programming rules. A constraint solver evaluates these rules to produce the output, i.e., facts describing which objects, values, and links conform or not, and adds these new facts to the CARE knowledge base. Examples for such evaluations are shown as implication rules in Table 2, using a pseudocode language in the style of ASP, with conjunction denoted by \( \land \) (cf. Fig. 5 in Sect. 4 for corresponding ASP code). The above mentioned object constraint oc2 is expressed in the form of two rules, shown in lines 3-4, stating that an object with a corresponding, non-abstract class fulfills oc2, whereas an object not fulfilling the rules for oc2 (which also requires oc1) violates the constraint (cf. oc2violated(oc1)). In line 9, the above mentioned OCL constraint is specified formally, which is currently achieved manually by the CARE user, but foreseen to be automated and subject to future work. Note, that object\((\text{OID}, \text{CID})\) denotes that an object with ID \(\text{OID}\) is typed to a class with ID \(\text{CID}\).

Based on these detection rules, for the introductory example the resulting violations are oc2violated(s1) and oc2violated(s2), since both State s1 and State s2 are typed to an abstract class in \(M'M'\). These detected violations are then used as inputs for repair actions in the next phase.

3.2 Phase 2: Model Repairing

Based on the violations from the first phase, a set of rules allows ontological conformance between a model and its metamodel to be re-established. For each kind of violation, CARE uses dedicated repair actions, which are systematically derived from the constraint violations listed in Table 1 and, thus, predefined within the CARE framework. Basically, the options to repair violations are to modify or to delete the violating model element. For instance, an object, which violates oc2 can either be retyped to another class (i.e., the type reference from the object to its class is modified, cf. ra1), or the object can be deleted (cf. ra2). Such alternative repair actions are denoted as rows in Table 1. Table 2 shows the examples discussed above in pseudocode. A disjunction for rules is expressed by using \(\lor\), thereby producing all possible different solutions, i.e., retyping and deletion. Line 5 in Table 2 states, that if oc2 is violated, either repair action oc2ra1 (retyping object) or oc2ra2 (delete object) has to be applied. For producing output elements, additional facts must be added to the knowledge base, which are denoted with suffix \(x\). For the deletion of elements, no further action is required in oc2ra2 (i.e., no object is added), since elements with suffix \(x\) are regarded as output elements, only. Consequently, the suffix \(x\) is also applied to all valid elements to declare them as output elements. Note, that on the left hand side of the rule in line 6 of Table 2 there is no binding of the object’s ID to a specific class ID. Thus, the object is retyped to all possible non-abstract classes, and consequently, all possible combinations of retyping are generated. However, since an object can only be the instance of one class at a time, another rule must prevent these solutions from being generated (cf. PREVENT rule in line 7 of Table 2).

For the example, this means that the two objects which violate oc2 (s1 and s2), can either be deleted, retyped as SimpleState, or retyped as FinalState. By applying additional metamodel constraints by means of OCL constraints (cf. line 10 in Table 2), invalid
they may incorporate specific facts about the domain. For that, as discussed in Section 2, CARE distinguishes between structural and semantic knowledge. Rules based on structural knowledge, for instance, may use matching heuristics, or determine information capacity of solutions, based on inputs, outputs, and intermediate results, and, therefore, assign costs for structural difference or information loss. Additionally, semantic knowledge from external knowledge bases, such as domain ontologies or linked data, or provided by users, may be used for such heuristics as well. Another interesting criterion in the course of CARE is whether a heuristic employs single model elements, whole objects including features, or the complete model, i.e., the granularity. For instance, computation of similarity between input and output models may base on matching of single objects, values, and links by ID, on the comparison of the number of features per class, or on the usage of characteristics of the complete model, such as connectivity between objects. A more detailed classification of this knowledge, however, would go beyond the scope of this paper. Consequently, different kinds of knowledge allow for a variable configuration of the ranking phase, firstly by adding or removing rules and facts, and secondly, by adjusting cost parameters.

Again, Table 2 shows a simple example of such heuristics, as used in the introductory example. The rule in line 8 restrains information loss, by assigning costs to solutions where repair action oc2ra2 (delete another artifact) is taken. As additionally specified, line 11 assigns costs for objects with label “final” that are not retyped to FinalState. Thus, for the example, this means that $M_2$ is the best solution (cost = 0), and therefore ranked first. $M_1$ is ranked second, since s2 with label “final” was retyped to SimpleState (cost = 1). Concerning the remaining four, $M_5$ is ranked last, since both objects were deleted (cost = 4).

As a result, CARE provides a list of ranked solutions, which are transformed back to Eco. To support the user in inspecting and choosing solutions, additional meta information in terms of provenance information and statistics is provided as annotations using the dedicated Open Provenance Model (OPM) [21] vocabulary. Thus, a model element in $M'$ is an artifact in OPM, which is derived from another artifact in $M$, and which is used in processes, i.e., CARE.

Table 2: Pseudocode for Selected Conformance and Metamodel Constraints
repair rules. This information may be derived directly from intermediate results, whereas statistics, in contrast, require additional rules. For instance, the retyping of $s_1$ in $M_2$ is expressed as $\text{wasDerivedFrom}(s_1, \text{SimpleState})$. For counting how many states were retyped to $\text{SimpleState}$, however, an additional computation step is required, which may also be performed on the transformed Ecore model. Ultimately, this meta-information may be presented to the user to facilitate the inspection of the ranked solutions.

4 Proof-of-Concept Prototype

After having discussed the three phases of CARE, this section presents the prototype for re-establishing ontological conformance between Ecore-based models and metamodels, implemented with ASP.

The CARE proof-of-concept prototype supports the core concepts of Ecore, as discussed in the previous section, as well as OCL constraints. All these inputs may be represented as logical axioms in ASP, thereby resulting in a similar representation to those presented in [5, 25]. Consequently, transformations between Ecore/OCL and ASP are required, which are already automated for Ecore using Xtend\(^4\), and therefore to be automated for OCL by utilizing methods presented in [20] and [25]. Rules and constraints for all three phases are implemented in ASP, which is a form of declarative programming language, based on the stable model semantics of logic programming [18]. An ASP program is a finite set of rules, consisting of a head (on the left) and a body part (on the right). In CARE, such rules are used for detecting and repairing violations and as a basis for ranking. Along with two special kinds of rules, they constitute CARE’s knowledge base. First, rules with an empty body represent facts, describing models and metamodels [5]. Second, rules with an empty head, called ASP constraints\(^5\), are used to restrict repair actions and, thereby, prevent the generation of invalid solutions. Figure 5 shows an excerpt of such an ASP program, including the transformed metamodel and the input model from Figure 2 and the ASP representation of the rules from Table 2.

Encoding of Models and Metamodels. For the subset of Ecore specified in Section 3, each metamodel element is transformed into its corresponding ASP fact, i.e., class, attribute, and reference. Regarding the motivational example, a class fact is defined for each class, i.e., class('State'), class('SimpleState'), and class('FinalState'). Additional constructs (attributes of classes, relationships between elements) have been systematically derived from the Ecore subset, and corresponding ASP facts were introduced. Thereby, facts for attributes also contain the cardinality, represented by integer, e.g., a fact attribute(id, 'State', 1, 1) defines the attribute id for the class State with a minimum and maximum cardinality of 1. Regarding the motivational example, an additional fact for the abstract class State is added, i.e., isAbstract('State'), as well as facts for the hierarchy relationships, i.e., isParentOf('State', 'SimpleState'), and isParentOf('State', 'FinalState').

In a similar manner, the model elements object, value, and link are transformed to the ASP facts object, value, and link, respectively. Thereby, facts for objects consist of the name and type of the object, resulting in object('s1', 'State') and object('s2', 'State') for the example. Values are represented by their id, the actual value, and the object they belong to, resulting in, e.g., value(id, 1, 's1').

Encoding of Constraints and Repair Actions. Based upon this representation, the conformance constraints, repair actions, and generic ranking rules representing the generic core part have been manually defined once (cf. rules 1-8 in Table 2). Figure 5 shows the ASP code required for re-establishing ontological conformance for the motivational example. Lines 25-32 as well as line 35 are part of the CARE core functionality, while line 33 and line 36 have been derived from the metamodel-specific OCL expressions. OCL constraints consisting of boolean expressions, select, or size operations are candidates to be transformed automatically into ASP constraints (cf. rules 9-10 in Table 2), which may be established utilizing the approach of [25] in future versions of the prototype. In contrast, domain-specific ranking rules have to be specified manually (cf. rule 11 in Table 2, which has been transformed to line 35 in Figure 5). Although not required for generating valid solutions, they facilitate proper ranking of these solutions.

Re-Establishing Conformance. To execute ASP programs, CARE employs the DLV\(^6\) solver, which provides several extensions to ASP, such as the support for weak constraints [10], i.e., a special kind of ASP constraints to specify costs for non-compliance of constraints. Cost bounds and a limitation in terms of the number of results can be used to rank and restrict the resulting models, in order to compute best solutions meeting these quality criteria, only. In particular, computation can be speeded up, since solutions that score worse than a given cost bound can be eliminated early. The generated solutions are then transformed back to Ecore, allowing the user to select one of the ranked solutions.

5 Lessons Learned

For the experiments we extended the introductory example based on the UML 1.4 state machine metamodel, focusing on re-establishing ontological conformance of objects. In the following, we present lessons learned gained from these experiments.

CARE Core Depends on Structural Diversity. Given that CARE is solely used on basis of the generic core part, the ranking quality heavily depends on structural diversity of the classes in the metamodel. Consequently, if classes in the metamodel are structurally similar and no additional knowledge sources are incorporated, CARE suffers from the theoretically large solution space in terms of computational complexity.

Constraints Allow to Speed up Computation Process. As already introduced, CARE is capable of including metamodel-specific OCL constraints as well as user constraints, e.g., mapping instructions. Since these constraints represent hard facts, i.e., they narrow the solution space, they are an essential source for efficiently generating valid solutions. Consequently, with many constraints to be exploited in the computation process, CARE exhibits a considerably increased runtime performance. Nevertheless, in certain cases the presence of OCL constraints may lead to an empty solution, if no repair actions for OCL violations are present. Those repair actions have to be defined manually in the current prototype. An inves-

\(^4\)http://www.eclipse.org/xtend/

\(^5\)ASP constraints are dissimilar to the notion of constraints in this paper.

\(^6\)http://www.dlvsystem.com/
tigation of how to automatically derive repair actions for OCL constraints is part of future work.

**Ranking Depends on Favored Quality Criteria.** As mentioned, CARE provides a default configuration for ranking, which focuses on the prevention of information loss as a quality criterion for the resulting models. This may be undesired, e.g., in case that classes in the metamodel have been deleted and, consequently, corresponding instances should be deleted as well. Thereby, the incorporation of semantic knowledge may be beneficial to recognize semantically related concepts and, thus, rank corresponding solutions higher. Consequently, the user may influence the ranking by providing additional knowledge, which in turn favors different quality criteria.

**Runtime Complexity & Scalability.** As seen by means of the presented example, the solution space exponentially depends on the size of the metamodels and on the number of non-conforming instances. Referring to the formula for complexity \((c + 1)^3\) for the example (cf. Sect. 2), one may see that in the case of many non-conforming instances counter-measures are indispensable for efficient computation. Therefore, CARE provides several such measures including the usage of constraints, which reduce the number of possible solutions and, therefore, heavily improve runtime performance. Another way to alleviate this problem is to reclassify by class, instead of per object (e.g., all State objects become reclassified to the same class), effectively reducing the complexity to be linear only.

**Meta Information Provides a Valuable Source for Selection.** In cases, where multiple solution models remain for selection, which all exhibit the same costs according to the provided quality criteria, the generated meta-information in terms of provenance information and statistics represents a valuable source for the user to inspect and distinguish the solutions. Consequently, differences may be determined, to ultimately select the best solution.

In summary, CARE allows to re-establish conformance without comprising additional knowledge, but is especially useful for metamodels with comprehensive constraints, to generate the best solutions enriched by provenance information.

### 6 Related Work

This section discusses related approaches on detecting and repairing ontological conformance violations. Table 3 compares related approaches as well as our own approach CARE, regarding criteria discussed in Section 1 comprising tool coupling and application scope, user involvement, and provision of multiple solutions, as well as criteria related to the three phases of the CARE approach, i.e., ① detection, ② repairing of constraint violations, and ③ ranking of multiple solutions. Approaches are classified with respect to tight coupling, i.e., deeply woven into an IDE, or loose coupling to a specific modeling environment. Application scope indicates the specific or general applicability of the approach, by specifying the kinds of artifacts between which conformance is re-established. Detection of constraint violations considers both, the specification and the examination of constraints to expose their violations. Regarding repairing of constraint violations, we investigated generation of solutions, checking for contradictory repairing, and user involvement by examining user effort for repairing, e.g., specification of rules, and whether or not interactive user guidance is required for finding a valid solution. Approaches generating multiple solutions are examined for automatic ranking. Finally, we investi-
gated if meta information by means of provenance of the generated solutions is provided.

The examined approaches have been selected due to their capability for fixing inconsistencies in models or re-establishing conformance between models and their metamodels. They are classified according to tool coupling into either tight [9, 11, 14, 19, 22] or loosely [1, 4, 16, 24, 28, 30] coupled approaches, whereas the latter are more closely related to the CARE approach, since CARE is not coupled to or woven into a specific tool. The application scope ranges from detecting or repairing of constraint violations in UML models [1, 9, 22, 24, 28], over viewpoint synchronization [11], and re-establishing conformance in graph based models [16], to fixing of inconsistencies of models using the OCL related language BeanBag [30], and coupled evolution of metamodels and models [14]. CARE proposes a more generic approach for re-establishing conformance of models to a given Ecore-based metamodel. Specification and examination for detecting constraint violations is performed by logic programming [1, 4, 11, 22, 24, 28] using different execution engines, graph transformation rules [19], Java [14], or using the language BeanBag [30]. Like most approaches, CARE is also based on logic programming (in terms of ASP), but is unique in using the DLV solver, which allows the usage of weak constraints for ranking, by downgrading non-optional solutions. Most approaches support repairing or re-establishing conformance, either by generating a single solution [9, 14, 19, 22, 28, 30], multiple repair solutions [11] or repair rules [19], or by creating (partial) repair plans [1, 24]. Like only one other approach [11], CARE is able to generate multiple solutions in parallel, from which the user may select the most appropriate one, but is unique with respect to ranking them. Checking for non-contradictory repairing is essential, to either restrict the solutions or generate valid repair plans, and is performed by various approaches [9, 11, 19, 22, 24, 28], as well as by CARE, which computes valid solutions, only. Furthermore, we examined approaches with respect to the required extent of user involvement. Several approaches require manual effort to enable detection of non-conformance in models [1, 22, 30], while others rely on interactive user guidance for finding a valid solution [9, 11, 14, 19, 22]. Unlike those approaches, CARE does not rely on user interaction, as the provided generic conformance constraints as well as metamodel constraints allow to generate multiple valid solutions, but optionally user knowledge can be incorporated to further improve the generation and ranking of results. In contrast to the examined approaches, which support the generation of multiple, but unranked solutions [11, 16], CARE provides built-in heuristics which allow for configuration and, thus, an appropriate ranking of the generated solutions. CARE is unique with respect to providing provenance information of the generated solution.

In summary, when surveying related work with respect to CARE, one may see that CARE is unique with respect to the combination of (i) loose coupling to a specific tool or modeling environment, (ii) low user effort in the repairing process, and (iii) offering multiple valid and ranked solutions that are further enhanced by provenance information, which is then reported to the user for inspecting the differences between generated solutions.

### 7 Conclusion & Future Work

In this paper, we have shown, how a violated ontological conformance relationship between a model and its metamodel may be re-established based on conformance constraints, facilitated by logic programming. In particular, the presented approach supports the detection of conformance violations and provides repair actions to re-establish this conformance relationship. In contrast to other approaches, not only a single solu-

---

**Table 3: Comparison of Approaches for Re-Establishing Conformance**

<table>
<thead>
<tr>
<th>Tool Coupling</th>
<th>Application Scope</th>
<th>Specification</th>
<th>Examination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loose UML MM -&gt; UML model</td>
<td>Prolog Rules</td>
<td>Logic pre-dicates (Praxis)</td>
<td>(Partial) repair plans</td>
</tr>
<tr>
<td>Loose UML MM -&gt; UML model</td>
<td>Prolog Rules</td>
<td>Logic pre-dicates (Prolog)</td>
<td>X</td>
</tr>
<tr>
<td>Tight UML model A -&gt; UML model B</td>
<td>Abstract Rule Language</td>
<td>(Model profiler)</td>
<td>Single solution</td>
</tr>
<tr>
<td>Tight Viewpoint A -&gt; Viewpoint B</td>
<td>ASP</td>
<td>Logic pre-dicates (Smodels)</td>
<td>Multiple solutions</td>
</tr>
<tr>
<td>Tight MM -&gt; M co-evolution</td>
<td>Java</td>
<td>(Edapt)</td>
<td>Single solution</td>
</tr>
<tr>
<td>Loose Graph based models</td>
<td>Triple Rules (TGG)</td>
<td>Consistency rules</td>
<td>Multiple repair actions</td>
</tr>
<tr>
<td>Tight UML MM -&gt; UML model</td>
<td>Graph transformation rules</td>
<td>(AGG)</td>
<td>Single solution</td>
</tr>
<tr>
<td>Loose UML class diagram A -&gt; UML class diagram B</td>
<td>PDDL</td>
<td>Logic pre-dicates (PDDL)</td>
<td>Only plans</td>
</tr>
<tr>
<td>Loose UML model A -&gt; UML model B</td>
<td>Description Logic</td>
<td>Logic pre-dicates (Loom)</td>
<td>Single solution</td>
</tr>
<tr>
<td>Loose Model inconsistency fixing</td>
<td>BeanBag program</td>
<td>(BeanBag)</td>
<td>Single solution</td>
</tr>
<tr>
<td>CARE</td>
<td>Re-establishing M -&gt; MM conformance</td>
<td>ASP</td>
<td>Logic pre-dicates (DLV Solver)</td>
</tr>
</tbody>
</table>
tion, but a set of ranked solutions is generated based on several quality criteria exploiting structural and semantic knowledge that is enriched with provenance information and statistics to facilitate inspection by the user.

The current realization shows great potential, but there are several lines of future work. As already mentioned, the transformation to and from ASP is foreseen to be fully automated in the future. In this context, automating the transformation of OCL constraints into ASP rules leverages efficient measures to improve runtime performance. Furthermore, the generation of repair actions for OCL violations is of great interest, including the potential for reusing the already implemented repair actions. Regarding correctness and completeness of the proposed repair actions in Table 1, these properties are not proven formally, yet.

Ultimately, given the appropriate transformations, the implementation may be adapted to other domains and technical spaces besides Écoeur, e.g., database schemas with according instances. Alternatively, CARE may also be applied for the evolution of interdependent semantic web ontologies with accompanying owl constraints. Thereby, multi-domain ontologies such as DBpedia may be employed as source for semantic knowledge with the divi2hex [10] solver.

Acknowledgements

This work has been funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) under grant FFG BRIDGE 832160 and FFG FIT-IT 825070 and 829598, FFG Basisprogramm 838181, and by ÖAD under grant AR18/2013 and UA07/2013.

References


Variability in Artifact-Centric Process Modeling: The Hetero-Homogeneous Approach

Christoph Schütz  Michael Schrefl
Department of Business Informatics – Data & Knowledge Engineering
Johannes Kepler University Linz, Austria
e-mail: {schuetz,schrefl}@dke.uni-linz.ac.at

Abstract

Today’s dynamic business environment demands from companies variable and flexible processes. Rather than imposing a single fixed process, process models must account for the variability of real-world business problems. Many companies are hierarchically organized with top-down decision making processes. On the one hand, company policies and legal regulations often require compliance with standard process models prescribed by higher-level management. On the other hand, lower-level employees should be flexible within the prescribed boundaries. In this paper, we propose a hetero-homogeneous approach to modeling process variability. We employ the multilevel business artifact (MBA) in order to represent within a single object the homogeneous schema of an abstraction hierarchy of processes. We employ multilevel concretization for the introduction of heterogeneities into sub-hierarchies which comply with the homogeneous global schema.

Keywords: Business Artifact, Multilevel Modeling, Process Variability, Process Flexibility

1 Introduction

Business process models should reflect the variability of real-world business problems which are rarely solved by a single fixed process. Rather, different variations exist for tackling the same problem, depending on the exact situation. Thus, in order to accurately represent reality, a process model incorporates several variants for handling different situations. For example, a car rental company handles walk-in rentals differently from advance rentals, both being variants of a car rental.

An artifact-centric process model represents data along with the business processes that work with these data (Nigam & Caswell 2003, Kappel & Schrefl 1991). These business processes are represented as life cycle models of classes of data objects. It is common to model object life cycles with variants of finite state machines (Hull 2008). During its life cycle, a data object assumes several states which are defined in the life cycle model. Depending on the state, different operations may be performed on a data object.

In artifact-centric process modeling, behavior-consistent specialization of life cycle models allows for the representation of process variability. The specialization of a life cycle model may be considered a variant of the more general life cycle model. This variant must follow specific, well-defined rules in order to ensure consistency with the more general life cycle model (Stumptner & Schrefl 2000, Schrefl & Stumptner 2002, van der Aalst et al. 2002).

Many, if not most, companies are hierarchical organizations. From higher-level management to lower-level operatives, the different levels of the organization have their own business processes which are interconnected with each other. For example, top management decides what businesses a company operates in, area managers are concerned with shaping the businesses, and low-level operatives handle the specific business events. Multilevel process models represent processes at multiple levels of an organization together with the interactions between the processes at the different levels.

While higher-level management sets out general business policies, the exact processes may differ between the various subparts of the organization. Lower-level operatives must comply with the general policies but are flexible in adapting their respective processes within the limits specified by higher-level management. For example, the top management of a car rental company defines general policies for handling car rentals. The area managers for private and corporate renters may extend and refine these policies according to the particularities of each segment.

The hetero-homogeneous modeling approach provides modelers with increased flexibility for the representation of variability in multilevel process models while preserving the advantages of homogeneous schemas. Previously, the hetero-homogeneous modeling approach has been successfully employed in data warehouse modeling (Neumayr et al. 2010). A hetero-homogeneous business process model consists of a generally homogeneous schema but allows for the introduction of heterogeneities in well-defined sub-hierarchies. The extended and refined process models of the sub-hierarchies comply with the more general models and are themselves the homogeneous schema of their respective sub-hierarchy. The process models of sub-hierarchies of sub-hierarchies may again be extended and refined, and so on.

Figure 1 illustrates the hetero-homogeneous approach to modeling business process variability. Several multilevel artifact-centric process models (Rental, Private, Corporate, Rental2175) describe data and life cycle models at various hierarchically-ordered levels of abstraction (business, renterType, rentalAgreement, rental). Each process model consists of several boxes connected by dotted lines. Each box consists of several compartments, the top compartment containing, in angle brackets, the name of the level.
The life cycle model of class ‘business’ contains state ‘restructuring’ and state ‘in business’.

A renter type may be added to a business only if the active state of the business is ‘in business’.

The active state of business ‘Rent’ is ‘in business’.

renterType >

center type has maximum rate.

The life cycle model of class ‘renter type’ contains state ‘in development’ and state ‘on offer’.

A renter type may be added to a renter type only if the active state of the renter type is ‘in offer’ and the active state of the business of the renter type is ‘in business’.

rental >

rental has actual pick-up date and rate and rental duration.

The life cycle model of class ‘rental’ contains state ‘open’ and state ‘closed’. The rate of a rental must not exceed the maximum rate of the renter type.

advance rental specializes rental.

advance rental has scheduled pick-up date.

The life cycle model of class ‘advance rental’ contains state ‘reserved’ and state ‘assigned’.

<table>
<thead>
<tr>
<th>Process Model</th>
<th>Concretization of</th>
<th>Concretization of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private renter</td>
<td>renterType &gt;</td>
<td>Corporate renter</td>
</tr>
<tr>
<td>The maximum rate of renter type is ‘1 000’.</td>
<td>The maximum rate of renter type is ‘1 000’.</td>
<td></td>
</tr>
<tr>
<td>The active state of renter type ‘Private’ is ‘in offer’.</td>
<td>The active state of renter type ‘Corporate’ is ‘in development’.</td>
<td></td>
</tr>
<tr>
<td>private renter specializes rental.</td>
<td>Corporate renter specializes rental.</td>
<td></td>
</tr>
<tr>
<td>private renter has credit card number.</td>
<td>Corporate renter has an upgrade fee.</td>
<td></td>
</tr>
<tr>
<td>The life cycle model of class ‘private rental’ contains state ‘booked’.</td>
<td>The life cycle model of class ‘rental agreement’ contains state ‘under negotiation’ and state ‘in effect’.</td>
<td></td>
</tr>
<tr>
<td>private advance rental specializes private rental.</td>
<td>rental</td>
<td></td>
</tr>
<tr>
<td>private advance rental has deposit.</td>
<td>corporate rental specializes rental.</td>
<td></td>
</tr>
<tr>
<td>The life cycle model of class ‘private advance rental’ contains state ‘deposited’.</td>
<td>corporate rental has an upgrade fee.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process Model</th>
<th>Concretization of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rental2175: rental</td>
<td></td>
</tr>
<tr>
<td>The rental ‘Rental2175’ is instance of class ‘private advance rental’.</td>
<td></td>
</tr>
<tr>
<td>The rate of rental ‘Rental2175’ is ‘€ 80’.</td>
<td></td>
</tr>
<tr>
<td>The rental duration of rental ‘Rental2175’ is 10 days.</td>
<td></td>
</tr>
<tr>
<td>The credit card number of private rental ‘Rental2175’ is ‘1234567890’.</td>
<td></td>
</tr>
<tr>
<td>The scheduled pick-up date of advance rental ‘Rental2175’ is 12/08/2013.</td>
<td></td>
</tr>
<tr>
<td>The deposit of private advance rental ‘Rental2175’ is ‘€ 120’.</td>
<td></td>
</tr>
<tr>
<td>The active state of private advance rental ‘Rental2175’ is ‘deposited’.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process Model</th>
<th>Concretization of</th>
</tr>
</thead>
<tbody>
<tr>
<td>rentalAgreement</td>
<td></td>
</tr>
<tr>
<td>rentalAgreement has negotiated rental rates and maximum rental duration.</td>
<td></td>
</tr>
<tr>
<td>The life cycle model of class ‘rental agreement’ contains state ‘under negotiation’ and state ‘in effect’.</td>
<td></td>
</tr>
<tr>
<td>corporate rental</td>
<td></td>
</tr>
<tr>
<td>corporate rental specializes rental.</td>
<td></td>
</tr>
<tr>
<td>corporate rental has an upgrade fee.</td>
<td></td>
</tr>
<tr>
<td>The rate of a corporate rental must be included in the negotiated rental rates of the rental agreement.</td>
<td></td>
</tr>
<tr>
<td>The rental duration of a corporate rental must not exceed the maximum rental duration of the rental agreement.</td>
<td></td>
</tr>
<tr>
<td>carsharing rental specializes corporate rental.</td>
<td></td>
</tr>
<tr>
<td>carsharing rental has driven distance.</td>
<td></td>
</tr>
<tr>
<td>The life cycle model of class ‘carsharing rental’ contains state ‘active’ and state ‘closed’.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Modeling variability in business processes using the hetero-homogeneous approach

- The life cycle model of class ‘business’ contains state ‘restructuring’ and state ‘in business’.
- The life cycle model of class ‘rental type’ contains state ‘in development’ and state ‘on offer’.
- The active state of business ‘Rent’ is ‘in business’.
- The life cycle model of class ‘rental’ contains state ‘open’ and state ‘closed’.
- The life cycle model of class ‘advance rental’ contains state ‘reserved’ and state ‘assigned’.
- The life cycle model of class ‘private rental’ contains state ‘booked’.
- The life cycle model of class ‘private advance rental’ contains state ‘deposited’.
- The life cycle model of class ‘corporate rental’ contains state ‘active’ and state ‘closed’.

These business processes are interdependent and interact with each other. The encapsulation of multiple processes at different levels of abstraction within a single model allows for a definition of variants of entire process hierarchies.

2 Variability in the Large: Hierarchies of Multilevel Process Models

A multilevel process model represents a hierarchy of business processes at different levels of abstraction. These business processes are interdependent and interact with each other. The introduction of heterogeneities in the Sub-hierarchy does not affect the schema of the Private sub-hierarchy, and vice versa.

Besides having variants of entire process hierarchies through concretization of multilevel process models, a single multilevel process model may define different process variants within an abstraction level. For example, in Figure 1, the Rental multilevel process model defines advance rentals as a variant of rentals. Different sub-hierarchies may specialize these variants or introduce additional variants. For example, the Private multilevel process model specializes the schema of rentals and advance rentals. The Corporate multilevel process model specializes the schema of rentals and introduces carsharing rentals as a variant of corporate rentals.

In this paper, we adopt and extend the multilevel business artifact (Schütz et al. 2013) for the hetero-homogeneous modeling of artifact-centric process models. The encapsulation of information about an entire hierarchy of artifact-centric process models within a single object together with a concretization mechanism allows for a flexible introduction of heterogeneities while preserving the advantages of homogeneous process models.

The remainder of this paper is organized as follows. In Section 2, we present the modeling and incremental evolution of hierarchies of multilevel process models. In Section 3, we present the modeling and incremental evolution of hierarchies of process models within the individual levels of a multilevel process model. In Section 4, we review related work. We conclude with a summary and an outlook on future work.
2.1 Modeling

A business artifact (Nigam & Caswell 2003) encapsulates, within a single object, a data model as well as the corresponding life cycle model.Artifact-centric (or data-centric) process models focus on data and the operations that manipulate the data as well as their execution order. Many artifact-centric approaches to process modeling, for example, object/behavior diagrams (Kappel & Schreff 1991), rely on variants of finite state machines for the representation of life cycle models, defining a set of states which an artifact runs through as the data change.

A multilevel object (m-object) encapsulates, within a single object, data models at various levels of abstraction (Neumayr et al. 2009). The abstraction levels are arranged in a hierarchy from most abstract to most concrete with a single top level which is the most abstract. To each abstraction level, an m-object links a class. The classes are related by aggregation relationships according to the level hierarchy. Modelers are free to choose the exact semantics of the aggregation relationships between the classes, the possibilities ranging from part-of relationships to aggregation with a materialization flavor (Dalchour et al. 2002). Besides defining classes at various levels of abstraction, an m-object instantiates its single top-level class, yielding a certain “class/object duality” (Atkinson & Kühne 2001) similar to clabjects.

The multilevel business artifact (MBA) is an extension of the m-object for artifact-centric process modeling (Schütz et al. 2013). An MBA encapsulates, within a single object, data and life cycle models at various levels of abstraction. To each abstraction level, an MBA links a class as well as the corresponding life cycle model. This life cycle model defines the legal execution order of the methods of the respective class. Being an instance of its top-level class, an MBA also has an active state (or several) from the top-level life cycle model.

Figure 2 defines the MBA metamodel using UML. Note, however, that MBAs are outside of traditional object-oriented thinking even though UML serves as the language for the definition of the MBA metamodel. The definition of the MBA metamodel in UML allows for the use of OCL constraints for the synchronization of life cycle models on different abstraction levels. Other modeling languages, for example, O-Telos (Jeusfeld et al. 2009), are equally well-suited for defining the MBA metamodel.

Class MBA is the metaclass of all MBAs. An MBA references several abstraction levels (Level) which are hierarchically ordered; an MBA has a single top level. For each level, an MBA defines a single class or a class hierarchy (see Section 3). Each class is only linked to a single MBA and level. Each class has a state machine which defines the life cycle model of the class. An MBA is also an instance of a class (ClassInstance), the classifier being a class that is linked to the MBA’s top level. In order to ensure that each instance of an MBA’s class is again an MBA, each of these classes is a specialization of the MBA metaclass. Each class has at most one direct instance, but may have several indirect instances via sub-classes which are defined by concretizations.

An MBA may be the concretization or, conversely, the abstraction of another MBA (see Section 2.2). An MBA inherits a set of levels from its abstraction. The concretization must maintain the relative order of inherited levels but may introduce additional levels between inherited levels. For simplicity of presentation, an MBA explicitly references the newly introduced as well as the inherited levels. The hierarchical order of levels is then defined locally for each MBA using an association class (MBAtoLevel). Each link between an MBA and a level references the parent level in the context of the MBA. In a similar way, class definitions are attached to these links.

The MBA metaclass defines methods for the navigation along the concretization hierarchy. Method descendants takes a level as parameter and returns the set of the MBA’s (transitive) concretizations with the argument top level. Similarly, method ancestor takes a level as parameter and returns the MBA’s (transitive) abstraction with the argument top level.

Figure 3: MBA Rental for the management of car rental data
dotted lines. The top compartment of the top level’s box contains the name of the MBA in addition to the level number underlined and separated by a colon, reflecting the instantiation of the top-level class by the MBA. Furthermore, attributes of the top level’s box have values assigned.

Figure 3 illustrates an MBA model for the management of car rental data with MBAs Rental and Corporate; the example is based on the EU-rent use case (OMG 2008, p. 267 et seq.). MBA Rental has levels business, renter type, and rental. The business level represents the company’s rental business, consisting of several renter types (renterType), each having several individual rentals (rental). The rental business has a description. A renter type has a maximum rate (maximumRate), a rental duration (rentalDuration), an actual pickup date (actualPickUp), a rental duration (rentalDuration), a rental rate which determines the total rental fee, and an assigned car (assignedCar). MBA Corporate is a concretization of Rental (see Section 2.2).

We use UML (protocol) state machines (OMG 2011, p. 535 et seq.) for the representation of life cycle models. We stress, though, that the employed process modeling language is substitutable. We use the UML state machine formalism since it is an industry standard. A state machine is defined in the context of a class and consists of states and transitions between these states. A transition has a source state and a target state and is linked to a call event for a method in the context class. A method may be called for a particular object if in the object’s life cycle model there exists a transition that is linked to the called method and originating an active state of the object. Furthermore, possibly specified pre- and post-conditions must be satisfied. A valid method call triggers the transition of the object from source state to target state. Methods that are not linked to any transition may be called in any state. A state may have several sub-states which are also linked by transitions. Forks and parallel regions allow for an object to be in multiple states simultaneously.

In the graphical representation of UML state machines, a rounded box with a caption inside represents a state, a filled black circle represents the initial state, and an arrow with a method name represents a transition. Pre-conditions and post-conditions of transitions are placed in square brackets before the method name and after the method name, respectively. The name of an object’s active state is underlined (non-standard notation).

For example, each level of MBA Rental (Figure 3) has a state machine as life cycle model. A business is either Restructuring or In Business and moves between these states. A renterType moves from In Development to On Offer, Phase Out, and Discontinued. A rental moves from Opening to Open and Closed. Since Rental is also an instance of its top-level class it has an active state, In Business, from the top-level (business) life cycle model.

The life cycle models at the various abstraction levels of an MBA constitute the model of a multilevel business process. The different life cycle models of a multilevel business process are interdependent; the MBAs that instantiate the corresponding classes interact with each other. For example, a new renter type may only be added to a business while it is Restructuring. When a business moves from In Business to Restructuring all associated renter types move to the Phase Out state. Similarly, a new individual rental may only be added to a renter type while it is On Offer and the business is In Business. A renter type may only be discontinued if all associated individual rentals are closed. A rental’s rate must not exceed the maximum rental rate defined by the renter type. The life cycle models of an MBA are thus connected by synchronization dependencies which are pre- and post-conditions, expressed in OCL, for the transitions between states.

We define a set of frequently used patterns of synchronization dependencies between abstraction levels, called multilevel predicates, as syntax macros (Leavenson 1966) for OCL (Figure 4). Multilevel predicates are classified into attribute synchronization, state synchronization, and concretization predicates. Attribute synchronization refers to pre- and post-conditions demanding that the value of a given attribute of descendants or an ancestor satisfies some condition. State synchronization refers to pre- and post-conditions demanding that descendants are or an ancestor satisfies some condition. Concretization predicates trigger the creation of new MBAs and can be used only in post-conditions. Multilevel predicates are translated into standard OCL constraints which use the methods descendants and ancestor of the MBA metaclass for navigation along the level hierarchy.

The multilevel predicates allDescendantsAtLevelSatisfy, someDescendantAtLevelSatisfies, isDescendantAtLevelSatisfies, and ancestorAtLevelSatisfies handle attribute synchronization. Predicate allDescendantsAtLevelSatisfy demands that all descendants at a given level satisfy some condition over an at-

```
Figure 4: Multilevel predicates for the definition of synchronization dependencies as macros for OCL

<table>
<thead>
<tr>
<th>Attribute synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>allDescendantsAtLevelSatisfy(level, attrName, value, 0)</td>
</tr>
<tr>
<td>someDescendantAtLevelSatisfies(level, attrName, value, 0)</td>
</tr>
<tr>
<td>isDescendantAtLevelSatisfies(obj, level, attrName, value, 0)</td>
</tr>
<tr>
<td>ancestorAtLevelSatisfies(level, attrName, value, 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>allDescendantsAtLevelInState(level, state)</td>
</tr>
<tr>
<td>someDescendantAtLevelInState(level, state)</td>
</tr>
<tr>
<td>isDescendantAtLevelInState(obj, level, state)</td>
</tr>
<tr>
<td>ancestorAtLevelInState(level, state)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concretization</th>
</tr>
</thead>
<tbody>
<tr>
<td>newDescendantAtLevel[level]</td>
</tr>
<tr>
<td>newDescendantAtLevelSatisfies(level, attrName, value, 0)</td>
</tr>
<tr>
<td>newDescendantAtLevelUnder(level, obj)</td>
</tr>
<tr>
<td>newDescendantAtLevelUnderSatisfies(level, obj, attrName, value, 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auxiliary</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDescendantAtLevel(obj, level)</td>
</tr>
</tbody>
</table>
```
tribute. Predicate someDescendantAtLevelSatisfies demands that at least one descendant at a given level satisfies some condition over an attribute. Predicate isDescendantAtLevelSatisfies checks whether a given object is a descendant at a given level and satisfies some condition over an attribute. Predicate ancestorAtLevelSatisfies demands that the ancestor at a given level satisfies some condition over an attribute.

The multilevel predicates allDescendantsAtLevelInState, someDescendantAtLevelInState, isDescendantAtLevelInState, ancestorAtLevelInState handle state synchronization. Predicate allDescendantsAtLevelInState demands that all descendants at a given level are in a particular state. Predicate someDescendantAtLevelInState demands that at least one descendant at a given level is in a particular state. Predicate isDescendantAtLevelInState demands that a given MBA is a descendant at a given level and the MBA is in a particular state. Predicate ancestorAtLevelInState demands that the ancestor at a given level is in a particular state.

The multilevel predicates newDescendantAtLevel, newDescendantAtLevelSatisfies, newDescendantAtLevelUnder, and newDescendantAtLevelUnderSatisfies are concretization predicates. Predicate newDescendantAtLevel demands that a new descendant at a given level exists after the execution of the method. Predicate newDescendantAtLevelSatisfies demands that a new descendant at a given level exists after the execution of the method and that this new descendant satisfies some condition over an attribute. Predicate newDescendantAtLevelUnder demands that a new descendant at a given level exists after the execution of the method and that this new descendant is also the descendant of a given other MBA. Predicate newDescendantAtLevelUnderSatisfies combines predicates newDescendantAtLevelSatisfies and newDescendantAtLevelUnder.

We provide graphical notations for state synchronization and concretization predicates (Figure 5). These kinds of synchronization are visualized by dashed arrows between states and transitions of different levels. Depending on the direction of the arrow, the synchronization dependency is either a pre-condition (Figures 5a and 5c) or a post-condition (Figures 5b, 5d, and 5e-5h) for a method call. The annotation of a dashed arrow with the symbol for existential quantification (∃, not shown) denotes the someDescendantAtLevelInState predicate. The isDescendantAtLevelInState state synchronization predicate does not have a graphical notation.

For example, in Figure 3, synchronization dependencies between levels are defined using the graphical notations for multilevel predicates. At the business level of MBA Renter, method restructure has as a pre-condition an allDescendantsAtLevelInState predicate with the Phase Out state of the renterType level as argument. Method developRenterType has as a post-condition a newDescendantAtLevelSatisfies predicate with the renterType level as argument and a condition over the name attribute which every MBA has implicitly defined. At the renterType level, method openRental has as a pre-condition an ancestorAtLevelInState predicate with the In Business state of the business level as argument. Method openRental has as a post-condition a newDescendantAtLevelSatisfies predicate with the rental level as argument and a condition over the rentalId attribute. Method discontinue has as a pre-condition an allDescendantsAtLevelInState predicate with the Closed state of the rental level as argument. At the rental level, method setRate has as a pre-condition an ancestorAtLevelInState predicate defining that the rate that is to be set must not exceed the value of the maximumRate attribute of the ancestor at the renterType level. The ancestorAtLevelSatisfies predicate has no special graphical notation.

2.2 Incremental Evolution

An MBA defines a multilevel process model and multilevel concretization allows for the definition of abstraction hierarchies of multilevel process models. Each MBA represents an entire multilevel abstraction hierarchy of artifact-centric process models. For this hierarchy, an MBA defines a homogeneous schema. Through multilevel concretization, modelers may extend and refine the homogeneous schema for a particular sub-hierarchy. The extended and refined schema becomes the homogeneous schema of the sub-hierarchy. For a sub-hierarchy of this sub-hierarchy, in an incremental, iterative manner, modelers may again extend and refine the homogeneous schema.

Multilevel concretization is a relationship between a concretizing MBA, referred to as the concretization, and a concretized MBA, referred to as the abstraction. Multilevel concretization combines charac-
A concretization of MBA Rental in Figure 3 is a concretization of MBA. MBA Rental is part of the Rental business. MBA Corporate has top level renterType, the second level of Rental. MBA Corporate inherits from Rental all levels from renterType downwards and including, that is, levels renterType and rental. MBA Corporate specializes the class that is linked to the rental level of MBA Rental. A corporate rental has an upgrade fee (upgradeFee) which allows employees to upgrade the assigned car for a privately paid fee. The inherited attributes are not shown in the graphical representation. MBA Corporate is an instance of the class at the renterType level of MBA Rental, assigning a value of 1000 to attribute maximumRate. The active state of MBA Corporate is In Development. MBA Corporate also specializes the life cycle models of levels renterType and rental.

The concretization’s life cycle models that are linked to inherited levels are observation-consistent specializations of the abstraction’s life cycle models. Intuitively, observation consistency guarantees that if states and transitions added by the specialized life cycle model are ignored and refined states are considered unrefined, any processing of a data object according to the specialized life cycle model can be observed as correct processing from the point of view of the more general life cycle model (Schrefl & Stumptner 2002). An observation-consistent specialization of a life cycle model may extend the more general life cycle model with additional, parallel paths and refine inherited states with sub-states. Pre- and post-conditions in the specialized life cycle model must be at least as strong as in the more general model. The rules for observation-consistent specialization heavily depend on the employed modeling formalism. We refer to other work (Stumptner & Schrefl 2000, Schrefl & Stumptner 2002, van der Aalst et al. 2002) for a formal specification of these rules.

For example, MBA Corporate (Figure 6) extends and refines the inherited life cycle models of Rental. In the graphical representation, the inherited states and transitions are depicted in gray. At the renterType level, MBA Corporate extends the inherited life cycle model with a parallel region after the In Development state. Besides being On Offer, the Corporate renter type is, at the same time, also either in the Expanding or the Consolidating state. At the rental level, MBA Corporate extends and refines the inherited life cycle model. A corporate rental has an upgrade fee which may only be set when the corporate rental is in the Opening state. The Closed state is refined by sub-states Returned and Archived.

We permit the introduction of new transitions between inherited states in observation-consistent specializations. For example, at the rental level, MBA Corporate introduces a transition that is linked to the setUpgradeFee method, with the inherited Open state as source and target. Depending on the modeling formalism, the introduction of transitions where either source or target state is inherited violates observation consistency (Schrefl & Stumptner 2002). For modeling formalisms that take into account the run time of methods, transitions may only be introduced between newly introduced states. Due to the run-to-completion assumption (OMG 2011, p. 574 et seq.) in UML, however, the introduction of transitions between inherited states may be considered observation-consistent. We stress, though, that the employed modeling formalism is not an important aspect of multilevel process models.

A concretization may also introduce additional levels with respect to the abstraction. A newly introduced level must be underneath the top level; the relative order of the inherited levels must be preserved. For example, MBA Corporate (Figure 6) introduces rentalAgreement between levels renterType and rental. A rental agreement is a contract which defines general conditions for individual rentals of a corporate client, specifying a maximum rental duration (maximumRentalDuration) and a set of rental rates (negotiatedRentalRates). While Under Negotiation, the maximum rental duration is determined and a set of rental rates is negotiated.

The observation-consistent specialization of synchronization dependencies is a particular case of specialization of pre- and post-conditions by strengthen-
ing. The inherited state synchronization must still be satisfied; the specialization may provide additional semantics. In particular, the concretization may refine a state synchronization by replacing the original state with a sub-state. In this case, however, the modeler must ensure that no deadlocks may occur due to this specialization. For example, at the renterType level, MBA Rental (Figure 3) defines a pre-condition for the continue method in the transition between Phase Out and Discontinued: a renter type may only be discontinued if all associated rentals are closed. This pre-condition is inherited by MBA Corporate which adds a transitive synchronization dependency to a sub-state of Closed at the rental level. In order to avoid possible deadlocks, the archive method has as pre-condition a state synchronization with the ancestor at the rental-Agreement level.

When introducing additional levels, modelers must specialize the concretization predicates. For example, at the renterType level of MBA Rental (Figure 3), the openRental method has a concretization predicate as post-condition, emphasizing the creation of a new concretization at the rental level by this method. MBA Corporate introduces an additional level and specializes this post-condition. A descendant of this renter type at the rental level must be added as a concretization underneath a rental agreement. The agreement must be a descendant of the renter type, which is emphasized by the additional pre-condition of the openRental method. As a modeler-defined constraint, the agreement must be In Effect.

3 Variability in the Small: Hierarchies of Process Models within Levels

For an abstraction level, a multilevel process model may define several variants. The selection of a variant for the process execution is deferred to a later point in time, thereby increasing the flexibility of employees.

3.1 Modeling

A multilevel business artifact (MBA) may link an entire specialization hierarchy of classes to an abstraction level, which allows for the definition of process variants within a single level. In this case, instead of a single class, an MBA defines a set of classes for the abstraction level. A single most general class serves as the superclass for an arbitrary number of specializations. The life cycle models of these classes follow the rules for behavior-consistent specialization. Thus, each class in such a specialization hierarchy together with the life cycle model is a variant of an artifact-centric process model.

For example, in Figure 7, MBA Corporate links an entire class hierarchy to the rental level. In this hierarchy, CorporateRental is the most general class, with CorporateAdvanceRental and CorporateCarsharingRental being specializations. The life cycle model of class CorporateAdvanceRental refines the Open state. An advance rental has a scheduled pick-up date (scheduledPickUp) and separates the recording of basic rental information (done in the Booking state) from the assignment of an actual car which is carried out when the rental is already Booked. The life cycle model of class CorporateCarsharingRental refines the Open state. A carsharing rental is billed by driven distance (drivenDistance) and may involve changes of the assigned car. The renter may pause an Active rental and choose another car from a car pool before resuming the rental.

After creation, an MBA is, by default, an instance of the single most general class that is linked to the top level. An MBA may then change its classifier during the life cycle. The setClassifier method of the MBA metaclass allows for the explicit consideration of classifier change in the life cycle model. This possibility of incremental classification defers a final instantiation decision, increasing the flexibility of employees.
Incremental classification allows for the dynamic specialization and generalization of an MBA’s classifier. Instance specialization refers to a change of an MBA’s classifier from more general to specialized. Instance generalization, in turn, refers to a change of an MBA’s classifier from specialized to more general. Both types of incremental classification can be combined for instance migration which allows for a change of an MBA’s classifier to another classifier that is a sub-class of a common super-class.

In order for instance specialization to be valid, certain conditions must be met by the MBA. A change of an MBA’s classifier from more general to specialized is valid if the previous processing steps of the MBA in the more general life cycle model also represent a valid execution of the specialized life cycle model. In this case, the MBA can resume execution in the specialized life cycle model. For example, consider an MBA that is an instance of CorporateRental in the Opening state. A change of this MBA’s classifier to CorporateAdvanceRental is possible and puts the MBA in the Booking state afterwards, a sub-state of Opening. Consider now an MBA that is an instance of CorporateRental and in the Open state. A change of this MBA’s classifier to CorporateAdvanceRental is not allowed. The change of classifier would put the MBA in the Open state. As an instance of CorporateAdvanceRental the MBA would have had to run through the refined Opening state in order to present a valid life cycle. A change of classifier to CorporateCarsharingRental, however, is possible and puts the MBA in state Active, a sub-state of Open.

Instance generalization is always possible unless explicitly prohibited by the life cycle model. Values of attributes that are introduced by the specialized class are dropped. If in a refined state at first, after the change of classifier, the MBA is in the unrefined state of the general life cycle model. For example, consider an MBA that is an instance of CorporateAdvanceRental in the Assigned state. A change of this MBA’s classifier to CorporateRental puts the MBA in the Opening state, the unrefined super-state of Assigned. The value of scheduledPickUp is dropped.

Instance migration refers to a change of an MBA’s classifier to another sub-class of the MBA’s current classifier’s super-class; instance migration is realized as a sequence of instance generalization and specialization. For example, consider an MBA that is an instance of CorporateAdvanceRental. A change of this MBA’s classifier from CorporateAdvanceRental to CorporateCarsharingRental is a two-step procedure. First, the classifier changes from CorporateAdvanceRental to the more general CorporateRental, the common superclass of CorporateAdvanceRental and CorporateCarsharingRental. Second, the classifier changes from CorporateRental to CorporateCarsharingRental.

### 3.2 Incremental Evolution

For each inherited level, a concretization inherits all of the linked classes and life cycle models. If an inherited level is linked to a class hierarchy, the concretization inherits the entire class hierarchy. This inherited class hierarchy may be specialized. On the one hand, the concretization may introduce additional sub-classes. On the other hand, the concretization may specialize only individual classes of the inherited class hierarchy.

With class hierarchies involved, multilevel concretization may lead to double specialization of classes and life cycle models. Each class of a concretization’s inherited class hierarchy is a specialization of the abstraction’s corresponding class. When the concretiza-
defines classes Rental and AdvanceRental which are in a specialization/generalization relationship with each other. MBA Private, at the rental level, defines classes PrivateRental and PrivateAdvanceRental. Class PrivateRental is a specialization of class Rental which is defined by MBA Rental. A private rental must have credit card information and is either Unbacked or Backed, depending on the availability of credit card information. Class PrivateAdvanceRental is a specialization of class PrivateRental as well as class AdvanceRental which is defined by MBA Rental. A private advance rental requires the customer to deposit an amount of money in order to guarantee the reservation. Once deposited, a private advance rental turns from Authorized into Guaranteed, thereby refining the Backed state.

In this paper, we do not focus on the details of behavior consistency under double specialization. We provide, however, two modeling guidelines for the realization of observation-consistent double specialization. These guidelines simplify consistency checking under double specialization but restrict the freedom of the modeler. General consistency checking under double specialization is an open issue for future work. The issue of double specialization in process modeling is not the main issue in the hetero-homogeneous approach.

In order to avoid double specialization, a modeler may choose to specialize only the leaf nodes of a class hierarchy. In this case, behavior consistency must only be checked against the life cycle model of the super-class in the inherited class hierarchy. This simplification, however, limits the freedom of the modeler and reduces flexibility. Thus, it is desirable to allow double specialization of life cycle models.

Double specialization of life cycle models bears no conflict if the specializations occur in parallel regions or independent states of the life cycle model. For example, the specialization of the Rental class’s life cycle model by the AdvanceRental class (Figure 8) is a refinement of the Opening state. The specialization of the Rental class’s life cycle model by the PrivateRental class is an extension with a region that is parallel to the Opening state. These specializations are independent from each other. A combination of the two life cycle models in the PrivateAdvanceRental class’s life cycle model is without problems.

4 Related Work
Just like a multilevel object (m-object), a multilevel business artifact (MBA) is very similar to a powertype. Powertypes present metamodeling capabilities (Ode 1998, p.28). The instances of a powertype are subtypes of another object type (Gonzalez-Perez & Henderson-Sellers 2006). Using the notion of “clabject”, the instances of a powertype are class and object at the same time. Relating the MBA approach to powertype-based approaches (Erickson et al. 2013), a level of an MBA may act both as partitioned type and powertype; in an MBA’s level hierarchy, a parent level is a powertype of the child level. MBAs (and m-objects) also present characteristics of deep instantiation and materialization. Deep instantiation (Atkinson & Kühne 2001) facilitates the modeling of arbitrary-depth instantiation hierarchies where data objects can instantiate (certain aspects of) other data objects which instantiate other data objects, and so on. Materialization (Dahchour et al. 2011) on the other hand, blurs the boundaries between aggregation and instantiation. In this respect, multilevel concretization is similar to materialization. Neumayr et al. (2011) provide a comprehensive comparison between m-objects and other multilevel modeling techniques.

In the context of business processes, model abstraction commonly refers to the hiding of unnecessary details from the user (Smirnov et al. 2012). The guard-stage-milestone (Hull et al. 2010) approach, for example, introduces this kind of abstraction for business artifacts. The issue of process model abstraction in the traditional sense is orthogonal to multilevel process modeling with MBAs. The MBA represents interdependent processes of objects at various levels of abstraction which are in some sort of aggregation relationship. A process space (Motahari-Nezhad et al. 2011), on the other hand, provides several views on the same business process, each view with the emphasis on a different aspect of the process. For example, the CEO of a company has a more high-level view on the sales process than the sales manager.

Many business process modeling approaches account for the variability of real-world processes. Configurable business process models (La Rosa et al. 2011, La Rosa 2009) incorporate several variants of a process which provide the process owners with different options. Business process families (Gröner et al. 2013) introduce the well-known principle of software product lines to business process modeling. A business process family comprises a reference model and a set of features which adhere to the core intended behavior specified by the reference model. Process owners may customize the business process by using different selections of features. Case handling (van der Aalst et al. 2005) provides the process owners with a choice of options, offering a great deal of flexibility.

The MBA approach employs behavior-consistent specialization of life cycle models for the representation of variability. A behavior-consistent specialization may be regarded as a variant of the more general life cycle model. Many frameworks for behavior-consistent specialization exist using various modeling languages, for example, Petri nets (van der Aalst et al. 2002), UML state machines (Stumptner & Schrefl 2000), or object/behavior diagrams (Schrefl & Stumptner 2002). In recent work, Yongchareon et al. (2012) investigate the observation-consistent specialization of synchronization dependencies.

Related to the issue of variability are the notions of flexibility and agility in business process modeling. Reichert & Weber (2012) provide a comprehensive treatment of flexibility in business process modeling. Milanovic et al. (2011) identify a set of rule patterns which can be used for the modeling of agile business processes. Liu et al. (2012) propose an integration of reflective operations into the process model in order to explicitly account for the manipulation of a business artifact’s schema. The MBA approach provides flexibility through incremental classification. MBAs could also incorporate reflective operations for schema manipulation. Rules for behavior-consistent specialization constrain the possibilities of on-the-fly schema manipulation.

5 Summary and Future Work
The hierarchical organization is arguably the predominant organizational structure among large companies. A hierarchical organization often has rigid top-down decision-making processes. Today’s dynamic business environment, however, demands increased flexibility from companies. The hierarchical modeling approach overcomes the dichotomy between the rigidity imposed by a hierarchical organization...
and the flexibility that is required in a dynamic business environment. Future work will integrate the hetero-homogeneous approach into existing modelling languages and tools, for example, BPMN or the guard-stage-milestone model for business artifacts.

References


La Rosa, M. (2009), Managing variability in process-aware information systems, PhD thesis, Queensland University of Technology, Brisbane, Australia.


Conflict Resolution for On-the-fly Change Propagation in Business Processes

Shamila Mafazi1 Wolfgang Mayer2 Markus Stumptner2

1,2 School of IT and Mathematical Sciences,
University of South Australia, Adelaide, SA, 5095, Australia,
1 Email: shamila.mafazi@mymail.unisa.edu.au
2 Email: firstname.lastname@unisa.edu.au

Abstract

Process models are widely used in organisations and can easily become large and complex. In the context of business process management, views are a useful technique to reduce complexity by providing only those process fragments that are relevant for a particular stakeholder. A key challenge in view management is the handling of changes that are performed concurrently by different stakeholders. Since the views may refer to the same process, the performed changes may affect the same region of a business process and cause a conflict.

Many approaches have been proposed for resolving conflicts in a post-analysis phase after all changes have been applied. They can become costly when dealing with multiple changes that lead to multiple conflicts which cannot be resolved automatically and require an additional negotiation phase between stakeholders.

In this paper we propose a framework for the on-the-fly conflict resolution of changes that have been performed on views their underlying reference process. Different to existing approaches this framework applies behaviour consistency rules for business processes which consider the execution semantics and can be checked efficiently on the structure of processes without generating all possible execution traces or keeping track of change operations.

1 Introduction

Since size of a process model is a fundamental factor in its understandability (Rosa et al. 2010), different stakeholders of a process model are usually interested in individual, specific views over a detailed reference process model. This requires creating different process views considering the various requirements. However, models evolve and changes may cause discrepancies between the models.

There are several internal and external drivers for change in a process model. Once a view of a reference model is changed, it may no longer be consistent with the reference process model and other views. Inconsistencies and conflicts are two issues resulting from applying change operators (Barrett et al. 2008). Inconsistencies can arise from changes in the structure of a process model, such as addition or removal of a task, attribute, and data flow, as well as changes to attributes of said elements. Inconsistencies in processes and supporting systems can elevate costs and slow the development (Spanoudakis & Zisman 2001).

In general change may need to be propagated to other models in order to preserve consistency among related views and the reference process. Semi-automated change propagation techniques have been proposed (Weidlich et al. 2012, Gerth 2013) but are usually performed in a separate step after all changes have been applied. However, a separate post-analysis step has the disadvantage that it may start another change cycle and can become costly. For example, if multiple changes performed on the same region of a process model by different stakeholders cannot be resolved automatically then the stakeholders must negotiate in a subsequent phase and resolve the conflicts manually.

In Mafazi et al. (2013) we proposed a view management framework that incorporates on-the-fly change propagation for non-conflicting changes. Based on simple rules for checking behaviour consistency (Schreff & Stumptner 2002), we can efficiently identify inconsistencies caused by changes and point them out to users. Our view management framework supports all aspects of consistency management identified by Branco et al. (2013): defining consistency properties, matching model elements, checking, identifying, and fixing inconsistencies, but not conflict detection and resolution.

This paper focuses on conflict detection, propagation, and resolution of conflicting change operations applied in multiple views. The core contributions are (1) a framework for automatic conflict identification in views based on a common reference process, (2) an analysis of conflict resolution strategies that ensure behaviour consistency, and (3) an consistency restoration mechanism for conflicting attribute changes based on domain ontologies. The proposed approach has the potential to increase automation of conflict detection and resolution, support for control flow semantics without the need to analyse all possible execution traces, and support changing the process structure as well as task attributes.

The remainder of this paper is structured as follows: the next section introduces a motivating example, Section 3 discusses the view management framework and explains the main components, Section 4 covers the conflict resolution, Section 5 provides a literature review covering change propagation and conflict resolution for process modelling and model merging, and the last section provides an outlook on future work.

2 Motivating Example

Rich process models complement control flow information with a variety of entities and attributes that should be considered when making changes. For example, the fact that particular tasks in a process model are executed by a specific role or use the same data document may be used as a relevant criterion for abstracting a process model for a particular stakeholder or purpose. Figure 1 illustrates a conflicting change propagation scenario involving two
process views and a reference process model. In this process model, after issuing the renting request a car is assigned to the customer who then have the option to proceed with the booking or to cancel the request. *Process View 1* is for a stakeholder who is interested in observing those tasks which are executed by the role Customer which their execution duration exceeds 7 minutes. Likewise *Process View 2* captures only those tasks which are executed by Customer or Receptionist. Other tasks that do not satisfy the abstraction criterion are hidden in the respective process view. Each model element in a view is associated with the corresponding model element(s) in the reference process, as captured by a correspondence relation between the two models. For example, tasks *Cancel Late* and *Send Cancellation Confirmation* from the reference model are aggregated and mapped to the task Cancel in View 1.

Let us assume that User A replaces task *Use in View 1* in Figure 1 with two consecutive tasks *Drive* and *Select Drop-off Loc*. Meanwhile, User B replaces this task with two alternate tasks *Drive* and *Postpone Reservation* in process View 2. Since both changes affect the same region in the reference model and they are in conflict, a conflict situation arises as applying one change renders the other one inapplicable. Section 4 provides the solutions for dealing with such conflicts.

### 3 View Management Framework

This section introduces the main components of our framework. We adopt a process life cycle model where processes transit between the stages *process design, generating views, changing views, on-the-fly change propagation, conflict detection and conflict resolution*. A novelty of our approach is that view generation is supported by specific constraints which define how the reference model is abstracted in a view. Using predefined abstraction patterns, we ensure that the abstraction constraints are satisfied and the resulting view generated views are observation consistent Schreff & Stumptner (2002) with the reference model.

Views and the reference model are subject to change, as they must be aligned with new business rules, regulations, policies and user’s demand. Since in large organisations changes are usually performed by multiple stakeholders in parallel, conflicting changes can arise easily. In order to resolve conflicts, suitable changes must be identified and propagated to relevant models.

#### 3.1 Process Design

Although BPMN has become the de-facto standard for process modelling languages, a more formal representation is required for specifying the relationship between reference process and process views and for the verification of consistency rules. We base our framework on a labelled Petri net representation for process models, which is well-suited for expressing behaviour consistent specialisation and correspondences between models (Schreff & Stumptner 2002), and leverage BPMN only for presentation. Formal mappings from BPMN to Petri net formalisms have been described by Dijkman, Dumas & Ouyang (2008). Figure 2 shows a Petri net representation of the process model in Figure 1. The labels have been omitted for clarity of presentation.

A labelled Petri net is a Petri net that has labels attached to its arcs. Schreff & Stumptner (2002) introduced the idea of labelling arcs, which is an analogy to the way different copies of a form are handled in business processes based on paper work. Each task deals with a different copy of a form where it collects the copies from different sources and delivers them to various recipients. The labels determine which copies are used by which tasks.

**Definition 1 (Labelled Petri Net)** A tuple \((N, F, A, D, \delta_A, L, l)\) is a labelled Petri net model, where \(N\) is a finite set of nodes partitioned into disjoint sets of tasks \(N_t\) and states \(N_s\), \(F \subseteq (N_s \times N_t) \cup (N_t \times N_s)\) is the flow relation such that \((N,F)\) is a connected graph, \(D\) is a set of value domains, one for each data label. We further require that each \(D_i \subseteq D\) is a semi-lattice \((D_i, \preceq_i, v)\) with partial order \(\preceq_i\) and least upper bound operator \(\cup\). It is assumed that \(u \in D_i\) provides less information about a data property than \(v \in D_i\) iff \(u \prec_i v\).

\(\delta_A : N_t \rightarrow 2^A\) is a function mapping each task node to its set of variables. For brevity we write \(n_i.A\) for \(\delta_A(n_i)\). \(L\) is a finite set of labels, and \(l : F \rightarrow 2^L \setminus \emptyset\) is a function that assigns a label or set of labels to each control flow. Moreover, we write \(F^*\) to denote the transitive closure of the control flow relation \(F^*\).

**Definition 2 (Immediate Pre-Node and Post-Node)** The set of pre-nodes of node \(n \in N\) is defined as \(n_i = \{n_1 \in N | (n_1, n) \in F\}\). Likewise the set of the post-nodes of node \(n \in N\) is defined as \(n_o = \{n_1 \in N | (n, n_1) \in F\}\).

**Example:** \(\bullet*\text{CancelLate} = \{s_2\}\) and \(\text{CancelInvoice} \bullet* = \{s_4\}\) in Figure 2.

The execution semantics of a labelled Petri Net is the same as that of a Petri Net where a task produces a token to each of its immediate post states and consumes a token from each of its immediate pre states. In this paper we assume that all models are labelled Petri nets satisfying the properties: safe, activity-reduced, deadlock free, label preservation, unique label, common label distribution. We refer the interested readers to (Schreff & Stumptner 2002) for a complete description of these properties.

#### 3.2 Generating Views

Mafazi et al. (2012) proposed a knowledge based framework for process views based on user-specified constraints and applied the approach to a real-world business process repository from the product and system engineering domain containing software development process models with up to 260 tasks in a single process model. In some use cases, a view could be generated that reduces the complexity by up to 90%. Since views were derived using the consistency rules proposed by Schreff & Stumptner (2002), views can be seen as abstract supertypes of the more detailed reference model, which can be regarded as the intersection type of its parent views.

While there are several interesting techniques for identifying corresponding process model elements, such as different metrics and measures based on linguistic similarity of labels (Rosa et al. 2010, Pedersen et al. 2004), in our framework the correspondences between process model elements are captured while creating the views. We capture the relationship between elements in the reference process and a process view by a view-specific total mapping function \(h\) such that:

1. \(h\) maps each state, task, attributes and control flow relation of the reference process model to their corresponding entities in the view;
2. Different labels in the reference process model can be mapped to a single label in the view;
3. The initial state of the original model is mapped to the initial state of the changed model.

More formally, the correspondence between elements in the reference model and in a view can be captured as follows:
Figure 1: A conflicting situation example (BPMN)

Figure 2: A conflicting situation example (Petri net)

Definition 3 (Process View) Let M and M’ be labelled Petri net models. The components of M and M’ are understood as per Definition 1. Model M is a view of the underlying reference process model M’ if a mapping function \( h : N^0 \cup F^0 \cup L' \mapsto N \cup F \cup L \), satisfying the following properties:

1. \( n \in N' \cup F' \Rightarrow h(n) \in N \cup F \),
2. \( l' \in L' \Rightarrow h(l') \in L \),
3. \( A \in A' \Rightarrow h(A) \in A \),
4. \( A' \in N'_0, A \in N \Rightarrow h(a) = a \).

For each view \( V \), there exists a separate mapping function \( h_v \) mapping the reference process elements to the corresponding view elements. The correspondences between the models are captured in a set of mapping functions \( H = \{ h_1, h_2, ..., h_n \} \).

Example: Figure 2 shows two process views that have been generated from the reference process M’ where, for example task Cancel Late' in the reference model has been mapped to state Empty in View 2. The relationship between the reference process and this view is specified as \( h_2(\text{CancelLate}') = \text{Empty} \). Likewise the relations between the nodes of reference model and View 1 is captured by a function \( h_1 \). In the given example \( h_1(\text{CancelLate}') = h_1(s_2') = h_1(\text{SendCancellationConfirmation}') = \text{Cancel} \).

Since each element in the reference model has a corresponding element in each view, correspondences between view elements can be established indirectly via their mapping to the common reference model.

Example: In the given example in Figure 2 task Cancel Late’ from the reference model is mapped to the task Cancel in View 1, that is \( h_1(\text{CancelLate}') = \text{Cancel} \), and to task Empty in View 2. Moreover, task Send Cancellation Confirmation’ from the reference model is mapped to the tasks Cancel and Send Cancellation Confirmation in View 1 and 2 respectively. Hence, it can be concluded that task Cancel from process View 1 corresponds to the tasks Empty and Send Cancellation Confirmation from process View 2.
3.3 Changing Views

Our framework supports the following operations in each model: insert, update, delete of model entities including tasks, places, arcs, task’s attributes and labels. The changes can be applied to a process fragment which can be a single node or a group of nodes with their associated control flows. Among all the possible combinations of the changes, applying update-update, update-delete, and insert-insert of different process fragments in corresponding regions of different models may cause conflict.

Example: As illustrated in Figure 4, task Use has been replaced by the fragment containing the two consecutive tasks Drive and Select Drop-off loc in View 1, and by another fragment containing two alternative tasks Drive and Postpone Reservation in Process View 2. For replacing task Use by the fragment containing the two tasks Drive and Select Drop-off loc in View 1 in Figure 2, firstly the operator RemoveProcessFragment(Use) needs to be applied. Subsequently, InsertProcessFragment(Drive, s8, Select Drop-off Loc; s2; s3) operator needs to be applied.

After applying the changes we verify that the resulting model is well-formed, safe, activity-reduced, deadlock-free, and satisfies the label properties. These properties can be checked efficiently using techniques such as SESE fragmentation (Fahland et al. 2009). If properties are violated then the process model must be revised before we continue with change detection and resolution. In our example, no issues arise, since the models remain safe, activity-reduced, and deadlock-free after replacing task Use in Process View 1 and 2.

3.4 On-the-Fly Change Propagation

Mafazi et al. (2013) proposed a design-based approach where consistency criteria are checked and non-conflicting changes are propagated on-the-fly from a process view to its reference model and related process views. Moreover, strategies for dealing with data flow inconsistencies after change propagation were presented. The work presented here complements the previous framework by focusing on conflict detection and resolution during change propagation. In the following we describe our framework in which conflicting changes are detected, possible resolutions are inferred, and resulting updates are propagated among models.

3.5 Consistency Criteria

Branco et al. (2013) define consistency management as a set of methods and tools for establishing and maintaining consistency among software artifacts, such as models, code, documentation, and test cases which are usually created and maintained by different stakeholders.

To ensure the consistency between the process models in our framework, we use the consistency criteria defined by Schreffl & Stumptner (2002).

Observation consistent specialisation allows us to identify model elements that are defined in one view and unreferenced in another view. Moreover, elements that are defined in one view as view-specific extensions, for example view specific alternatives, that are not represented in other views can be present. Hence, the observation consistent specialisation framework provides refinement and extension rules. Informally, observation consistent specialisation guarantees that if features refined or added in the detailed model (denoted as the sub type) are ignored, any instance of the sub type can be observed identically from the point of view of the more abstract model (denoted as the super type). For precise formal definitions of extension and refinement rules we refer the interested readers to (Schreffl & Stumptner 2002, Mafazi et al. 2013).

3.6 Conflicts

According to Gerth et al. (2013) two change operations are conflicting if application of one operation renders the other one inapplicable. We follow the same definition in our approach to detect conflicts where the changes applied in different models conflict indirectly via their corresponding fragments in other views.

If a conflict arises, it is sufficient to notify the user about the conflict and to resolve the conflict by either combining the two conflicting changes or applying one of them. The rules of behaviour consistent extension and specialisation can guide the selection of suitable candidate operations. As a result, the effect of one of the conflicting changes is captured and reflected in the other relevant process models. On the other hand, if conflicting changes are combined, for example by inserting both change fragments as alternative branches, the effect of both of the changes can be reflected in the relevant models. In the following we discuss different types of conflicts and their identification and resolution strategies.

4 Conflict Detection and Resolution

Mens (2002) introduces five types of conflicts in the field of software merging: textual, syntactic, semantic, structural and behavioural conflicts. In the context of process models, we deal with structural conflicts including conflicts in the behaviour of the model before and after change. To deal with conflicting changes, Gerth et al. (2013) suggests three conflict resolution strategies: applying one, both, and none of the operators. We base our resolution mechanisms on the first and second strategies.

4.1 Type of Conflicts

Koegel et al. (2010) define two types of conflicts based on level of severity of the conflicting changes as hard and soft conflicts. For hard conflicts, the effect of only one of the changes can be reflected to the remaining model. For soft conflicts, both can be incorporated into the model. In our framework, the conflicting changes that can be combined, where one change contains the other one, or where the changes are identical in terms of structure and tasks are considered as soft conflicts. Others are considered hard conflicts, where only one change can be applied or conflicts must be resolved cooperatively.

4.2 Conflict Detection

Conflicting changes are identified based on their affected fragments in the underlying reference model. Affected regions in the reference model can be identified based on the correspondence relationship established by the mapping function h. Corresponding regions between views can be inferred based on their mappings to the shared reference model. In any case, conflicts arise if two changes affect corresponding regions in mutually incompatible ways.

Table 1 indicates different conflicting situations resulting from applying conflicting changes in two corresponding regions of two process models as well as the resolution strategy for dealing with each situation. The table outlines the type of conflict for given overlapping changes and states appropriate resolution strategies that may be applied in each situation.

Insert-Insert Conflicts: Insertion conflicts arise if different model elements are added in corresponding regions in different models. For example there would be a hard
conflict if one stakeholder inserts a parallel or a sequence fragment while another stakeholder inserts an alternative fragment such that the inserted fragments have identical tasks.

**Update-Update Conflicts:** Another type of conflict that can occur relates to updating the same attribute of two corresponding tasks in the two models. There would be a soft conflict if one change in a model can satisfy the constraints attached to both of the changed model. A hard conflict can also occur if a change violates the constraint attached to the models. For example, an update-update conflict arises if the owner of View 1 in 5 updates the role of task *Postpone Reservation* from *Customer to Admin*, and the owner of View 2 changes the execution role for the task *Postpone Reservation* from *Customer to Manager*.

**Update-Delete Conflicts:** Updating one task while deleting a corresponding task in another model causes this conflict. There would be a hard conflict if the update on the attribute of a task in a the view can satisfy the constraint attached to that view while its corresponding task is removed from another model. A soft conflict situation can occur if the update on the attribute of the view violate the constraint attached to that view. For example, assume that the owner of View 1 in Figure 5 removes task *Select Drop-off loc.* from View 1 while the owner of View 2 updates the attribute role of this task in their view to *Receptionist*.

### 4.3 Conflict Resolution

This section gives an overview on resolving operation-based conflicts between changed views.

**Insert-Insert Conflict Resolution:** Table 1 suggests four of conflict resolution strategies:

1. **Combine:** applying the changes individually in different process models result in soft conflicts, and the result of both of the changes can be captured by combining the conflicting fragments. For example, if the conflicting fragments are two alternative fragments, there is a soft conflict that can be solved by combining the two fragments. Figure 3b and 3c show a soft conflict situation. In Figure 3b a single task is inserted whereas two tasks are inserted in the other view. Likewise, in Figure 3c a sequence of two task conflicts with the insertion a parallel fragment containing three tasks. This conflict can be resolved by inserting both fragments as alternative branches in both models.

2. **Apply One:** In case of soft conflicts when the changes on the models are identical, only one of the changes is considered and propagated. In case where one of the changes satisfies the constraints of both views, a soft conflict arises that can be resolved by propagating the satisfying change only.

3. **Apply Specific Changes:** In case of a soft conflict, where one change contains the other one, the more comprehensive change is propagated. For example assume that one change inserts a parallel fragment while another one inserts a task in a parallel fragment. In this case the parallel fragment is propagated as it subsumes the other change.

4. **User Decides:** In case of a hard conflict, a change that satisfies the constraints of both views must be selected and propagated to the relevant models. If both of the changes violate the constraints of both views, the changed views must be abstracted individually such that they satisfy their attached constraint. If after further abstraction the constraints are satisfied but the change in the views are conflicting then the owners of the views decide which change to propagate.

Soft conflicts arising from updating attributes of corresponding tasks in two models are resolved by propagating a change that satisfies the constraint of all affected models, possibly further generalising the views if necessary. For hard conflicts, the authors of the conflicting changes are notified of the conflict, and if they choose to proceed, the view must be abstracted further to resolve the inconsistency. **Example:** States *s2* in View 1 and 2 in Figure 2 are corresponding, and *s3* in View 2 and task *Use* in View 3 are corresponding. Given the changes applied to each view (*RemoveProcessFragment(Use)* and *InsertProcessFragment(Drive, s8, Select Drop-off Loc.: s2; s3)* in View 1, and *RemoveProcessFragment(Use)* and *InsertProcessFragment(Drive, s8, Select Drop-off Loc.: s2; s3)* in View 2), a soft conflict situation arises according to Table 1. This situation can be resolved by combining the conflicting fragments.

**Update-Update Conflict Resolution:** In case of a soft conflict, an update that satisfies the constraints attached to each of the changed models is propagated to the relevant models. In case of a hard conflict where one or both of the changes violate the constraints, the owners of the views performing the changes are notified about the violation of the constraint and if they intend to proceed with the change, the views that violate the constraints is abstracted further such that the constraints are satisfied. Subsequently, the structural changes are propagated to the relevant models.

**Example:** Continuing the example in Figure 5, since the change in the attributes of both of the views violate the constraint attached to the views, the owners of the views are notified about the violation of the constraint and if they intend to proceed with the change, the views are abstracted such that task *Postpone Reservation* is removed from both views.

**Update-Delete Conflict Resolution:** In case of a soft conflict, the changes removing a task are propagated. In case of a hard conflict where both of the conflicting changes can satisfy the constraints attached to their individual view, if updating the attributes in one model can satisfy both of the constraints attached to the two changed views, the change that performs the delete needs to be discarded. In case the update of the attribute does satisfy the constraints attached to the two views, the change that applies delete is propagated to the relevant models.

**Example:** Removing task *Select Drop-off loc* in Figure 5 does not violate the constraint of View 1, whereas changing the attribute of this task in View 2 to *Admin* violates the constraint of View 2. Hence the deletion applied in View 1 is applied to the models.

Since our information about the intent of the owner of a view for applying a certain change is not captured in the change operators, the suggested resolution strategies for hard conflicts may not be acceptable and the stakeholders must decide which change to discard and which change to propagate.

We use a divide and conquer strategy for resolving...
conflicts caused by more than two stakeholders. For this purpose, firstly the conflicts are resolved between two models fragment by fragment. Once the two models are consistent, the conflicts between the third model and one of the consistent models are resolved. This process continues until all the models in conflict become consistent. In this way problem of resolving multiple conflicts between multiple models is reduced to resolving one conflicts between two views in each step. We employ behaviour consistent generalisation to ensure that the resulting views converge to mutually consistent processes and that the change propagation eventually terminates.

4.4 Consistency Restoration

Figure 4 continues the example given in Figure 2. After applying the changes in View 1 and 2, to restore consistency between the views, firstly the conflicts must be identified, resolved and then the changes need to be propagated to the reference model. The changed fragments in the views can be determined by the $h()$ function. If the applied changes are on the corresponding regions of the views based on the changes Table 1 suggest the type of conflict as well as the resolution strategy. Once the conflicts in the views resolved and propagated to the reference model, the two views need to be updated based on the applied conflict resolution strategy.

**Example:** As illustrated in the reference model from Figure 4 based on the correspondences, task Use* from the reference model is replaced by the combined fragment containing both changes made on View 1 and 2. Subsequently, as shown in Figure 5, the combined fragment is propagated from the reference model to the views. For this the $h$ function is applied to map each affected element in the reference model to a corresponding update in the view.

4.5 Attribute Change Propagation

One may be interested in changing the label of a task or an attribute’s value in a process view or in the reference model. Once a task’s name changes, the name of all its corresponding tasks in the other process models must be changed accordingly in order to restore consistency. If the corresponding tasks in the two models are at the same level of details, then the same name can be used to update the name of the other corresponding node. Otherwise appropriate names must be selected. For this purpose we make use of domain ontologies such as meronymy relations (Smirnov et al. 2010) available in MIT Process Handbook (Malone et al. 2003). Likewise, domain knowledge including public resources, such as WORDNET1, and clustering methods (Günther & Aalst 2007) can be used to assign appropriate attribute values to the corresponding tasks at different levels of detail. In this context, using domain knowledge and ontologies has the advantages that appropriate labels and attribute values can be assigned to the corresponding tasks which are at different levels of detail. For example, if the name of one of the two tasks in a refinement relation changes, the appropriate label can be assigned to the other task by using the domain knowledge. Similarly, ontological knowledge can be used to guide the view abstraction mechanism. Let us consider the meronymy relation for task Cancel represented in Figure 6. The relation shows that this task can be decomposed in two ways. If a stakeholder changes the name of the task Cancel to Schedule in View 1, the name of the corresponding tasks can be updated accordingly using the meronymy relations; that is, the label Cancel Late can be replaced by Schedule Later and Send Cancellation Confirmation by Reschedule Confirmation.

For other attributes, we employ a partial refinement relation that associates values with their more abstract parents in a hierarchy. We rely on a partial order defined over the set of domain values for the respective attributes and task labels for decomposition or generalisation of attribute’s values including task’s labels. Definition 1 provides a generic framework for their introduction, where suitable decomposition or generalisation relations can be defined by a partial order for each value domain associated with an attribute. Least upper bound and greatest lower bound on the attribute lattices reflect abstraction and decomposition of attributes. Figure 7 depicts an abstraction semi-lattice for attribute Role. The is-a relation connecting general roles with their specialisations forms a partial order over the set of role names.

5 Related Work

In the following we discuss related work directly addressing conflict resolution strategies in change propagation scenarios as well as approaches related to co-evolution.
of process models. The mentioned approaches complement our approach in some aspects. We first discuss conflict resolution in change propagation scenarios, and subsequently investigate conflict resolution by merging process models and consistency criteria used in the literature. Table 2 summarises our findings. The table rows specify the papers and indicate their support towards the following criteria:

Goal: This column classifies the related work based on their goal.

Support for attribute: This column indicates whether the related work considers changes in attributes and preserves their consistency. Since several real world process models have attributes attached to their tasks, and these attributes may be changed, it is important to have strategies to restore the attribute consistency. The possible values include (+) supported, (−) not supported, (N/A) the criterion is not in the scope of the paper, and (n.d.) the criterion is in the scope of the paper but it has not been discussed.

Bidirectional Change Propagation: This column indicates the support of the approach for a top down and bottom up change propagation where different levels of abstraction hierarchy have been considered by the approach. This is an important aspect since the change cannot be restricted to any particular abstraction levels.

Support for Process Views: This column indicates the support of the approach for different process views. Considering the large size of the process models, having different process views can facilitate not only the model comprehension but also aids business analysts.

Consistency Preservation: This column discusses the consistency criteria proposed by the related work in a change propagation scenario. This is an important criterion as in practice there is no single process model but a group of relevant process models. In order to preserve the uniformity between such models, the consistency between them must be evaluated against formal criteria.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Taentzer et al. (2012)</td>
<td>Conflict detection/ resolution</td>
<td>-</td>
<td>N/A</td>
<td>N/A</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Brosch et al. (2010)</td>
<td>Conflict detection/ resolution</td>
<td>n.d.</td>
<td>N/A</td>
<td>N/A</td>
<td>n.d.</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Mendling &amp; Simon (2006)</td>
<td>View consolidation by merge</td>
<td>-</td>
<td>N/A</td>
<td>+</td>
<td>n.d.</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Blanc et al. (2008)</td>
<td>Model inconsistency</td>
<td>n.d.</td>
<td>N/A</td>
<td>-</td>
<td>+</td>
<td>N/A</td>
<td>+</td>
</tr>
<tr>
<td>Küster et al. (2009)</td>
<td>Conflict resolution</td>
<td>-</td>
<td>+</td>
<td>N/A</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Weidlich et al. (2012)</td>
<td>Change propagation</td>
<td>n.d.</td>
<td>+</td>
<td>N/A</td>
<td>+</td>
<td>N/A</td>
<td>+</td>
</tr>
<tr>
<td>Cicchetti et al. (2008)</td>
<td>Conflict resolution</td>
<td>+</td>
<td>N/A</td>
<td>n.d.</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

Legend: + supported, − not supported, N/A not applicable, n.d. not discussed.

**Conflict resolution:** This column indicates whether the proposed conflict resolution strategy is structured-based, based on execution traces, or on change logs.

**Support for behavioural models:** This column discusses whether the related work supports behavioural model, such as BPMN and Petri nets, or if it only supports the structural models such as UML class diagrams. Considering the behaviour of the process model while dealing with conflicts helps identifying potential false-positive conflicts, that is semantically equivalent changes which are conflicting (Gerth et al. 2013).

### 5.1 Conflict Resolution in Change Propagation

It is important to identify the conflicts and dependencies of change operators in a process model before applying changes. That is if two change operators are conflicting then only one of them should be applied, or if two changes are dependent then one change needs the application of the second one. Küster et al. (2009) investigate dependencies and conflicts of changes in process models. They encode critical pairs, used for detecting dependent and conflicting transformations, as conditions for the change.

Gerth et al. (2013) represent a method for identifying semantic and syntactic conflicts in model versioning scenarios. For this purpose they use process model terms to capture the execution order of the elements in a process model and three-way merge. To identify conflicts they assume that all the changes are provided in the form of change operations, including insert, delete and move, stored in change logs.

Küster et al. (2009) use change logs to identify conflicting change operations. They suggest three options to resolve a conflicting situation: discarding one of the complete subsequence, combining the conflicting operations and modifying one or both of the conflicting operations. An appropriate option must be identified and selected by the user.

In contrast to our approach, these works assume that all the changes are recorded in change logs while building a version from the original model. Recording and reasoning about change logs can be expensive in large process models especially if composite operators such as replacing or swapping process fragments are applied. Moreover, building process model terms to capture the execution order of the elements in a process model to establish correspondences between model elements can be expensive considering the large size of process models in the real world scenarios.

Dam et al. (2010) propose a semi-automated change propagation technique for UML class diagrams based on Alloy facts. The technique provides repair plans for resolving conflicts and inconsistencies. Although conflicting situations and resolutions are not directly discussed, such are included in their proposed repair plan. Hermann et al. (2012) propose a semi-automated conflict resolution approach in synchronising the concurrent model modifications based on rules expressed as Triple Graph Grammars.

Cicchetti et al. (2008) propose a domain specific language to control the conflicts from cooperative changes over the same process model elements. This semi-automated approach is based on a meta model representation which enables detection of semantic and syntactic conflicts. To represent the conflicting modifications, the changed models are merged and the appropriate conflict resolutions are recommended.

Altmanninger (2008) introduces the conceptual design of the SMoVer version control system. The system provides information about the models’ execution semantics which facilitates better conflict identification than what purely syntactic and structural approaches can offer. The focus of the framework is on detection of conflicts; their resolution is left to the user.

Küster et al. (2012) propose an approach to synchronise process views at different abstraction levels through a shared process model. The shared process model is initialised by a single process view, as it is assumed that different process views evolved from a single model. If one of the views changes, it is checked into the shared process model to update other views. This approach synchronises the views but not the reference model, and the approach assumes that changes to different views do not happen concurrently.

Taentzer et al. (2010, 2012) define two syntax-based conflict notions, operation and state-based conflicts, in model versioning based on graph theory. Operation-based conflicts include conflicts such as adding a node in one version and removing it in another version. By giving the priority to the add this conflict is resolved. State-based conflicts occur where the final state of the graph after applying the changes contradicts consistency rules specified in the modelling notation. This categorisation has been elaborated by Taentzer et al. (2012) for EMF-based models, while the modelling features of the models such as controlling, multiplicity, and ordered features have been formalised by graph constraints. Conflict patterns are presented which illustrate the conflicting operations, such as delete-use, delete-move, delete-update, update-update, move-move, and insert-insert. For resolving any of these conflicts a strategy has been presented. However it is not clear how changes are set to a pre-defined change operation. This can cause difficulties in the identification of the potential operations especially where the investigated versions differ significantly.

Weidlich et al. (2012) use behavioural profiles for...
identifying changed regions in a model and propagate the changes between models at different levels of abstraction. In this approach data flow and other task properties are not considered. Moreover, the approach depends on the existence of execution traces of a process model.

5.2 Conflict resolution in model merging

Brosch et al. (2010) introduce a model versioning system called AMOR. This tool provides an exact conflict report in model merging as well as semi-automatic executable resolution strategies. For this purpose the predefined resolution patterns based on the UML class diagram recommend a resolution for conflicts. In this framework consistency preservation is not discussed, and neither the execution semantic of the modelling notation nor its behaviour can be captured. In contrast, our approach is language specific and considers the execution semantics and behaviour of the process model.

Mendling & Simon (2006) propose a method for integration of model views expressed in conceptual modelling notations such as EPC. Firstly the semantic relations between views are derived based on the relations between the model elements, where two elements are either equivalent or in sequence. Secondly a merge operator creates the merged model by taking the semantic relationship and the EPC models as input. Lastly a set of restructuring rules are applied on the merged model to eliminate any unnecessary model elements. In contrast, we assume that the semantic relationship has been established earlier when the views are created. Moreover in our approach the observation consistency across the models is preserved in all the stages.

5.3 Consistency management in process modelling

Weidlich & Mendling (2012) discuss the notion of view consistency, which is concerned with achieving and preserving a certain degree of uniformity and consistency between views created by multiple stakeholders as well as a reference process model. In this context, models are consistent if they cover exactly the same part of a scenario and there are no behavioural contradictions between them. As the process models are changed frequently, having an appropriate consistency notion can help propagating changes from one model to another.

Dijkman, Quartel & van Sinderen (2008) address issues arising in information system design where several stakeholders collaborate to create a system. Their framework establishes and supports a common terminology for stakeholders and consistency management mechanisms between different views. The paper considers observation and weak invocation consistency, and deals with models that are either in refinement or overlap relations.

Blanc et al. (2008) represent an approach to detect (structural and methodological) state-based model inconsistencies expressed as logic formula. Structural consistency rules enforce the relationship between model elements of two models without considering how models have been constructed. Methodological consistency rules enforce an appropriate order of editing operations on a model. The consistency rules are based on domain-specific use cases and compare the previous state of the model with the state resulting from changes. Mens et al. (2006) address inconsistencies resulting from change propagation by introducing a framework for inconsistency management for UML class diagrams based on graph transformation rules. After applying changes, the model is evaluated against the graph transformation rules to determine conflicts, and resolution rules are applied to eliminate conflicts. The approach can identify syntactical conflicts such as contradicting changes on the meta models and violation of OCL constraints, although semantic conflicts of the models and behaviour inconsistency are left aside.

Perry et al. (2001) notes that configuration management systems are only able to identify conflicts resulting from applying changes in one version which overlaps with some other changes in another version. However, no criteria for view abstraction or change resolution have been proposed.

Table 2 illustrates a comparison of the discussed approaches. Although our framework can support all the criteria stated in this table, it is limited in terms of supporting certain change patterns discussed by Weber et al. (2007). In particular moving and swapping process fragments are not supported, as these patterns violate support the observation consistency rules underlying our framework.

6 Conclusion and Future Work

We considered the detection and resolution of conflicting changes in view management of business process models. We identified shortcomings of existing approaches in resolving conflicts such as relying on execution traces analysis, support for static models only rather than dynamic models, and dependence on change logs in a post-analysis phase. To overcome these shortcomings, we presented a framework for on-the-fly change propagation and extended it with selected conflict resolution strategies. We showed how behaviour consistency rules for checking the consistency can be used for conflict resolution and restoration of behavioural consistency among a set of related views. Our initial efforts have revealed that considering the purpose and intention of view can help to resolve soft- and some hard conflicts automatically. The validation of our approach by integrating the conflict resolution and change propagation with our tool for generating views from a reference model is subject to future work.

References


Applying a Test for Atomicity of Method Fragments

Ben Rogers¹ and Brian Henderson-Sellers¹

¹University of Technology, Sydney, Broadway, NSW 2007, Australia
brian.henderson-sellers@uts.edu.au, benrogers@iinet.net.au

Abstract

One aspect of the conceptual modelling of processes is their quality. Here, we examine one aspect of quality – atomicity – as evaluated for a number of method fragments. High quality fragments will increase the quality of software development process models by application of the tenets of situational method engineering. Here, we identify a number of fragments from a previously developed methodbase/repository as being potentially non-atomic and suggest possible revisions to increase their quality.

Keywords: Metrics, Method Fragments, Situational Method Engineering, Atomicity

1 Introduction

For software development processes to be successful in their enactment, they need to be modelled conceptually, primarily in terms of a process model but also with respect to the underpinning metamodel. The elements in such a design-level process model need to be conformant to a metamodel and, at the same time, provide the templates needed for the creation and enactment of processes enacted on a specific project. Here, we examine these process models in the context of situational method engineering (SME).

Situational method engineering relies on small elements of methods being available from which a complete methodological approach can be constructed e.g. Henderson-Sellers and Ralyté (2010). Although there are many ‘flavours’ of method parts, we focus here on method fragments e.g. Brinkkemper (1996), which are generally described as being ‘atomic’ parts of a method.

One approach that is based on the use of method fragments is that of the OPEN Process Framework (OPF: Graham et al. 1997, Firesmith and Henderson-Sellers 2002), which, in turn, employs the Software Engineering Metamodel for Development Methodologies: SEMDM (ISO/IEC 2007). Although the original OPF publications presented method fragments that were atomic, subsequent modifications and revisions led to some of them growing to such a size that their atomicity could be challenged. Indeed, Henderson-Sellers and Gonzalez-Perez (2011) presented an analysis based on the granularity of method fragments (Hobbs 1985, Mani 1998), a theory based in turn on abstraction theory (Giunchiglia and Walsh, 1992, Kaschek 2004, Keet 2007), and concluded that at least one Task fragment (Construct the object model) could no longer be recognized as being of an atomic nature. However, these authors did not use any measure to determine objectively whether this particular fragment (and others) are or are not atomic in nature. The current study aims to define some metrics that can be applied to method fragments to assess their quality. We focus on only one aspect of quality: whether or not fragments are atomic. That is, can the fragment be broken down into smaller entities? In this project only the fragments constructed from the OPF/SEMDM methodology framework will be assessed to illustrate the appropriateness of the proposed metric whilst recognizing its more global applicability i.e. to fragments from sources other than the OPF. Once some metrics have been defined, they will be applied to existing fragments in the OPF repository. This is not only to measure the quality of the fragments but also to verify that the metrics do fulfil their purpose. One of the objectives of the project is to provide a means of evaluating the quality of new fragments, so that fragment authors have a tool to help them design fragments that are of good quality. This paper describes the metrics and also the principles behind them, so that fragment designers can understand what makes a fragment atomic.

In Section 2 we describe what is meant by the atomicity of method fragments. Section 3 discusses software engineering metrics and how we might calculate appropriate metrics for determining the atomicity of method fragments. In Section 4, we present the results of applying the atomicity calculations to a large number of fragments conformant to the OPF and/or SEMDM metamodels. Section 5 concludes, together with suggestions for future research on this topic.

2 Fragment atomicity

An atomic fragment is one that is not made up of other fragments, i.e. it has fine-grained granularity (Henderson-Sellers and Gonzalez-Perez 2011). A fragment that is not atomic is more likely to be usable in a limited set of situations, as it will not have the flexibility to be reused in a wide variety of circumstances. Furthermore, if fragments are not atomic, the functionality of one fragment could overlap that of another fragment. The benefits of maximising fragment reuse include the following. The more a fragment is reused, the greater the chance it will become expert in what it does. This is because it will be used in a wide variety of situations with different demands placed on the fragment. As the fragment is widely reused, all the uses of it will benefit from the increased expertise. Furthermore, the reasons for having atomic fragments are similar to why a software class should have high cohesion. A class with high
cohesion enjoys the property of togetherness, while one with low cohesion could be broken down into two or more classes (Henderson-Sellers 1996). The elements of a class with high cohesion work together in a consistent purpose (Booch 1993). For example, a highly cohesive Car class contains the behaviour for a car and nothing but a car. The class with high cohesion is more likely to be able to be reused because of its targeted purpose. Also a class with high cohesion can be expert in its functionality because of its tight focus. The same can be said about atomic fragments in SME. They are more reusable and they can be refined to be expert in their purpose.

3 Metrics for fragment atomicity

A metric “is the mapping of a particular characteristic of a measured entity to a numerical value” (Lanza et al. 2006, p.11). An example of an entity could be a person and the characteristic could be the height of the person. Although it is possible to measure a wide range of attributes such as height, weight, age, sex of a person, the measurements only have meaning if there is a clear purpose to the exercise. For example, the purpose may be to determine if the person would be a good athlete.

Basili and Rombach (1988) proposed the Goal/Question/Metric paradigm to refocus on the measurement goal rather than simply a procedure to count something with no goal in mind. The first step is to define the goals that the use of the metrics will achieve. From the goals, a list of questions that need to be answered for the goals to be met are compiled. For each question, the metrics are chosen that will answer these questions. Two well-known metrics are coupling and cohesion (Stevens et al., 1974).

An atomic fragment is like a class with high cohesion and low coupling (as noted above). A class with low cohesion implies that it should be divided into multiple classes, and, conversely, one with high cohesion should not be divided (Henderson-Sellers 1996). Thus, an atomic fragment can be identified because of its high cohesion value. Secondly, a class with low coupling makes it easier to reuse and is also indicative of high cohesion or atomicity. Coupling could mean object-to-object coupling only or could include coupling due to inheritance. For atomic fragments, the interest is restricted to object-to-object coupling, which is simply called coupling hereafter. That is, two classes are coupled if the methods in one class make use of methods or instance variables in the other class.

A class has high cohesion if its methods work towards one purpose that is easy to identify (Henderson-Sellers 1996). Cohesion in a class is often measured in terms of its complement: the ‘lack of cohesion’ e.g. Chidamber and Kemerer’s (1991) LCOM. This metric looks at each method’s use of the class instance variables. The use of instance variables by methods determines the intersections of methods. If two methods access completely different instance variables then probably they should belong to different classes. However, there has been an extensive debate on the precise definition of LCOM and the mathematics to calculate it (Henderson-Sellers et al. 1996). The problem with various formulae for the Lack of Cohesion metric is that it may not be possible to discriminate between dissimilar entities. That is, two classes with obvious differences in cohesion may be given the same score by the LCOM metric. Also, although a high LCOM score would indicate low cohesion, a score of zero does not necessary indicate high cohesion (Henderson-Sellers et al. 1996). Despite the problems with LCOM, the study of it does help in understanding the nature of class cohesion.

Another metric to measure cohesion is Tight Class Cohesion (TCC): “the relative number of method pairs of a class that access at least one common attribute of that class” (Lanza et al. 2006, p.17). Its value is between zero and one, where a low value indicates low cohesion and a high value indicates high cohesion.

There are also several existing metrics for coupling. The coupling between objects metric CBO (Chidamber and Kemerer 1994) applies to a given class. It counts the number of classes that are coupled to the given class. Fan-out for a given class measures the count of classes that the given class makes use of (Henry and Kafura 1981). Fan-in for a given class measures the count of classes that makes use of the given class. A high fan-in value is desirable because it indicates that the given class is being reused extensively, while a low fan-out value is desirable because that shows that the given class does not need too many other classes to operate.

3.1 Metrics for fragment relationships

There are, at least, three methodological approaches that have a common semantic core: the open Process Framework (OPF) (Graham et al. 1997, Firesmith and Henderson-Sellers 2002), SEMDM (ISO/IEC 2007) and the Software Process Engineering metamodel (SPEM) (OMG 2008). All three identify three critical areas: work units, work products and producers – although here we focus on examples of fragments conformant primarily to SEMDM.

Henderson-Sellers and Gonzalez-Perez (2011, p56) have argued that the “Construct the object model” task fragment described in the OPEN Process Specification is not atomic because more than one technique had to be chosen out of thirty seven techniques. Given this precedent, of counting the number of techniques for a task fragment to determine whether a fragment is atomic or not, we postulate that other relationships could also be used. Genero et al. (2005) used metrics to measure the complexity of a Software Process Model conformant to SPEM. For activities there are work product and role counts, while for work products there are activity counts where the work product is consumed or produced, and for roles there are activity counts for which they are responsible. Genero et al. (2005) carried out some experiments to check the validity of the metrics thus defined. In the first experiment, ratings made by students and professors on a set of software process models were compared with the results of the measurements performed on the software process models. The null hypothesis was that there was no significant correlation between the structural complexity metrics and the students’ and professors’ ratings. The ratings included understandability, analysability and modifiability of the software process models. For some of the metrics that
applied to the software process model, the null hypothesis was rejected (Genero et al. 2005).

It can be the case that a number of measurements are needed for answering only one question (Lanza et al. 2006). That is the case for this research. For example, to answer the question of whether a given Task fragment is atomic there are the Technique Count, Work Product Create Count, Work Product Update Count, Work Product Read Count, Role Count and Team Count metrics. Looking at relationships between fragments not only gives a view on coupling, it also indicates cohesion – does the fragment have more than one purpose?

Just as different fragment types have different relationships with other fragments, different metrics are applicable. In the SEMDM metamodel, producer fragments have relationships with work unit and work product fragments; so, for producer fragments, metrics that count related work units and work products would be used. In contrast, work unit fragments have relationships with producer and work product fragments; so, for work unit fragments, metrics that count related producer and work products would be used. To facilitate identification of pairings, a matrix was constructed, with the fragment types listed vertically and the metric types listed horizontally.

All the metrics used here are counts of distinct related entities. For example, the role count is the number of distinct producer roles related to the fragment type. The relationship between work unit fragments and work product fragments is defined by the use of Actions, which define whether work products are created, updated or read only. The proposed metrics are divided by these action types. For instance, there is not just a Work Product Count, there is also a Create Work Product Count, Update Work Product Count and Read Work Product Count.

On producer fragment counts, only roles and teams are included. This is because, at the repository level, persons would not be defined. A role is only included in a role count if it participates directly in a Work Performance; as opposed to roles that are in teams and the teams that are in the Work Performance.

As a metric for atomicity, we propose here counts of relationships between each fragment and roles, teams etc.

3.2 Calculating the atomicity metric

It is important that measurements for metrics are easy to calculate. If possible, it would be good for these to be calculated automatically. To achieve this, a database has been created to capture the repository of fragments and their relationships. Then, SQL queries calculate the measurements of metrics for all of the fragments. The database schema and measurement SQL queries have been constructed to cover all the elements in the SEMDM metamodel.

3.3 Setting thresholds

Metrics themselves give no information on model quality. They can be used indicatively if appropriate thresholds are included (e.g. Szentes and Gras 1986, Kitchenham and Linkman 1990). Thresholds can be used to specify regions, so that conclusions can be made from the data with respect to an appropriate threshold regarding atomicity/non-atomicity. Although these can only be statistical conclusions, the usefulness is increased if the underlying statistical distribution is known or can be assumed (Haynes and Henderson-Sellers 1997). Setting two thresholds is useful: the first to indicate a “watch” if exceeded and the second (higher) one to flag as statistically likely to be an outlier. Explicable thresholds are set based on good arguments and are rarely perfect (Lanza et al. 2006). Some thresholds can be determined by generally accepted knowledge. For example, people expect to eat three meals a day. Another way to set thresholds is through statistical measurements. For example, is ten thousand hairs on a head a lot of hair or is the person balding? The number of hairs could be counted on a large population of people and an average calculated. If the average is between eighty thousand and one hundred and twenty thousand then we can conclude the person is balding (Lanza et al. 2006).

For the case of a normal distribution, the average and standard deviation are useful values. However, it is expected that the distributions for the proposed metrics will not only be discrete (i.e. the normality of the distribution can only be approximate) but are also likely to be skewed to the right since there will be no values less than zero and most of the values will be in a low range with only a few high values. This means that the numerical summary needs to be resistant (Sullivan 2011). That is, the value of the numerical summary does not change much if an extreme value is added (Sullivan 2011). For example, say there are five observations: 179, 201, 206, 208 and 217. The median is 206 and the mean is 202.2.

Now if an observation of 1000 is added the mean increases substantially to 335.1666 while the median is 207. Since the underlying data here are discrete, the median is the more useful.

The median is the fiftieth percentile. The percentile gives the position of an observation within a set of data. Quartiles are special cases of percentiles. They break the set of data into four pieces. Like the median, quartiles and percentiles are resistant numerical summaries. The interquartile range, IQR, measures the range of values between the first quartile and the third quartile; that is, the range of the middle fifty percent of values. Outliers are unusually very low or very high values. To identify outliers, fences are calculated. Fences are thresholds used as boundaries so that outliers can be found. If a value is lower than the lower fence or higher than the upper fence then it is deemed to be an outlier (Sullivan 2011). The formulae for the fences are

\[
\text{Lower fence} = Q_1 - 1.5 \times \text{IQR} \\
\text{Upper fence} = Q_3 + 1.5 \times \text{IQR}
\]

(Sullivan 2011, p.160)

In this research, since the distribution of the measurements will be right skewed with no negative values, the outliers will only be unusually high values. It is important to determine whether a measurement is an outlier or not, because measurements that are outliers will indicate that the relevant fragment is probably not atomic. That is, the upper fence of each metric’s distribution will be used as the threshold for fragment atomicity.
Figure 1: Sample fragment distribution using 20 fragments

Figure 1 provides an example of an artificially constructed distribution of fragments over some metric counts. Each fragment has an integer metric count value. This is the case for all metrics proposed in this paper. In this example, there are twenty fragments that have measurements between one and four. The distribution is right skewed. The median is given by the value at the \((n+1)/2\) position when \(n\) is odd or by the average of the values at the \(n/2\) and the \((n+1)/2\) positions when \(n\) is even. Thus, for \(n=20\), the median is the average of the 10th and 11th values = \((1+2)/2 = 1.5\).

Q1 (25th percentile) is the average of the 5th and 6th values = \((1+1)/2 = 1\)
Q3 (75th percentile) is the average of the 15th and 16th values = \((2+2)/2 = 2\)

Hence the IQR = Q3 – Q1 = 2-1 = 1
Then the upper fence = Q3 + 1.5*IQR = 2 + 1.5*1 = 3.5
Outliers, in this case, are thus metric counts of four or above.

Figure 2: Box plot for sample fragment distribution of Figure 1.

Figure 2 shows the box plot diagram for the Sample Fragment Distribution of Figure 1. The first quartile, median and third quartile are shown on the box. The square bracket indicates the upper fence, and the horizontal line either side of the box indicates values that are not outliers.

4 Results and discussion

For the first assessment, fragments were sourced from Tran et al. (2009a) for Producer and Work Product and from Tran et al. (2009b) for Work Unit fragments, supplemented by fragments from the sample repository provided with the software tool MethodMate, supplied by Cesar Gonzalez-Perez.

Table 1 lists Tasks wherein the Technique Count for the Task: Software Coding in Agile is an outlier (a value of 4; upper fence value of 3.5). What is interesting about Agile software coding is that it involves more than traditional coding; for instance, Pair Programming involves two developers working together on the same task at the same computer (Beck 2000).

The Create Work Product Count for the Initiation Process is also an outlier (Table 2). It is reasonable that the Initiation process creates a number of artefacts, because of the planning involved in the process.

We also examined counts for Stage Types, Work Product Types and Producer Types but were unable to identify any other potentially non-atomic types.

The metrics do not prove atomicity, they only flag the possibility that the fragments may not be atomic. For the candidates identified above (Tables 1 and 2), one must determine what action to take to ensure atomicity of these tasks. For example, for the Task: Software Coding in Agile, one solution may be to split the task into three: 1) Coding plus Pair Programming and Collective Ownership (as two techniques); 2) Integration Coding as a task together with an associated technique.

What is encouraging about the metrics is that the results in this paper have identified fragments that may not be atomic. However, the metrics would be more powerful if there was a larger repository of fragments in the database. Furthermore, the fragments in the database need to have a rich assignment of relationship fragments such as WorkPerformance, Action and TaskTechniqueMapping. Also it would be useful for the fragments to make use of the aggregation relationships specified in the metamodel. The more information there is, the greater the chance of resultant outliers correctly identifying fragments that are not atomic.

It proved not to be too onerous to load fragments into the database and then run SQL queries to calculate the measurements. That is, the metrics were easy to measure once the fragments had been loaded into the database. Furthermore, there is scope to introduce new metrics, simply by crafting new SQL queries.

5 Conclusions and future work

5.1 Conclusions

In the context of situational method engineering and process models, we have examined the atomicity of a large number of method fragments that are conformant to the SEMDM metamodel. Whilst most are found to be atomic, our metric counts and thresholds employed suggest that a small number of these fragments require detailed scrutiny since there is a high likelihood that they are not truly atomic in nature. Consideration is therefore to be given to either splitting tasks into more than one
task or converting multiple techniques to subtasks plus associated techniques such that tasks and subtasks are all atomic in nature.

5.2 Future Work

A useful contribution to this research would be to revise the existing fragments by associating them with relationship fragments. Increasing the sample size in the database will improve the reliability of the results (Sullivan 2011). That is the determination of the upper fence will be more accurate.

Also useful would be to undertake further research on the atomicity of fragments that have a whole-part relationship with other fragments. This impacts the construction of the SQL queries that perform the measurement calculations.

A useful enhancement to systems like MethodMate, which builds and uses fragment repositories, would be to store fragment information in a relational database. This is so that the system can automate the measurement of the metrics proposed in this paper.

References


Genero, M., Piatinni, M., Calero, C. & Ebrary, I. (2005), Metrics for software conceptual models, Imperial College Press.


Stevens, W.P., Myers, G.J. and Constantine, L.L., 1974, Structured design, IBM Syst. J., 13(2), 115-139.


### Table 1: Counts for a number of SEMDM-conformant Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Source</th>
<th>Create Work Product Count</th>
<th>Modify Work Product Count</th>
<th>Role Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyse requirements</td>
<td>MethodMate</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>MethodMate</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Develop class models</td>
<td>MethodMate</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Document requirements</td>
<td>MethodMate</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Elicit requirements</td>
<td>MethodMate</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Perform peer review</td>
<td>MethodMate</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Unit-test code</td>
<td>MethodMate</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Write user stories</td>
<td>Agile Method</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Explore architectural possibilities</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze technologies</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Describe application</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype the architecture</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop release plan</td>
<td>Agile Method</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Monitor Work Products</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop iteration plan</td>
<td>Agile Method</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Software Coding in Agile</td>
<td>Agile Method</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Design agile code</td>
<td>Agile Method</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Refactor</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing tasks</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Integrate software</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Write Manuals</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Shared Artefacts</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify shared artefacts</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate shared artefacts</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify Permissions to Shared Artefacts</td>
<td>Agile Method</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mediate/monitor the performance of team’s tasks</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mediate/monitor team’s interactions</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict management</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor members’ performance</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Member motivation</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ensure workload balance</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify team policies</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify team structure</td>
<td>Agile Method</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**Upper fence value**: 3.5 2.5 3.5 3.5

### Table 2: Counts for a number of SEMDM-conformant Processes

<table>
<thead>
<tr>
<th>Process</th>
<th>Source</th>
<th>Create Work Product Count</th>
<th>Modify Work Product Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Level Modelling</td>
<td>MethodMate</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>MethodMate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Low-Level Modelling</td>
<td>MethodMate</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>MethodMate</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>MethodMate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Initiation</td>
<td>Agile Method</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Construction</td>
<td>Agile Method</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Delivery</td>
<td>Agile Method</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>Agile Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team Management</td>
<td>Agile Method</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Upper fence value**: 2.5 4
Data-driven Requirements Modeling: Some Initial Results with i*

Aditya Ghose1, Metta Santiputri2, Ayu Saraswati3, and Hoa Khanh Dam4

Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong, NSW 2522 Australia
Email: {aditya1, ms8042, sa7833, hoa4}@uow.edu.au

Abstract

Requirements acquisition is widely recognized as a hard problem, requiring significant investments in time and effort. Given the availability of large volumes of data and of relatively cheap instrumentation for data acquisition, this paper explores the prospect of data-driven model extraction in the context of i* models. The paper presents techniques for extracting dependencies from message logs, and for extracting task-dependency correlations from process logs. The preliminary empirical results are encouraging.

Keywords: Requirements acquisition, data-driven model extraction, i* model

1 Introduction

The connection between requirements and the data that might be used to generate, understand and analyze requirements has been largely ignored in the literature. Yet the growing ubiquity of data, the ability to access large-scale sensor instrumentation and the availability of “big data” tools has thrown up significant opportunities for developing a new generation of data-driven requirements engineering (RE) tools. These opportunities come in many forms.

First, data can alleviate the well-known challenges associated with requirements acquisition/elicitation [15]. Organizations are often unable to leverage the benefits of conceptual modeling and the principled use of enterprise architecture because of the (often steep) investment required. The phenomenon is an instance of the knowledge acquisition bottleneck - a problem with an even longer pedigree [1]. Conceptual modeling is a time consuming human task of considerable complexity. We will argue in this paper that developing a capability to “mine” requirements from data can pay rich dividends. Earlier work [4] suggests that tools that extract “snippets” of models (or proto-models) by mining legacy text and model artefacts (these latter being in different notations) can significantly improve modeller productivity (with some empirical results pointing to about a two-thirds reduction in modeller effort).

Second, data-driven requirements monitoring provides the ability to improve the quality of requirements specifications, which in turn lead to improvements in the quality of the systems delivered. Execution data provides the basis for extracting requirements (which may be viewed as abstract descriptions of the data that these are mined from). Deviations between the mined requirements and those originally specified by stakeholders can flag problems. Similarly, we might cluster data associated with the particularly “desirable” (as determined by stakeholders) parts of execution histories, and extract requirements from these. The requirements thus obtained would represent more accurate encodings of stakeholder intent.

Third, clustering data associated with “undesirable” instances of execution histories (again, determined by stakeholders) can help us mine requirements anti-patterns. Within the context of a given RE exercise, these anti-patterns would identify “no-go” areas (i.e., requirements that lead to undesirable consequences).

Finally, the ability to establish an online, real-time correlation between requirements and data can help us use requirements models as dashboards.

What we have outlined above are effectively four distinct hypotheses about how data (specifically, behaviour histories) can deliver value in the requirements engineering exercise. In this paper, we put the first two of these hypotheses to the test. We focus on a well-regarded early-phase requirements modeling language of long standing - the i* notation [20]. The use of i* makes the case for data-driven requirements engineering more compelling, for several reasons. i* is particularly effective in modeling high-level strategic requirements, and also supports distributed goal modeling. Consequently i* serves as a natural representation of complex organizational contexts. This paper presents some initial steps toward an evaluation of mining techniques that permit us to mine i* models from execution data. We restrict our attention to mining dependencies and the mining dependencies and the tasks within the depender and dependee actors that each dependence is associated with. We present two techniques: the Dependency Extraction (DE) technique which mines dependencies from message logs and the Task-Dependency Extraction (TDCE) technique which mines the tasks/goals in an i* SR model that are associated with each dependency from process logs.

Our focus on message logs and process logs is realistic. Message logs are routinely maintained within the enterprise context. Sometimes, these manifest as email repositories, but our current work does not address the deployment of sophisticated NLP techniques that would be required to mine these. Instead, we use an abstract, generalized messaging format that resembles a number of industry-standard electronic messaging formats such as RosettaNet [17], ebXML [5] and a host of EDI formats. These are clearly easier to mine for data acquisition. We explore a number of approaches to data mining. We consider initiating with “snippets” of data generated from a small set of well known i* models, and then mining data from (possibly) larger corpora of i* models. We use requirements models as dashboards.


correlation between requirements and the data can help...
than natural language message logs, but nonetheless provide a useful basis for a proof-of-concept tool. Process logs are also routinely maintained by firms. A variety of business process management tools as well as bespoke process logging tools can be leveraged to obtain these. Unlike process mining tools, however, we do not seek to extract process designs from process logs, but instead mine for patterns of task activations that point to the existence of a dependency.

We also simplify matters by assuming that the only i* models of interest are those that involve only goal dependencies. We keep softgoals entirely outside the purview of our current discussion. Techniques for mining other types of dependencies represent an important direction for future work.

These techniques only support the mining of partial i* models, specifically inter-actor dependencies, and tasks/goals associated with each dependency. We present two different evaluations of these techniques. First, we evaluate their effectiveness (in terms of precision and recall) in mining partial i* models from behaviour histories that simulate the execution of an initial complete i* model. Second, we validate the hypothesis that it is possible to generate better quality (i.e., more accurate) models by mining behaviour histories of perfect “as-is” models filtered by stakeholders (to remove behaviour traces that are undesirable). Much more detailed evaluation is possible, and is the focus of future work, but our preliminary results are encouraging.

The rest of this paper is organized as follows. The next section provides background on the i* notation. The following two sections describe the DE and the TDCE techniques respectively, in detail. We then provide an evaluation of this approach, in which we separately evaluate the DE and TDCE techniques and then bring them together in evaluating how user input into filtering behaviour histories can lead to more accurate models. We then provide a brief discussion of related work before presenting concluding comments and directions for future work.

2 Background

i* [20] is a well-known requirements modelling language which describes the organizational context of an information system based on the notion of intentional actors. In modeling an actor in i*, we specify its goals, the means available to achieve these goals and how other actors depend on it to achieve their goals. Actors depend on each other for goals to be achieved, tasks to be performed, resources to be furnished and performance measures to be optimized. Such dependencies are described in an i* strategic dependency (SD) model. There are four types of strategic dependencies that may be specified in an i* model. A goal dependency models situations where an actor depends on another actor to achieve a goal. A resource dependency exists when an actor relies on another actor to provide a resource. A task dependency suggests that an actor needs another actor to carry out a task. Finally, softgoal dependencies capture the non-functional properties of a model. As noted above, we focus only on goal dependencies in this paper (and argue that all other dependencies, except soft-goal dependencies, can be reduced goal dependencies in form or another).

An i* strategic rationale (SR) model describes internal interactions between goals and tasks within each actor. Specifically, it shows how a task can be decomposed into subtasks, subgoals, resources and softgoals (i.e. they need to be performed or satisfied in order for the task to succeed). In addition, it describes means-ends links, which describe alternative ways to achieve a goal. It may also describe how tasks contribute to achieving softgoals (positively or negatively). Figure 1 shows an example of an i* SR model for a meeting scheduler system which we adapted from [20]. There are three actors here: Meeting Initiator, Meeting Participant and Meeting Scheduler. There are a number of dependencies between the actors. For example, the Meeting Initiator depends on the Meeting Participant to achieve the goal of Attends Meeting. The SR model also shows the exact tasks that are involved in a dependency (these are of particular interest in this paper). For example, the task Obtain AvailDate of actor Meeting Scheduler depends on the task Find Agreeable Date Using Scheduler to attain goal EnterAvailDate. In the next sections, we will describe how we mine existing message logs and process logs to extract those dependencies between actors and their tasks.

3 The Dependency Extraction (DE) Technique

The Dependency Extraction (DE) technique is intended to mine message logs for i* dependencies, and is based on the following intuitive observations. All dependencies manifest themselves in messages, such as a request from the dependee to the dependee at the creation of a dependency, and a message in the reverse direction when the dependency is fulfilled. Hence, a message log that maintains a record of all messages (over a certain period) between the actors of interest represents a rich repository of clues about these dependencies. Message logs are ubiquitous. A corporate email repository can be viewed as a message log, although the messages are entirely unstructured. In many cases, messages are structured such as in a variety of Electronic Data Exchange (EDI) languages, or in more recent standards such as RosettaNet and ebXML. Our current approach assumes a structured message log. We use a generalized message format in our evaluation, inspired by (and representing the common core of) the messaging standards discussed above. For our purposes, a message log is a sequence of messages consisting, at a minimum, the following components:

- An interaction ID, which is used to identify a conversation or interaction, but not an individual message.
- Sender ID
- Receiver ID
- A timestamp which describes the time when a message is sent or received (we assume message transmission to be instantaneous).
- A message type, which would involve types such as requests, responses etc.
- A message payload, consisting of the semantic content of the message (which might be imperative or descriptive or a variety of other speech acts).

In the spirit of RosettaNet, we assume that all messages that involve responses to an initial message (that starts a conversation, such as a service request) refer to a unique ID generated by the initial message. We shall refer to the set of all messages pertaining to such a unique ID as an interaction, the unique ID as the interaction ID.
Given the availability of unique interaction IDs, it is easy to extract complete interactions from a noisy message log where multiple interactions might be interleaved. Our next task is to extract the goal (e.g., the service request or product order) that is the object of the conversation. Goals are often represented in natural language using verb phrases. The information extraction techniques used for extracting verb phrases admit considerable complexity. For the purposes of our proof-of-concept evaluation, we assume an even simpler textual format, consisting of ⟨verb, noun⟩ pairs (such as buy book, supply product, assess claim etc.). Our technique for extracting these is as follows:

- We extract the set of all ⟨verb, noun⟩ pairs that appear in a given interaction.
- We annotate each element of this set with the number of messages that it appears in.
- We identify the element with the highest frequency and if it passes the threshold \( k_{\text{message}} \), it referred to as the goal designator associated with the dependency.

We use the following procedure to identify dependencies from message logs:

- We partition the set of all interactions extracted from a message log into sets that share the same goal designator.
- We assume that a significance threshold \( k_{\text{interaction}} \) is provided by the user. If a cluster of interactions (with the same goal designator) represents \( k_{\text{interaction}} \% \) or higher of the set of all interactions, we treat that cluster as significant and indicative of a dependency.

4 The Task-Dependency Correlation Extraction (TDCE) Technique

In this section, we will present the Task-Dependency Correlation Extraction (TDCE) technique that identifies the task in the depender actor and the task in the dependee actor that are associated with a given dependency. This information will be mined from the process logs.

The mining of task dependency correlations starts with process logs from different actors where the execution of each actor generates a distinct log. A process log consists of a list of tasks executed by an actor over time. Multiple process logs from different actors could be combined into one process log, as shown in the example in Table 1. This process log lists all tasks executed by all actors (either as the depender or as the dependee). By examining this log, we can observe that when actor \( i \) activates task \( a \) at time \( t_x \), within some \( n \) units of time in the future, at time \( t_x + n \), actor \( j \) activates task \( b \). When this particular pattern of task activation become frequent (or satisfies a certain threshold), then there is an indication of a dependency between task \( a \) in actor \( i \) as the depender and task \( b \) in actor \( j \) as the dependee.

Each entry in the process log consists of:

- a taskID, which is used to identify certain task in an actor.
- a timestamp which describes the time when a task is activated by the actor. The timestamp of the first entry is \( t_0 \) which describes the initial time - the timestamps of subsequent entries is increased each by one unit time.

The list of these entries will comprise a process log.

In this example, from the first row, we can observe that at the initial time \( t_0 \), there are three different actors, each of which activates a task, i.e. actor \( A \) activates task \( a_0 \), actor \( B \) activates task \( b_1 \), and actor

Figure 1: An \( i^* \) SR model for a meeting scheduler system (adapted from [20])
actor C activates task $c_0$. In the second row, the time is increased by one unit to become $t_0+1$. At time $t_0+1$, actor $A$ does nothing, actor $B$ activates task $b_1$, and actor $C$ activates task $c_1$, and so on.

In the process log shown in the example above, we can generate a good guess of which tasks participate in a dependency by examining the task sequence patterns that occurs in the process log. We adapt the GSP (Generalised Sequential Patterns) algorithm in order to mine this sequence pattern. It generates pairs of tasks sequences whose the first task is from the depender actor and the second task is from the dependee actor. For example, a pair $<a_0,b_2>$ means that there is a pattern between $<a_0>$ and $<b_2>$ which indicates that there is a dependency between those two tasks with task $<a_0>$ as the dependee and task $<b_2>$ as the dependee. Then it will determine how frequent this pattern is in the log by counting the number of occurrence of each pair. This number of occurrence of each pair is called its support and the predetermined threshold is the minimum support.

The GSP (Generalised Sequential Patterns) algorithm was proposed by Srikanth and Agrawal (1996). The algorithm takes as input a process log which consists of set of tasks ordered by time. It finds all sequences of tasks whose support is greater than the minimum support threshold specified by the user. In addition to the minimum support threshold, there are timing constraints that must satisfied, namely the maximum time difference between the earliest and latest task activation and the minimum and maximum gaps between adjacent task. We leverage this algorithm, providing as input a process log ordered by time and obtaining as output all sequential patterns in the log.

The algorithm makes multiple passes over the log. The initial constraint for this part is that we are only interested in results that consist of two tasks, because our aim is to discover dependencies that relates pairs of tasks. Therefore we limit the pass to $k = 2$. The first pass of the algorithm finds all sequences with a single task in it along with their occurrence count (support). The output is 1-task long sequences or $L_1$. On the second pass, the algorithm generates 2-tasks-long candidate sequences $C_2$ with $L_1$ as its seed. This is motivated by the fact that for a sequence to be frequent, all of its subsequences must also be frequent. As the support counts are determined, the sequences with support greater than the determined threshold are included in $L_2$.

There are two main phases that are explained in detail below, in terms of how candidates are generated and how their support are counted.

1. **Join phase.** In this phase, we generates all candidates starting from candidates of length 1. The process is straightforward as all the tasks that were in the process log are placed in this set of candidates $L_1$. To generate candidates of length 2, $L_2$, a task from $L_1$ is joined with another that is also in the $L_1$. If $i$ and $j$ are tasks belonging to $L_1$, then task $j$ is added to $i$. But for all candidates of length 2, there is one more constraint i.e. any two tasks in a dependency must not happen at the same time. Therefore any candidate dependency relating two tasks that activate at the same time must be excluded from the result. Initially task $j$ should be added as a task-set ($<i,j>$) and as a separate task ($<i><j>$), but because of this constraint, we only add $j$ as a separate task.

In our example from Table 1, we start with all candidates of length 1, $L_1$. It would consists of $<a_0>$, $<a_1>$, $<a_2>$, $<b_1>$, $<b_2>$, $<b_3>$, $<c_1>$, $<c_2>$, and $<c_3>$.

Next we need to eliminate these candidates according to the value of minimum support in the prune phase.

2. **Prune phase.** We eliminate candidates according to our constraints:

(a) Since dependencies relate pairs of tasks from two different actors, any pattern that contains tasks from the same actor must be excluded from the result.

(b) All the candidates with a support value lower than minimum support is excluded from the result.

Note that constraint (a) is only applied to 2-task-long candidates and is not applied to 1-task-long candidates. On the other hand, constraint (b) is applied in both cases.

Returning to our example, we have generated $L_1$ and now all tasks in the $L_1$ must be examined against constraint (b). For this example, we set the minimum support as 2, which means that for any task or set of tasks to be classified as frequent, it must occur at least two times. In the log in Table 1, for actor $A$ there are three different tasks ($a_0$, $a_1$, and $a_2$). The support for these tasks are 3, 1, and 1 respectively. Because the support for $a_2$ is less than the minimum support, they do not included in the sequence of 1-tasks. We repeat this for all tasks and the result is shown in the first column of Table 2.

We continue to search for all candidates of length 2 by repeating the join and prune phases. This step is illustrated in Table 2. In the join phase, we start with the first candidate $<a_0>$ in the first column. Thus, all sequences of form $<a_0,X>$, where $X$ is any task, are searched. Recall that we do not search for $<a_0,X>$ because it implies that the two tasks occur at the same time. By combining $<a_0>$ with the second candidate $<b_1>$, we find 2-task-candidate $<a_0,b_1>$. A similar procedure is repeated for all sequences of the first column. All 2-task-long candidate sequences generated in the join phase are shown in the second column.

Next, according to constraint (a) in the pruning phase, we begin by determining whether the two events are executed by the same actor (if they do, then the sequence is eliminated). For example, in the second row, in sequence $<b_1,b_2>$, both tasks are executed by the same actor, namely actor $B$. Therefore the sequence is eliminated by applying constraint (a). Thus the remaining candidates for the second row are
\(<(b_1)(a_0)\) and \(<(b_1)(c_2)\). The same also applies to sequence \(<(b_j)(b_j)\) in the third row. We then determine the support count for each of the remaining candidates. The first candidate \(<(a_0)(b_1)\) has support count 1, which is less than the minimum support - it is therefore eliminated. Next candidate, \(<(a_0)(b_1)\), also has support count of 1, and is also eliminated. Candidate \(<(a_0)(c_2)\) has support count of 2 - it is therefore included in the result. In the second row, we have two remaining candidates, \(<(b_1)(a_0)\), and \(<(b_1)(c_2)\). Both have a support count of 2 and are thus included in the result. We repeat this procedure for the rest of the candidates. The result is shown in the third column of Table 2.

The result patterns of this algorithm are all sequential patterns that occur between two tasks in the process log. For our process log example in Table 1, these are \(<(a_0)(c_2)\), \(<(b_1)(a_0)\), and \(<(b_1)(c_2)\).

### 5 Evaluation

The purpose of the evaluation exercise is to establish the following:

- The Dependency Extraction (DE) technique and the Task-Dependency Correlation Extraction (TDCE) technique generate reasonably reliable results.
- Both these techniques can be leveraged to improve the quality of i* models (assessed in terms of how closely a model corresponds to the "ideal" model, and hence to the reality being modeled) by leveraging user tagging (or filtering) of the logs recording the behaviour of an "as-is" system or process that the target system is intended to replace. Since i* is also particularly effective as a domain modeling tool, an improvement in model quality might also entail obtaining a better representation of the context in which the target system is to be situated (or even more generally, a better model of the organizational context).

Our evaluation involved the generation of simulated behaviour histories (message logs plus process logs) given an i* model. To achieve this, we randomly executed these models, the sense described below. For each distinct dependency in an i* model, we generated a large number of interactions (specific numbers in the following subsections), with configurable levels of noise (thus we had noisy messages within interactions, and we had entirely noisy interactions that would not point to any reasonable goal dependency). The non-noise components involves messages and interactions that were deliberately constructed to conform to the i* model at hand. The sum total of these interactions provided the message log that we mined. We similarly generated process logs by randomly selecting tasks/goals from the i* model and allocating random timestamps to them. We, however, ensured that each dependency in the model was reflected at least once in the process log. In other words, if there was a dependency relating task \(a_i\) in actor \(A\) to task \(b_j\) in actor \(B\) in the model, we would ensure that the process log contained at least one entry for task \(b_j\) at a time point after that for task \(a_i\).

### 5.1 Evaluation of the DE technique

We started with the i* model shown in Figure 1, originally used in [20]. The model consists of 3 actors and 6 dependencies.

To simplify the process of obtaining goal designators, we assume that they consist of \((\text{verb}, \text{noun})\) pairs. We permit goal designators to also consist of \((\text{verb}, \text{verb})\) since some verbs in the past tense resemble nouns (participle adjectives or normalizing an adjective). When processing the payload, we extracted the pair by getting the first \((\text{verb})\) in the payload and the first \((\text{noun})\) following said verb.

We used the Stanford Log-linear Part-Of-Speech Tagger v3.2.0 [8] for tagging message payloads. The tagger takes a sentence such as This is a sample sentence and assign parts of speech, e.g., noun ver, adjective, etc, like so This DT is V BZ a@1 T sample N N sentence N N. These tags conform to the Penn Treebank Tagset [14] where tag starting with V is verb and tag N is noun. So we can parse our payload with these tags to find the \((\text{verb}, \text{noun})\) pattern.

We performed experiments with 4 parameters. \(n_{\text{message}}\) describes the proportion of noise messages in the complete message log (all non-noise messages permitted the extraction of the correct goal designator). \(n_{\text{interaction}}\) describes the proportion of noisy interaction in the set of all interactions in the message log, where a noisy interaction is one which does not lead to any single identifiable goal designator. \(k_{\text{message}}\) and \(k_{\text{interaction}}\) are as defined in Section 4. We initially created a message log with no noise \((i.e., \text{neither noisy messages nor noisy interactions})\) consisting of 7381 interactions. Setting \(k_{\text{interaction}} = 10\%\) and \(k_{\text{message}} = 10\%\), we were able to extract all of the dependencies, as shown in Table 3.

We plotted the performance of the DE technique (in terms of recall - there were no false positives and hence precision was always 1) against each of these parameters. We show 3 of the 4 results below (the final graph was omitted due to space considerations). As expected, recall decreases as the amount of noise increases \((i.e., n_{\text{message}}\) and \(n_{\text{interaction}}\) increase). Similarly, recall decreases as we get more selective in identifying dependencies \((i.e., k_{\text{interaction}} increases)\). These results were generated using message logs that were between 25000 to 30000 messages long, with about 2000 interactions. The results were consistently the same.

### 5.2 Evaluation of the TDCE technique

Given a set of tasks as the input of our tool, we use two sets: the set of expected dependencies which were in the input model and the set of dependencies actually discovered in the result. Recall is defined by the number of correct dependencies discovered by our
<table>
<thead>
<tr>
<th>Depender</th>
<th>Dependee</th>
<th>Goal Designator</th>
<th>Interaction Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting Scheduler</td>
<td>Meeting ParticipantA</td>
<td>Enter AvailDates</td>
<td>15.13%</td>
</tr>
<tr>
<td>Meeting Scheduler</td>
<td>Meeting ParticipantA</td>
<td>Agreement</td>
<td>18.02%</td>
</tr>
<tr>
<td>Meeting Initiator</td>
<td>Meeting ParticipantA</td>
<td>Attends Meeting</td>
<td>17.18%</td>
</tr>
<tr>
<td>Meeting Scheduler</td>
<td>Meeting ParticipantA</td>
<td>Propose Date</td>
<td>15.76%</td>
</tr>
<tr>
<td>Meeting Initiator</td>
<td>Meeting Scheduler</td>
<td>Meeting BeScheduled</td>
<td>17.55%</td>
</tr>
<tr>
<td>Meeting Initiator</td>
<td>Meeting Scheduler</td>
<td>Enter DateRange</td>
<td>16.37%</td>
</tr>
</tbody>
</table>

Table 3: Results using non-noisy logs with $k_{interaction} = 10\%$ and $k_{message} = 10\%$

The minimum support. We performed separate experiments for each of those variable by varying the value of one variable and keeping the other fixed.

From the result, the recall is 1.0 indicating that every dependency in the expected result set is discovered. This can be explained by the fact that if there is a dependency between two tasks, there are frequent patterns between those two tasks in the log and it will be detected.

On the other hand, in addition to the expected dependencies, there are other dependencies that were discovered in the actual result but were not in the input model. This might happen because one factor that might affects the result is the interleaved of the entries in the task activation log. For example lets assume that we have a log consisting of two actors (let say actor $A$ and actor $B$) with one dependency between these two actors (between task $a_0$ of actor $A$ and task $b_0$ of actor $B$). Since the algorithm will find all the sequence in the log, there can be two different dependency discovered depends on the order of the entry in the log. This is illustrated below.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Time</th>
<th>act</th>
<th>actor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_0$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 1$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 2$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 3$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
</tbody>
</table>

Table 4: Example 1

<table>
<thead>
<tr>
<th>Actor</th>
<th>Time</th>
<th>act</th>
<th>actor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_0$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 1$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 2$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>$t_0 + 3$</td>
<td>$a_0$</td>
<td>$b_0$</td>
</tr>
</tbody>
</table>

Table 5: Example 2

In example 1, the result would be two dependencies $<(a_0,b_0)>$ and $<(b_0,a_0)>$, while the second example the dependency would be $<(a_0,b_0)>$. Therefore there are possibilities that any dependency might show up in the log although it was not in the model.

The minimum support count for our evaluation in the first scenario is fixed at 1.0 which means that support count = 1.0 * the number of execution. For example if we execute 10 times, then the support count is 10, so that any pattern that occurs 10 times or more is included in the result. For the second scenario, we use a task activation log with 2000 entries. The two graphs in Figure 3 show the precision of our technique in two different scenarios. As can be seen in Figure
increase the minimum support count. Two things, either increase the volume of the data or, if we want to get more accurate result, we can do very influential to the accuracy of the result. There-

Figure 3: Precision relative to min support

3(a), with small log, the precision is not very accurate (given just ten entries, precision is only 0.39). But with a larger log, the precision is increasing up to 0.82. Precision remains relatively constant at 0.82 when the log reach 1500 entries or more. While in Figure 3(b), the higher minimum support will give more accurate result. For example, given the support of 1.0, precision reach up to 0.82 but precision drop to 0.5 when support is set to 0.9 and keep decreasing until under 0.1 with support of 0.7 or less. Hence the result suggest that both the number of the entries in the task activation log and the minimum support are very influential to the accuracy of the result. Therefore if we want to get more accurate result, we can do two things, either increase the volume of the data or increase the minimum support count.

Since in this evaluation we artificially created the task activation log and deliberately executed all dependencies in the input model, we acknowledge that they would not be representative for all possibilities of task activation log in practice. For instance, there might be a case where a dependency in the input model was not executed at all or was executed but in a number of times which was lower than the minimal support. In such cases, our technique might not be able to detect it.

Another limitation of this technique involves settings where multiple dependencies exist between the same pair of actors. Our current approach works well if we have a guarantee that only one dependency would exist between a given pair of actors. Thus, when we determine that a pair of tasks are related via a dependency, we are able to leverage the DE technique to identify what the goal designator for that dependency is. In the case of multiple dependencies between the same two actors, the DE technique would suggest multiple goal designators, while the TDCE technique would suggest multiple task pairs, but we would not have the wherewithal to associate the task pairs with the goal dependencies identified by the DE technique.

5.3 Improving requirements quality: Evaluation

A key contribution of this work is the ability to achieve data-driven improvements in the quality of requirements models. There are two approaches to this that we explore. In the first, we explore settings where the user is able to describe the ideal behaviour (for our purposes, a behaviour will be described via a combination of a message log and a process log) of the system in question. We extract models from these idealized behaviours, as opposed to the noisy behaviours that have been the focus of the previous parts of the evaluation. In the second approach, we explore settings where the requirements engineering exercise is conducted in the context of an existing “as-is” system or process(es), the behaviour of which we are able to log. The user filters this (potentially imperfect) behaviour generated by the existing system/process (by removing entries from the message and process logs that (in the perception of the user) represent manifestations of imperfect behaviour, and our machinery extracts models from these filtered logs. The evaluation involved a trained i* modeler, who was asked to generate a model (Figure 4) which was not revealed to the research team. This model played the role of the “ideal” model against which the quality of the extracted models was evaluated. The quality of an extracted model was evaluated by either: (1) assessing how closely it conformed to the user’s “ideal” model (which was revealed to the research team after the model extraction phase was completed) or (2) obtaining input from the user suggesting that some dependencies that existed in the user’s intuitive understanding of the domain (and had been manifested in the idealized behaviours supplied by the user, but not in the “ideal” model) has been discovered by our machinery.

User-generated idealized behaviours: With the model in Figure 4 in mind, the i* modeler gave us a message log with 12 entries (each of which was a request message) and a process log with 35 entries. Our machinery then extracted a partial i* model with the following characteristics. Of the 23 dependencies in the original user model, we discovered 19 dependencies (relating the correct pairs of actors and tasks). We also extracted 9 new dependencies that were not part of the user’s original model. Figure 5 shows the model that was extracted. The bold lines denote false positives and the dashed lines denote false negatives.

User-filtered “as-is” behaviours: In this part of the evaluation, the i* modeler revealed the idealized model to the research team. This was used to generate 5 behaviours (i.e., 5 sets of (message log, process log) pairs). The message logs varied in length from 2 to 10 messages per log. The process logs varied in length from 9 to 13 entries. We additionally generated 5 incorrect behaviours, by randomly selecting 1 dependency in each case and randomly changing either the depender or dependee actor, or the source or target task. We then interleaved the 5 correct and 5 incorrect behaviours and presented these to the i* modeler. Our intent was to simulate the execution of an imperfect system/process, whose behaviour would be represented by the interleaved logs. The i* modeler was then asked to remove from the message and process logs entries that did not correspond to the intuitions that were represented in the idealized model. We then applied our machinery to extract a partial i* model from the filtered logs. We discovered 19 of
the original 23 dependencies in the idealized model (this was an identical result to the evaluation using user-generated idealized behaviours) and 10 new dependencies. Of these 10 new dependencies, 6 were distinct to the dependencies extracted in the evaluation using user-generated idealized behaviours. The * modeler also suggested that 9 of the new dependencies discovered were largely in accord with his intuitions about the domain being modeled, but had not been reflected in the idealized model that he had initially generated. This suggests that this approach can help surface implicit requirements via the filtering of noisy behaviours. The extracted model is shown in Figure 6 below. As before, the bold lines denote false positives and the dashed lines denote false negatives.

As discussed in the previous section, the existence of multiple dependencies between the same pair of actors in this model prevented us from correlating the task pairs generated by the TDCE technique with the dependencies generated by the DE technique. The net upshot was that we had several “unnamed” dependencies. Nonetheless, discovering the existence of dependencies, even in the absence of goal designators, provides valuable insights.

Overall, this part of the evaluation suggests that there is merit in the general idea of using this machinery to improve the quality of requirements extracted, although the machinery missed some dependencies and generated some false positives.

6 Related Work

The research reported in this paper is related in some ways to the existing body of work on process mining [18] [19], in that we also use process logs as one of several data sources. However, what we do with process logs is entirely different. Unlike process mining, we generate task-dependency correlations from this data. A proposal to leverage web logs to support non-functional requirements elicitation [16] shares some intuitions with our work (but uses different techniques as well as different inputs and outputs). A body of existing work on extracting requirements from natural language can inform the extraction of goal designators (see, for instance, [2] [3]), although our current evaluation uses a simpler proof-of-concept implementation. The Business Intelligence Model (BIM) [11] bears some relation to our proposal in its ability to serve as a data-driven dashboard for the enterprise. Approaches to run-time adaptation, such as in [6], concept discovery [12], ontology extraction [9] and model-based diagnosis [7] are relevant. The literature on requirements change (e.g., [10]) is also relevant. We expect that the literature on social network analysis [13] might provide techniques of relevance to this
7 Conclusions and future work

In this paper, we have provided preliminary evidence to support the hypotheses that data-driven extraction of requirements models can be effective, and that this approach, coupled with user involvement in identifying undesirable behaviour traces, can lead to more accurate models. We have performed this evaluation in the context of \textit{i*} models, and have further simplified the problem by focusing not on extracting complete \textit{i*} models, but only the dependencies and the task dependency correlations. We believe that this is a first step in a much larger program of research that considers the problem of data-driven extraction of models in a range of notations. We have also offered some innovative approaches to the evaluation of these techniques, but much deeper and careful evaluation remains to be done. Future work will also involve leveraging results from a range of other areas, including social network analysis.

References


Predicting Requirements Changes by Focusing on the Social Relations

Takako Nakatani¹ Toshihiko Tsumaki²

¹ Graduate School of Business Sciences, University of Tsukuba, 3-29-1, Ohtsuka, Bunkyo-ku, Tokyo, 112-0012, Japan
Email: nakatani@gsm.otsuma.tsukuba.ac.jp

² Information Systems Architecture Research, National Institute of Informatics 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
Email: tatsumaki@nii.ac.jp

Abstract

Software requirements are changed by various factors. Stakeholders that are analysed in traditional requirements engineering are mainly requesters or decision makers with regard to the requirements specifications. Such stakeholders are selected as intentional actors in the i* framework. The novel point of this paper is that we focus on the world of parties who are the environmental factors of the requirements of the intentional actors. Our purpose is to propose a method to predict requirements changes by focusing on the social relations. In this paper, we present our method and evaluate its effectiveness through a simulation of requirements changes.

Keywords: requirements engineering, requirements changes, requirements volatility, stakeholder analysis, social relations

1 Introduction

Requirements volatility produces various negative effects on the software development process: reduction of performance (Zowghi & Nurmuliani 2002) and increase of costs (Nurmuliani et al. 2004) to the project. How to cope with requirements volatility is one of significant themes in requirements engineering. If we can predict requirements volatility, our design techniques work well to prevent the deterioration of software. Design patterns (Gamma et al. 1995) is one of the solutions.

Basically, requirements are volatile. According to the observation of requirements elicitation (Nakatani et al. 2008), not a small number of requirements are elicited in the middle or late stages of projects. There have been researches done on requirements volatility. Ebert and Man focused on the problems that cause requirements volatility (Ebert & D. 2005). Also, the risks involved in requirements volatility have been discussed by Williams et al (Williams et al. 2006). According to our closed discussion with practitioners, the environmental factors of stakeholders have not been analysed well. The motivation of our study is to develop a method to analyse the environments surrounding stakeholders who are involved in the requirements analysis. What kinds of environmental factors affect these stakeholders?

The purpose of this paper is to predict requirements changes through the analysisation of the environmental factors that may or may not have an impact on the requirements. There are multiple parties in the environment. For example, executives, competitors, cooperative organisations, the natural environment, etc. We focus on these parties and the “social relations” between parties. Changes in social relations sometimes force the stakeholders to change their requirements. If a requirements analyst only analyses the requirements of stakeholders, the changes in social relations are set a side, and thus, out of the analyst’s scope. As a result, the analyst cannot realise requirements changes that may be caused from the social relations. We know that the analysts and stakeholders may not be able to control or manage such changes, but we have confidence that we can predict the changes that cause volatility within the requirements. In this paper, we present the effectiveness from the perspective of social relations in order to predict requirements volatility. It will help us clarify one of the aspects of requirements changes.

The structure of this paper is as follows: Section 2 presents the perspective of social relations to predict requirements changes and define the social relations; In Section 3, we describe a method to analyse these relations. After describing the method, we apply the method to an example in order to clarify the effectiveness of the method to predict requirements changes caused by the social relations in Section 4. More precisely, we will also show a conceptual model of the example domain, and designate the roles of conceptual analysis and social relations. Before we conclude the paper in the final section, we discuss the related work in Section 5.

2 Social relations

In this section, we will outline the strategic dependency model of i* (Yu 1997). Then, we introduce social relations that affect requirements volatility.

2.1 i* framework

The i* framework consists of two models: the strategic dependency model (SD model) and the strategic rationale model (SR model). In this paper, we focus on the SD model, and moreover, we extend the model with social relations.

The SD model contains dependency relations between intentional actors within the analysing world. The intentional actors are the stakeholders of the developing system. There are four dependency relations
between actors: task, resource, goal, and soft goal. These dependency relation types are represented by a hexagon, rectangle, oval, and cloud, respectively in the SD model. With each relation, the symbol “D” is used, which represents the direction from a dependee to a depender. Figure 1 is an example of the SD model. It covers a part of the stakeholders of a train service.

A requirements analyst can apply i* in order to extract the goals of each actor, then he/she analyses the goal-oriented analysis with the SR model. The i* framework is helpful to analyse the “why” aspect of requirements. The actors in the scope of i* have intention in deciding the requirements of the developing system. Hence, the environmental factors that affect the intention of the actors were set outside the scope of i*.

The environmental factors cause requirements changes. Though there are various factors, we regard the principles of these factors to be but a few. We have challenged ourselves to clarify the basic concepts of the environmental factors, and further, set them inside the scope of analysis in order to predict requirements changes.

2.2 Environmental factors
First of all, we divide the environmental factors that cause requirements volatility into internal and external factors of the software development project. Internal factors relate to the maturation of the project, and should be managed within the project. Examples of internal factors are, knowledge of engineers to understand the requirements correctly, technical issues on requirements analysis and design, and maturity of management. Conversely, external factors are uncontrollable for the project. These external factors include economic liberalism, changes in and of markets, changes in personnel, policy changes of organisations, and expansion in and of users’ variation. The external factors are the targets of this paper.

In order to analyse external factors, we focus on their two properties: variability and fluidity.

- Variability
  As much as the domain becomes more complex, and the number of concepts increases, we can apply a generalisation-specialisation structure to the domain, and thus, resolve its complexity. When we introduce a more abstract concept from the similar concepts, then we are able to predict the addition of other similar concepts. The variability of the concept can be modelled with a conceptual analysis.

- Fluidity
  There is a kind of change that propagates from one party to another and further, causes other changes. We refer to the change as fluidity. The fluidity may induce crucial requirements changes.

We can, for example, visualise and analyse the variability of environmental factors with class diagrams of UML. In order to analyse the propagation of the fluidity, we extend the scope of the SD model with four types of relations between parties. In the next subsection, we define these four types of the relations.

2.3 Social relations
The fluidity arises within a relation between parties, i.e. the addition of a new party, the transformation of relations, and the deletion of relations. The volatility of the requirements volatility can be investigated through the fluidity of relations. We refer to a relation between parties as a social relation. Fiske classified human relations into four elementary forms: communal sharing, authority ranking, equality matching, and market pricing (Fiske 1992). We adopt them to four social relations: sharing relation, ranking relation, exchanging relation, and contracting relation. Then, the SD model is enhanced in order to analyse the environment of intentional actors by the four social relations. The media connected to the relations are, for example, gifts, offerings, sharing properties, strategies, rules, constraints, force, rights, etc. The detailed definitions of social relations are as follows.

- Sharing relation
  Sharing relations are relations between parties who share the common interests or cultures and feel that good things for one are good things for another. They are sometimes competitors or rivals. Characteristics of sharing relations are as follows.

  - Parties are connected by a sharing medium that brings common profit and property.
  - The action of parties is performed based on a common aim.
  - The parties sometimes try to obtain more shares than other parties. In order to solve such a competitive situation, the relation is transformed into another type.

- Ranking relation
  The upper ranking party has privileges over the lower ranking party. Ranking is introduced into societies such as militaries and corporations with social responsibilities. Characteristics of ranking relations are as follows:

  - The upper rankings receive payments from the lower rankings as a privilege of their authority, and in return, the lower rankings receive rewards from the upper rankings.
  - The lower party behaves according to the intentions of the higher party.
  - The ranking relation exists based on a power balance. If the balance is broken, the relation is transformed into another type.

- Exchanging relation
  This type of relation guarantees interdependence and fair exchange. Characteristics of exchanging relations are as follows.

  - The parties sometimes try to obtain more shares than other parties. In order to solve such a competitive situation, the relation is transformed into another type.
The same valued medium is exchanged between the members based on mutual agreement.
If the medium is changed, the agreement may be broken, and then, the relation is transformed into another type.

- Contracting relation
  Social contracts include tradition, rules, promises, etc. A party who breaks the contracts receives social punishments. The contracting relation is a basis of our modern social life. Characteristics of contracting relations are as follows.
  - The value of medium is defined by authorised parties.
  - The parties behave according to the social system.
  - If the contents of the contract is changed, the relationship may be transformed.

3 Overview of the method

The method assumes that the SD model has been presented. The main part of our method is to provide a process of extracting four types of relations between parties. The relations are extracted from an open space. The following process is repeated until the effect on the intentional actors by a newly extracted relation is considered enough small to be ignored within the world of the SD model.

1. Focus on each intentional actor in the SD model.
2. Apply four types of social relations to each actor.
3. Define a medium on each relation that affects the intention of the actor. If no medium is found on the relation, delete the relation.
4. Define a party that is connected to the relation. Those parties may already be defined within the world of analysis, and add the relation to the model.
5. Analyse the probability of transformation of each medium based on the knowledge and/or social experiences of the analyst.
6. Change the medium
   There are two types of changes of a medium. The first type causes a transformation of the relation. This type of transformation changes the situation of the environment of the intentional actors, thus, the fluidity may change the intention of actors. The second type is a change of the medium without changing the type of the relation. This type can be modelled as the variability of a concept within a conceptual model.

The process is shown in Figure 2. In the next section, we show an example and simulate the process. According the simulation, we evaluate the effectiveness of the method.

4 Evaluation by example

4.1 Overview

We performed an empirical study on an application software development. The example was a rail transportation service support system of a passenger traffic company. The system was composed of functions to support safe and comfortable train services: train scheduling and rescheduling, traffic control, passenger reservation, ticket sales, users' claims to management, seat reservation, and so on. The main train route of TrainCompany had monopolised a certain district. TrainCompany told us that the purpose of the system is to support a part of the operations of the company.

The current requirements of the system are as follows:

- Req.1: Customers can reserve their desired seats through several different channels.
- Req.2: Customers can cancel the reservation up to a specified time before the departure.
- Req.3: Customers can know the information of trains that they can take as well as the fare according to the recommended travel plan.
- Req.4: Customers can select the payment method: cash, credit card, or a prepaid card produced by the company.

We predicted the changes of requirements with the method. The results were evaluated by comparing them with the work of actual requirements changes.

4.2 Models in the application

We built two models in order to predict requirements changes. One is a conceptual model for analysing the variability of concepts, while the other is an extended SD model meant to analyse the fluidity of social relations.

4.2.1 Conceptual model

The conceptual model of the train service is shown in Figure 4. The coloured classes represent concepts outside the scope of the current system, but represent possible variability of the concepts inside the scope. The variability is the source of requirements changes. For example, we know that the airplane service provides various classes of seats. Thus, new train services that provide such various kinds of reserved seats can be predicted. However, we need to analyse the fluidity of the environment with the extended SD model.

Simply, we can estimate the possibility of the modification of services through the conceptual model shown in Figure 4. HAZOP (IEC 2001) is a method to derive unexpected phenomena by using guide words, such as, over, less, slow, high, low, stop, clogged, intermittent, etc. For example, if some volume of oil goes in a pipe, such guide words as, over the volume, less volume, at a slower speed, at a higher speed, etc.,
helps us define unexpected scenarios. We refer to the
guide words of HAZOP and, define the following pos-
sible requirements changes. The attribute values and
services can be derived from a specialisation structure
within the conceptual model.

Table 1: The possible change in ticket services.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Current value</th>
<th>Possibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>from/to transitPoint</td>
<td>fixed</td>
<td>flexible</td>
</tr>
<tr>
<td>fare</td>
<td>fixed</td>
<td>flexible</td>
</tr>
</tbody>
</table>
| means of transpor-
tation             | several kinds | flexible, emergency  |
| travel info.       | fixed         | flexible             |
| Service            | none available| available            |
| Route variation    | none available| available            |
| Cooperation        | none          | available, air and bus|

4.2.2 Extended SD model

The SD model was already shown in Figure 1. We
extracted environmental factors with relations by fo-
cusing on Train within the SD model. After the third
iteration of the analysis, we got the model shown in
Figure 3. In the diagram, the scope of i* is shown as
the greyed area.

In Figure 3, we show the legend for these four so-
cial relations. Basically, the arrows connect parties
through objects. Only the ranking relation has a di-
rection from a depender to a dependee. Other rela-
tions: sharing, exchanging, and contracting relation,
are represented by bi-directional arrows. Especially,
the double arrows of exchanging relation represent
give and take processes. The extracted social rela-
tions are as follows.

- Sharing relations
  - CompetitiveTrains
  - CompetitiveTrains and Train share the
  - TouristResort
  - TouristResort and Train share TouristAt-
    tractions.

- Ranking relations
  - They are business operation, company operation,
    and service satisfaction. TrainCompany is a fac-
    tor that is connected with Train via a ranking
    relation named business operation.

- Exchanging relations
  - The following factors are connected with Train
    via exchanging relation. A_Bus and B_Bus have
    exchanging relations with Train via TrafficChannel.

- Contracting relations
  - There is a contracting relation between Train-
    Company and BusCompany_B.

4.3 Predicted requirements changes

Transformation of a social relation causes changes in
the behaviour or roles of parties, and finally, they
impact the intention of actors. We introduced four
social relations and their characteristics. It is possi-
ble for those social relations to transform into other
relations. By analysing such fragility, we can predict
requirements changes with the conceptual model, and
the extended SD model.

The following requirements on the services of Train
are defined according to our analysis.

- ReqChange 1: Add new functions to operate new
  kinds of seats and establish various new fares.
- ReqChange 2: Introduce various fare options.
- ReqChange 3: Add new functions to manage new
  stations and operate flexible routes.
• **ReqChange 4:** Add new functions to handle the fares for the customers who transfer trains to/from a competitor's means of transportation.

We traced these requirements within the new system. **ReqChange 1** and **ReqChange 2** were derived from the conceptual model. **ReqChange 3** and **ReqChange 4** were predicted by analysing the extended SD model. **ReqChange 3** was found in the transformation of the sharing relation (labeled “TouristAttractions”) to an exchanging relation (labeled “Station and/or route”). **ReqChange 4** was found in transformation of the sharing relation (labeled “Market”) to a contracting relation (labeled “Inter-connection”).

Here, we could predict the possibility of requirements changes with their rationales.

### 4.4 Discussion

In the real world, a competitor entered the area and started a faster train service. Then, TrainCompany decided to change the route to another route, of which, the time schedule was rather more sparse than the previous route. TrainCompany could out compete its competitor with regard to the speed. As a result, **ReqChange 3** and **ReqChange 4** were added. If we did not analyse the social relations, we could not realise the possibility of these requirements changes.

One of the changed requirements, which we could not discover, was a problem related to the location of reserved seats. Some of the stations on the new route were too short for the Trains, and so the protruding doors of the Trains must remain closed when the train stopped at the stations. Thus, before reserving seats for passengers, the system evaluates the length of all stations that the train stops at and, finds the available seats in the cars without the protruded doors. However, this additional requirement could not be predicted through our approach. The requirement depended on a business rule that states, TrainCompany has to provide a highly comfortable travel for passengers with seat reservations, and further, address the physical problem of the length of the stations. Our model does not have the ability to take the business rules into account. Our future work is to refine and evaluate the method by applying another example.

### 5 Related work

There are researches with regard to causes of requirements volatility. In our previous work, we introduced the concept of a speed of “requirements maturation” instead of requirements volatility (Nakatani et al. 2011). If requirements are volatile, their speed to mature is slow. The requirements maturation is calculated according to the stability of requirements and accessibility of stakeholders by a requirements analyst. Nurmuliani et al. categorised the causes of requirements volatility (Nurmuliani et al. 2004): they are evolution of customer and market needs, growth of users and developers and change of organisational policy. Bano et al. classified factors that cause requirements changes into three types: business, organisation and project, and estimated their impacts (Bano et al. 2012). However, they did not provide a method to analyse the environment factors.

According to the books of the late 20th century (MacAulay 1996, Sommerville & Sawyer 1997, Kotonya & Sommerville 1998), authors pointed out that the environmental factors cause requirements changes and focused on techniques to elicit requirements. Therefore, the environmental factors were not analysed in their scope. Alexander and Robertson developed the onion model to understand the project society (Alexander & Robertson 2004, Robertson & Robertson 2005). In the onion model, each stakeholder is categorised into zones. The model represents the stakeholders and their involvement with product development. In their scope, stakeholders are mainly requesters and/or decision makers with regard to the requirements specifications. Such stakeholders are selected and analysed by analysts who are positioned inside the system boundary. Therefore, their scope was still inside the system boundary. Pohl (Pohl 2010) proposed the term “context” and introduced a system context that contains all aspects that need to be considered during system development. The world of parties that we introduced in this paper fits well to the Pohl’s system context.

Goal-orientated analysis methods are developed to find the alternative requirements in the goal graph (Yu 1997, Chung et al. 1999, van Lamsweerde 2001). When a goal has subgoals with OR-relation, these subgoals are alternative goals to achieve the super goal. If one of the alternative goals is selected, it may be alternated with another goal. We applied the
conceptual model to derive the variability of concepts that imply alternative requirements. According to the goal-oriented analysis methods, goals are defined as properties of stakeholders. The scope of these methods is closed inside the world of stakeholders. Our extended SD model opens the scope of analysts to the world of parties whose changes have an impact on the intention of stakeholders. The dependencies between parties are not based on the personal intention in the original framework, but on the structure of organisations.

A power/interest grid, power/influence grid, influence/impact grid, and salience model are techniques introduced in the project management body of knowledge (PMBOK) (Project Management Institute 2013). Stakeholders are grouped according to their power, influence, interest, impact urgency, or legitimacy with regard to the requirements and, each group is placed in a two dimensional space. These techniques are useful in managing stakeholders, but the viewpoint of PMBOK is not concerned with requirements, but the management of stakeholders. We introduced two models, one is an extended SD model, and the other is a static conceptual model, both of which are aimed at the prediction of the requirements changes.

6 Conclusion

We proposed a method that consists of two tools: one is an extended SD model and the other is a conceptual model with a class diagram in UML. The basic concept of the method is that requirements analysts have to take into account the influence on software requirements by social relations in order to identify the volatility of requirements. The scope of the method is not only the world of intentional actors, but also the world of environmental factors. Both worlds are related utilising four relations: sharing, exchanging, ranking, and contracting relations. These relations are interpretations of Fiske’s psychological theory. The method was evaluated by applying it into a train service system. As a result, we could report that the method can predict requirements changes within the system. In our future work, we will apply the method to practical examples and refine the method.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 25330077.

References


IEC (2001), ‘IEC 61882 hazard and operability studies (HAZOP studies) - application guide’.


SQL-Sampler: A Tool to Visualize and Consolidate Domain Semantics by Perfect SQL Sample Data

Van Tran Bao Le\textsuperscript{1} \hspace{1em} Sebastian Link\textsuperscript{2} \hspace{1em} Flavio Ferrarotti\textsuperscript{1}

\textsuperscript{1} School of Information Management, Victoria University of Wellington, New Zealand
Email: van.t.le@vuw.ac.nz, flavio.ferrarotti@vuw.ac.nz
\textsuperscript{2} Department of Computer Science, University of Auckland, Auckland, New Zealand
Email: s.link@auckland.ac.nz

Abstract

SQL database designs can result from methodologies such as UML or Entity-Relationship modeling, Description Logic specifications, or relational normalization. Independently from the methodology, the use of good sample data is promoted by academia and commercial database design tools to visualize, validate and consolidate the database designs produced. Unfortunately, advice on what constitutes good sample data, or support to create good sample data are hard to come by. Armstrong databases provide a right notion of sample data that perfectly represent the domain semantics encoded in the form of SQL constraints. We present a tool that computes Armstrong sample tables for different classes of SQL constraints, and different interpretations of null markers. Armstrong tables illustrate the perceptions of an SQL database design about the semantics of an application domain. The tool exemplifies the impact of various design choices on Armstrong tables. These include the expressiveness of the classes of SQL constraints considered, and the semantics of null markers. Armstrong tables complement existing database design methodologies. In particular, they provide data samples that guide the transfer from relational approximations of an application domain to an actual real-life SQL table design.

1 Introduction

Classical database design comprises a variety of methodologies, including conceptual approaches with UML or Entity-Relationship modeling, Description Logic specifications or relational normalization. The output of these approaches is usually a database schema within Codd’s relational model of data. The ultimate classical goal, however, is to design an SQL database schema. Relational database schemata constitute only approximations of the target SQL database schema. The reason is that SQL provides features not available in the relational model. In SQL tables it is possible that a duplicate and partial information can occur. This makes data processing more expensive, and the occurrence of null markers provides simple yet efficient means for partial information to enter the database. Due to these features, the interaction of SQL constraints is delicate, difficult to comprehend for database designers, and even more difficult to communicate to other stake-holders of the target database. Since SQL database design is a challenging and essential task, academic and commercial database design tools, e.g., ERWin (CA Technologies 2011), promote the use of good sample data to visualize, validate and consolidate the database designs they produce. Unfortunately, advice on what constitutes good sample data, or support to create good sample data are hard to come by. Armstrong databases provide a right notion of perfect sample data (Beeri et al. 1984, Fagin 1982, Hartmann, Kirchberg & Link 2012, Mannila & Räihä 1986). They constitute single database instances that satisfy the SQL constraints currently perceived semantically meaningful by the team of database designers, and, for a given class of SQL constraints under consideration, violate all those constraints currently perceived meaningless. Hence, Armstrong databases are visualizations of abstract sets of SQL constraints.

2 Motivating Example

We will now examine an example that illustrates how Armstrong databases can be used to transfer a relational approximation of a target database schema into an SQL table definition. The example showcases the benefit in using good sample data to complement current database design methodologies. For this purpose we revisit a classical example, originally used to show that there are Boyce-Codd normal form decompositions that cannot preserve all functional dependencies (FDs) (Beeri & Bernstein 1979). Suppose the design team has obtained the relation schema CONTACT with columns Address, City, and ZIP, and FD set $\Sigma$ with Address $\rightarrow$ ZIP and ZIP $\rightarrow$ City. This schema is in Third normal form, but not in Boyce-Codd normal form (Beeri & Bernstein 1979). Normalization algorithms stop here, and cannot provide any further guidance on how to implement the relation schema within an SQL table definition. An inspection of an Armstrong relation for $\Sigma$ such as

<table>
<thead>
<tr>
<th>Address</th>
<th>City</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 Hudson St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>San Francisco</td>
<td>94107</td>
</tr>
<tr>
<td>15 Maxwell St</td>
<td>San Francisco</td>
<td>94129</td>
</tr>
</tbody>
</table>

armstrong relation

does also not help, as SQL features like duplicate rows and null markers are not featured in relations. We may therefore ask for an Armstrong table (Hartmann,
Kirchberg & Link (2012) for the given set $\Sigma$ of FDs and 
NOT NULL constraints, say on Address and ZIP, e.g.,

$$\text{Armstrong table } t_1$$

<table>
<thead>
<tr>
<th>Address</th>
<th>City</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 Hudson St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>03 Hudson St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>San Francisco</td>
<td>94107</td>
</tr>
<tr>
<td>15 Maxwell St</td>
<td>San Francisco</td>
<td>94129</td>
</tr>
<tr>
<td>46 State St</td>
<td>null</td>
<td>60609</td>
</tr>
</tbody>
</table>

An inspection of the table $t$ shows that a specification of FDs does not exclude occurrences of duplicate rows in SQL tables. In fact, $\Sigma$ does not imply any uniqueness constraints (UCs) over SQL tables (Hartmann, Kirchberg & Link 2012). At this stage, the design team decides that the FD Address, City $\rightarrow$ ZIP should be replaced by the stronger UC $u(\text{Address}, \text{City})$, meaning that there cannot be any different rows with matching total values on both Address and City. Furthermore, the interpretation of the null marker $\text{ni}$ is no information, i.e., a value may not exist, or it may exist but is currently unknown. This is the interpretation that SQL uses (Zaniolo 1984). The occurrence of $\text{ni}$ in the table above indicates that the column City is nullable. An Armstrong table $t'$ for the revised constraint set is

$$\text{Armstrong table } t_2$$

<table>
<thead>
<tr>
<th>Address</th>
<th>City</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 Hudson St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>Manhattan</td>
<td>10001</td>
</tr>
<tr>
<td>70 King St</td>
<td>San Francisco</td>
<td>94107</td>
</tr>
<tr>
<td>35 Lincoln Blvd</td>
<td>San Francisco</td>
<td>94129</td>
</tr>
<tr>
<td>15 Maxwell St</td>
<td>null</td>
<td>60609</td>
</tr>
<tr>
<td>15 Maxwell St</td>
<td>null</td>
<td>60609</td>
</tr>
</tbody>
</table>

Looking at the last two rows of the table $t'$, the design team notices that the UC $u(\text{Address}, \text{ZIP})$ is still not implied by the constraints specified so far. As the UC is considered to be meaningful, the designers decide to specify this constraint as well. Inspections of further sample data does not reveal any additional meaningful constraints. Thus, the design team finally arrives at the following SQL table implementation

```sql
CREATE TABLE Contact (    
    Address VARCHAR,    
    City VARCHAR,    
    ZIP INT,    
    UNIQUE(Address, City),    
    PRIMARY KEY(Address, ZIP),    
    CHECK(Q = 0));
```

where the state assertion $Q$ enforces the FD $\text{ZIP} \rightarrow \text{City}$ by

```sql
SELECT COUNT(*)
FROM Contact c1
WHERE c1.ZIP IN (
    SELECT ZIP
    FROM Contact c2
    WHERE c1.ZIP = c2.ZIP
    AND (c1.City <> c2.City)
    OR (c1.City IS NULL AND c2.City IS NOT NULL)
    OR (c1.City IS NOT NULL AND c2.City IS NULL));
```

on a data or middle tier.

3 Contribution

In this article we showcase a new tool, called SQL-Sampler, to generate Armstrong tables for different classes of SQL constraints including

- NOT NULL constraints,
- uniqueness constraints, and
- functional dependencies,

and the following two interpretations of null marker occurrences:

- $\text{ni}$, that is, no information, and
- $\text{unk}$, that is, value unknown at present.

The tool complements existing database design methodologies by creating sample data with provably good properties. The creation of such sample data is prompted by leading database design tools, but has not enjoyed any support yet. Our tool enables the effective visualization, consolidation and communication of database designs produced by currently available design methodologies. In particular, it can be used to transfer relational approximations of a target schema into an SQL implementation.

Organization. We discuss related systems and the novelty of our tool in Section 4. In Section 5 we give the necessary definitions of the SQL table model and the classes of constraints under investigation. An overview of the functionality of SQL-Sampler is given in Section 6. The use of the graphical user interface is illustrated by a simple example in Section 7. Details on the implementation of SQL-Sampler are given in Section 8. Finally, we conclude in Section 9.

4 Related Systems and Novelty

SQL-Sampler helps design teams identify those SQL constraints that encode the semantics of the given application domain. The help comes in form of Armstrong tables that provide an exact, sample-based representation of the SQL constraints currently perceived meaningful. Design teams can consolidate their current perceptions about the application domain’s semantics by inspecting Armstrong tables together with domain experts.

Armstrong databases are user-friendly representation formats of abstract constraints (Beeri et al. 1984, Fagin 1982, Hartmann, Kirchberg & Link 2012, Mannila & Räihä 1986). Several prototypes were developed that compute Armstrong databases for a given set of FDs, and the paradigm design-by-example was established (De Marchi et al. 2003, Mannila & Räihä 1986, Silva & Melkanoff 1979). However, all these tools produce relations. These are just the idealized special case of SQL tables where neither duplicate rows nor null values occur. Thus, relations never show what delicate interactions between NOT NULL constraints, UCs, and FDs are possible over SQL tables. Hence, previous tools do not help with SQL table design - and were not intended for this task.

SQL-Sampler is designed to create Armstrong databases for different classes of SQL constraints, and for different interpretations of null marker occurrences within SQL tables. The classes considered include NOT NULL constraints, UCs, and FDs. The interpretations of null marker occurrences include the no information (Zaniolo 1984) and the unk interpretation (Codd 1979). The choice of a class and an interpretation determines the semantics of SQL tables,
and thus also the Armstrong tables produced. SQL-Sampler is the result of implementing algorithms for different combinations of these SQL constraints, and different interpretations of null marker occurrences, published in our recent work (Ferrarotti et al., 2011, Hartmann, Kirchberg & Link 2012, Le et al. 2012b). Our tool is the first implementation of these algorithms. The tool has been exploited in comprehensive experiments that confirmed its usefulness in identifying semantically meaningful constraints that were incorrectly perceived as meaningless before its use (Le et al. 2013). This research has extended usability studies from the relational model of data (Langeveeld & Link 2010).

5 SQL tables and constraints

In this section we define the syntax and semantics for the different classes of constraints under different interpretations of null markers.

Let $\delta = \{H_1, H_2, \ldots \}$ be a countably infinite set of symbols, called columns. A table schema is a finite non-empty subset $T$ of $\delta$. Each column $H$ of a table schema $T$ is associated with an infinite domain $dom_H$ of the possible values that can occur in column $H$. To encompass partial information every column may contain occurrences of a null marker, $n_1 \in dom_H$.

For column sets $X$ and $Y$ we may write $XY$ for $X \cup Y$. If $X = \{H_1, \ldots, H_m\}$, then we may write $H_1 \cdots H_m$ for $X$. In particular, we may write $H$ to represent $\{H\}$. A row over $T$ is a function $r : T \rightarrow \bigcup_{H \in T} dom(H)$ with $r(H) \in dom(H)$ for all $H \in T$. For $X \subseteq T$, let $r(X)$ denote the restriction of the row $r$ over $T$. An SQL table $t$ over $T$ is a finite multi-set of rows over $T$. For rows $r_1$ and $r_2$ over $T$, $r_1$ subsumes $r_2$ if for all $H \in T$, $r_1(H) = r_2(H)$ or $r_2(H) = n_1$. For example, the row

$$(03 \text{ Hudson St, Manhattan, 10001})$$

subsumes the row

$$(03 \text{ Hudson St, n1, n1})$$

For a row $r$ over $T$ and a set $X \subseteq T$, $r$ is said to be $X$-total if for all $H \in X$, $r(H) \neq n_1$. Similarly, an SQL table $t$ over $T$ is said to be $X$-total if every row $r$ of $t$ is $X$-total. An SQL table $t$ over $T$ is said to be total if it is $T$-total.

A null-free sub-schema (NFS) over the table schema $T$ is an expression $nfs(T_s)$ where $T_s \subseteq T$. The NFS $nfs(T_s)$ over $T$ is satisfied by an SQL table $t$ over $T$, denoted by $\models_\exists nfs(T_s)$, if and only if $t$ is $T_s$-total. In practice, the NFS consists of those columns declared NOT NULL in the SQL table definition.

An SQL functional dependency (SFD) over a table schema $T$ is an expression $A \rightarrow B$ where $X \subseteq T$. An SQL table $t$ over $T$ satisfies the SFD $A \rightarrow B$ if for all rows $r$, $r' \in t$ the following holds: if $r(A) = r'(A)$ and $r$ and $r'$ are both $X$-total, then $r(B) = r'(B)$ (Lien 1982). An SQL uniqueness constraint (SUC) over table schema $T$ is an expression $u(X)$ where $X \subseteq T$.

An SQL table $t$ satisfies the SUC $u(X)$ if for all rows $r, r' \in t$ the following holds: if $r(A) = r'(A)$ and both $r$ and $r'$ are $X$-total, then $r(B) = r'(B)$. For examples, both SQL tables $t_1$ and $t_2$ from the introduction satisfy ZIP $\rightarrow$ City. While the table $t_1$ violates every SUC, the table $t_2$ satisfies $u($Address, City$)$ but violates $u($Address, ZIP$)$.

Let $C$ be a class of constraints, for example, the combined class of NOT NULL constraints, SUCs and SFDs. We say for a set $\Sigma \cup \{\varphi\}$ of constraints from $C$ over table schema $T$ that $\Sigma$ implies $\varphi$, denoted by $\Sigma \models \varphi$, if for every SQL table $t$ over $T$ that satisfies every constraint in $\Sigma$, $t$ also satisfies $\varphi$.

For example, the table $t_2$ from the introduction shows that the set $\Sigma = \{ZIP \rightarrow City, u(Address, City)$, and the NFS $nfs(Address, ZIP)$ does not imply $u(Address, ZIP)$.

For a set $\Sigma$ of constraints in $C$ over table schema $T$, we say that an SQL table $t$ over $T$ is $C$-Armstrong for $\Sigma$ if $t$ satisfies every constraint in $\Sigma$, and violates every constraint in $C$ over $T$ that is not implied by $\Sigma$.

For example, the table $t_2$ from the introduction is Armstrong for the set $\Sigma$ containing ZIP $\rightarrow$ City, $u(Address, City)$, and $nfs(Address, ZIP)$. By inspecting table $t_2$, we know that $\Sigma$ does not imply City $\rightarrow$ Address or $u(Address, ZIP)$, but does imply $Address, ZIP \rightarrow City$ and $u(Address, City)$.

Constraints can also be defined on tables that feature the Codd null marker $\text{unk}$, instead of $n_1$. In that case we speak of Codd tables. For a Codd table $t$ over $T$, the set $\text{Poss}(t)$ of all possible worlds relative to $t$ is defined by

$$\text{Poss}(t) = \{ t' \mid t'$ is a table over $T$ and there is a bijection $b : t \rightarrow t'$ such that $\forall r \in t$, $r$ is subsumed by $b(r)$ and $b(r)$ is $T$-total $\}$$

A Codd functional dependency (CFD) over table schema $T$ is an expression $\diamond(X \rightarrow Y)$ where $X, Y \subseteq T$. A Codd table $t$ over $T$ satisfies $\diamond(X \rightarrow Y)$ if there is some $p \in \text{Poss}(t)$ such that for all rows $r, r' \in p$ the following holds: if $r(X) = r'(X)$, then $r(Y) = r'(Y)$. A Codd uniqueness constraint (CUC) over table schema $T$ is an expression $\forall u(X)$ where $X \subseteq T$. A Codd table $t$ satisfies $\forall u(X)$ if there is some $p \in \text{Poss}(t)$ such that for all rows $r, r' \in p$ the following holds: if $r(X) = r'(X)$ and both $r$ and $r'$ are $X$-total, then $r = r'$. The notions of implication and Armstrong tables, defined in the context of SQL tables above, are defined analogously in the context of Codd tables. The use case in Section 7 discusses CUCs and CFDs on the same example from the introduction, thereby illustrating the differences between both semantics.

Algorithms to compute $C$-Armstrong tables were recently developed for the classes $C$ of NOT NULL constraints and i) SUCs in (Le et al. 2012b,a), ii) SFDs in (Hartmann, Kirchberg & Link 2012), iii) SUCs and SFDs in (Hartmann, Kirchberg & Link 2012), and iv) CUCs and CFDs in (Ferrarotti et al. 2011). Our tool implements all of these algorithms.
Figure 2: Main Interface of SQL-Sampler

Figure 3: Screenshot of Selecting the Context
6 System Overview

SQL-Sampler was developed in C#. The desktop version runs in Windows 7 (64 bit) and can be downloaded at armstrongtable.sim.vuw.ac.nz/ArmstrongData.zip and the web-based tool is available at armstrongtable.sim.vuw.ac.nz.

Its general workflow is depicted in Figure 1. The graphical user interface (GUI) of SQL-Sampler consists of four main modules, as shown in Figure 2.

6.1 Context Module

In the context module, users select the class of constraints they consider for their application domain. This choice also determines the interpretation of null marker occurrences within the Armstrong tables produced by the tool. Possible selections include the context of i) subsumption-free SQL tables where NOT NULL constraints and SFDs are considered, ii) SQL tables where NOT NULL constraints, UCs, and SFDs are considered, iii) Codd tables where NOT NULL constraints, CUCs, and CFDs are considered, and iv) SQL tables with keys where NOT NULL constraints and UCs are considered. Note that in subsumption-free SQL tables, the class of SFDs subsumes the class of UCs, but in arbitrary SQL tables the class of SFDs does not subsume the class of UCs (Hartmann, Kirchberg & Link 2012). For contexts i), ii), and iv), the corresponding interpretation of the null marker is fixed to unk for no information. That is, a value may not exist or it exists, but is currently unknown. For context iii), the interpretation is fixed to unk for value exists, but is currently unknown.

6.2 Input Module

In the input module the user defines a table schema, a set of columns declared NOT NULL and a set Σ of UCs and/or FDs. The constraints are specified by a simple selection of the columns involved. Figure 4 shows part of the input module. Users can also open saved inputs from a file, and values for the domains of columns can be defined. These values are then used in the computation module to populate Armstrong tables. If no values are provided by users, generic values will be chosen. Users always have the choice of suitably replacing values in the Armstrong tables by new values. The system guarantees that the replacements always result in Armstrong tables.

6.3 Computation Module

For the computation module several algorithms established in the recent literature (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012, Le et al. 2012b) have been implemented to compute Armstrong tables from the context and input specified. Figure 2 shows an Armstrong table produced for our example from the introduction. For users interested in the composition and structure of the Armstrong table, other computational features can be selected. These include the computation of closures of sets of columns, the computation of maximal set families, and the computation of duplicate rows (Hartmann, Kirchberg & Link 2012). In Section 7 we illustrate the definition of maximal and duplicate sets by a detailed example, and explain their instrumental role in computing Armstrong tables.

6.4 Output Module

The output module allows the user of SQL-Sampler to modify, present and save the Armstrong table produced. Figure 2 shows the legend interface where values from the table produced can be replaced manually by the user. The interface guarantees that the tables resulting from these replacements are always Armstrong tables for the context and input specified earlier. Finally, the user has the possibility to save the Armstrong table in a file.

7 Use Case

In this section we briefly illustrate the use of SQL-Sampler on a simple example.

As use case we select the relation schema Contact which consists of the columns Address, City, and ZIP. The null-free subschema nfs(\text{Contact},_s) is defined by \text{Contact}_s = \{ZIP\}, and as the input set of constraints we select the set \[ \Sigma = \{\circ u(\text{Address}, \text{City}), o(\text{ZIP} \rightarrow \text{City})\} \] that consists of a Codd uniqueness constraint and a Codd functional dependency. We illustrate how SQL-Sampler can be used to compute an Armstrong table for Σ and nfs(\text{Contact},_s) with respect to Codd uniqueness constraints, Codd functional dependencies and NOT NULL constraints.

7.1 Context

The use case description above tells us which context needs to be defined: We select the context Codd Table, which means that the interpretation of all null marker occurrences unk in the Codd table are fixed to “value unknown at present”. The Armstrong table is computed with respect to the combined class of UCs, CFDs with NOT NULL constraints. Figure 3 shows how the context can easily be selected in SQL-Sampler.

7.2 Input Data

Figure 5 shows a screenshot of the Input Data module of SQL-Sampler after the data from the use case was filled in.
Figure 5: Screenshot of Putting in Data

Figure 6: Screenshot of Output Data
7.3 Computing Armstrong Table

Figure 6 contains a screenshot of the Armstrong table computed by SQL-Sampler on the basis of the input data. Since no domain values had been supplied by the user, the Armstrong table was populated with generic data values. The screenshot also shows the duplicate sets computed by SQL-Sampler. The definition of maximal and duplicate sets was given in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012) and is instrumental to the computation of Armstrong tables in all contexts. Here, we illustrate their instrumental role on the use case.

An Armstrong table for a given set $\Sigma$ of uniqueness constraints and functional dependencies, and a null-free subschema $nfs(T_s)$ must violate all functional dependencies not implied by $\Sigma$ and $nfs(T_s)$. For every column $A$, however, it suffices to violate those FDs $X \rightarrow A$ where the left-hand side $X$ is maximal, under set inclusion, with the property that $X \rightarrow A$ is not implied. Hence, for a column $A$ the set $\text{max}_\Sigma T_s(A)$ contains all those sets $X$ that are maximal with the property that $X \rightarrow A$ is not implied by $\Sigma$ and $nfs(T_s)$. The computation of maximal set families from $\Sigma$ and $nfs(T_s)$ is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012).

In our use case where $\text{Contact}_s = \{ZIP\}$ and $\Sigma = \{\circ(Address, City), \circ(ZIP \rightarrow City)\}$ the set $\text{max}_{\Sigma \cap \text{Contact}_s}(ZIP)$ of maximal sets for ZIP is $\{\{Address\}, \{City\}\}$, as shown in Figure 6. Indeed, $\Sigma$ and $nfs(\text{Contact}_s)$ do not imply $\circ(Address \rightarrow ZIP)$ or $\circ(City \rightarrow ZIP)$, but they do imply $\circ(Address, City \rightarrow ZIP)$.

When computing an Armstrong table for $\Sigma$ and $nfs(T_s)$ it is ensured that for each set $X$ that is maximal for some $A$, there are two different rows in the table that have matching non-null values on all the columns in $X$ and different values on $A$. This ensures that the Armstrong table violates the FD $X \rightarrow A$, and thereby also every FD $X' \rightarrow A$ where $X' \subseteq X$ holds. The exact construction of the Armstrong table depends on the context and is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012).

In our use case, for example, the maximal set $\{Address\}$ for the column ZIP is represented by the first and third row in the Armstrong table, while the maximal set $\{City\}$ for the column ZIP is represented by the first and fourth row in the Armstrong table, as shown in Figure 6.

An Armstrong table for a given set $\Sigma$ of uniqueness constraints and functional dependencies, and a null-free subschema $nfs(T_s)$ must also violate all uniqueness constraints $u(X)$ not implied by $\Sigma$ and $nfs(T_s)$. If for some column $A$, the FD $X \rightarrow A$ is not implied by $\Sigma$ and $nfs(T_s)$, then $X$ is the subset of some set that is maximal for $A$. Hence, the construction of the Armstrong table - as described above - ensures that the uniqueness constraint $u(X)$ is violated by the Armstrong table. In our use case, for example, the uniqueness constraint $\circ u(City, ZIP)$ is not implied by $\Sigma$ and $nfs(\text{Contact}_s)$, but also not the CFD $\circ u(\text{Address, Contact}_s)$. Therefore the set $\text{dup}_{\Sigma \cap \text{Contact}_s}(T)$ of duplicate rows contains those column subsets $X \subseteq T$ which are maximal with the property that $u(X)$ is not implied by $\Sigma$ and $nfs(T_s)$. In our use case, for example, the CUC $\circ u(Address, ZIP)$ is not implied by $\Sigma$ and $nfs(\text{Contact}_s)$, but the CFD $\circ (Address, ZIP \rightarrow City)$ is implied by $\Sigma$ and $nfs(\text{Contact}_s)$. Consequently, $\{Address, ZIP\}$ is a duplicate set.

When computing an Armstrong table for $\Sigma$ and $nfs(T_s)$ it is ensured that for each duplicate set $X$, there are two different rows in the table that have matching non-null values on all the columns in $X$, and either one (for Codd constraints) or both rows (for SQL constraints) carry null marker occurrences on every other column outside of $X$. Note that this is well-defined as every duplicate set $X$ contains all the columns of the null-free subschema $nfs(T_s)$, and therefore, every column outside of $X$ can carry null marker occurrences. The construction ensures that the Armstrong table violates the UC $u(X)$ and satisfies the FD $X \rightarrow T$. The exact construction of the Armstrong table depends on the context and is detailed in (Ferrarotti et al. 2011, Hartmann, Kirchberg & Link 2012). In our use case, for example, the duplicate set $\{Address, ZIP\}$ is represented by the first and fifth row, where the fifth row is $\text{null}$ on column City.

7.4 Output Data

Figure 6 shows a screenshot of how the generic values in an Armstrong table can be replaced by real data values. This can be done by activating the “Legend” button in the “Armstrong table output” menu. The user can then manually enter the real data values that replace the generic ones.

8 Some Implementation Details

Apart from computing Armstrong tables for different classes of constraints and different interpretations of null markers, SQL-Sampler should allow users to re-use and modify previous data. This ensures the effective use of the tool in the requirements acquisition process. For that reason we implemented SQL-Sampler in C# which is a powerful programming language to build a graphical user interface application. We have stored the data in an SQL Server database, consisting of nine main tables, and provided also a web application that database designers can access from everywhere without any installation concerns. To develop the web-based application we have re-implemented the algorithms in ASP.NET. We chose ASP.NET due to its properties which allow multiple users to share the same requested data for resources concurrently. ASP.NET utilizes the C# syntax, which enabled us to re-use the code already developed for the desktop version. The web-based application is hosted under the domain of .sin.vuw.ac.nz.

Users can utilize SQL-Sampler as a desktop application or as a web application. For the desktop version, the 32-bit Windows operation system and the .NET framework version 3.5 are necessary for SQL-Sampler to operate. For the web application, a system with a common Internet browser such as Firefox, Internet Explorer, or Google Chrome are required at the client side. At the server-side, SQL-Sampler requires Internet Information Services (IIS) and the ASP.NET platform 4.0 and SQL Server 2005/2008 to operate SQL-Sampler.
The algorithms have been implemented as a library of C# objects to handle four different types of Armstrong tables. For each type, relevant components have been coded to handle the sets of columns for the subsequent table, and not NULL columns, UCs, and FDs as entries in an SQL database. This enables the Web-based system to efficiently and securely perform operations on the different sets of data. It is stressed that the implementation includes authentication and authorization mechanisms, as well as access abilities for multiple users. The conceptual diagram of the SQL database is shown in Figure 7.

Figure 7: SQL Server database for SQL-Sampler

We explain the function of each table in the database next.

**ArmstrongTables** (ArmsID, ArmsName, ContID, CreateUserID, CreateDate, LastUpdate, Notes): This table contains the name, ID, and user-related information for each Armstrong table. Most information in this table is automatically updated except for the name of the Armstrong table which is provided by the user.

**RelationSchemata** (AttrID, ArmsID, AttrName, NullFree, ValueType, ValueStart, ValueEnd, ValueSets, Notes): It stores properties of columns including their name, domain type, null-free value, start value and end value which define a range of automatic data values to populate an Armstrong table. If users specify their own domain values, the values will be stored in the ValueSets column.

**FDs** (ArmsID, AttrID, LeftString, RightString, Notes): It contains the functional dependencies the user specifies during the input data stage.

**UCs** (ArmsID, AttrIDSets, AttrNameSets, Notes): It contains the uniqueness constraints the user specifies during the input data stage.

**DataInputs** (DataID, DataName, Notes, MenuURL): It contains the categories of input data which SQL-Sampler assembles to generate each type of Armstrong table. Currently, this table consists of four rows to encode names of columns, functional dependencies, uniqueness constraints, and domain values as input data categories. Data in this table cannot be updated by users.

**Contexts** (ContID, ContName, Notes): It stores the different contexts in which Armstrong tables can be computed by SQL-Sampler. Currently, this tables contains four rows to encode subsumption-free SQL tables, SQL tables, Codd tables, and SQL tables for keys, as previously described. Data in this table cannot be updated by users.

**Inputting** (ContID, DataID, Notes): This table specifies the inputs for each context in which Armstrong tables can be computed. For example, this table has four rows to specify the names of columns, null-free subschema, functional dependencies, and uniqueness constraints as input for the context SQL table; and it has three rows to specify the names of columns, null-free subschema, and functional dependencies with nulls as input for the context Subsumption-free SQL table. Data in this table cannot be updated by users.

**Computations** (CompID, CompName, Notes): It contains the possible types of outputs computed by SQL-Sampler. Currently, this tables consists of five rows to encode Armstrong tables, closures of column subsets, maximal sets, duplicate sets, and anti-keys as possible output categories for SQL-Sampler. Data in this table cannot be updated by users.

**Generating** (ContID, CompID, Notes): This table specifies the types of outputs available in each context of SQL-Sampler. In the context Codd table, for example, output is available as Armstrong tables, column set closures, maximal, and duplicate sets. Data in this table cannot be updated by users.

9 Conclusion and Future Work

Humans learn a lot from good examples. SQL-Sampler creates Armstrong sample data that visualizes perfectly the delicate interactions of SQL constraints. It can thus be used by design teams to communicate and consolidate their perceptions of an application domain with different stakeholders of the target database. Leading database design tools (e.g. ERWin) advertise the use of sample data to validate database schemata produced by existing database design methodologies, for example, ER modeling and relational normalization. SQL-Sampler produces sample data with provably perfect properties, complementing existing methodologies. The inspection of its Armstrong tables leads to the recognition of many meaningful uniqueness constraints and functional dependencies (Le et al. 2013). It thus helps design teams consolidate real-world SQL table designs, and not just relational approximations of application domains.

In future work SQL-Sampler can be enhanced to handle more classes of SQL constraints such as other types of key constraints (Hartmann et al. 2011, Thalheim 1989), cardinality constraints (Hartmann, Köhler, Link & Thalheim 2012, Liddel et al. 1993), referential constraints (Fagin & Vardi 1983), or multi-valued dependencies (Fagin 1977, Hartmann & Link 2012, 2006, Link 2012, 2008). It is also desirable to study integrity constraints under different representations of incomplete information, or in data models such as XML (Buneman et al. 2002, Ferrarotti et al. 2013, Hartmann & Link 2009, Vincent et al. 2004), RDF (Lausen et al. 2008, Paredaens 2012), or probabilistic models (Demetrovics et al. 1998, Link 2013a, Suciu et al. 2011).
10 Acknowledgement

This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand.

References


Fagin, R. (1982), Armstrong databases, Technical Report RJ3440(40926), IBM Research Laboratory, San Jose, California, USA.


A Conceptual Model for Human-Robot Collaborative Spatial Navigation

Cui Jian and Hui Shi

SFB/TR8 Spatial Cognition
University of Bremen
Enrique-Schmidt-Str. 5, 28359, Bremen, Germany
Email: {ken, shi}@informatik.uni-bremen.de

Abstract

This paper describes our work on developing effective, efficient and user-friendly interaction between a human operator and a mobile robot on performing spatial navigation tasks. In order to solve the spatially related communication problems caused by the disparity between human mental representation about spatial environments and that of a mobile robot, a qualitative spatial knowledge based four-level conceptual model is proposed. With a computational framework based on an application dependent instance of this model, high-level conceptual strategies are implemented and used to support the human-robot collaborative spatial navigation. An empirical study is then conducted to evaluate the computational framework implemented into a practical interactive system using a real environment map regarding different conceptual strategies.

Keywords: Conceptual Modelling, Qualitative Spatial Representation and Reasoning, Communication of Spatial Information, Human-Robot Interaction.

1 Introduction

As intelligent service robots are receiving more and more attention in academic and industrial areas, considerable research efforts have been dedicated to the development of effective, efficient and user-friendly human-robot interaction in different application domains (Fong et al (2003), Goodrich and Schultz (2007)). The major concern of our work is placed on solving communication problems during human-robot interaction in the domain of spatial navigation, where a mobile service robot is collaboratively controlled by an intelligent embedded system for low-level autonomous navigation and a human operator for giving high-level conceptual route instructions using natural language. The human operator can tell the robot, for example, to turn around, go straight ahead, take a right, and then pass a coffee machine on the left, until it reaches the copy room.

Much research has been devoted in this area, e.g., (Koulouri and Lauria 2009) and (Shi and Tenbrink 2009) performed corpus-based analysis on natural language route directions with mobile robots; (Kollar, et al 2010) and (Marge and Rudnicky 2010) studied the relationship between features of spatial environment and language, especially the role of natural language in route instructions; (Zender, et al 2008) and (Mozos 2010) proposed and improved a multi-layered conceptual model corresponding to spatial and functional properties of typical indoor environments based on topological information, then used this model to support a mobile robot’s indoor navigation. Diverging from these methods and models concentrating on empirical data, natural language and topological conceptual information, our work is focusing on human perspectives according to the following two important aspects.

First, in human-robot collaborative navigation, human operators usually use natural language expressions containing qualitative relations and conceptual landmarks (Hirtle 2008), such as “go to the end of the corridor, turn right, and then go until the coffee machine on the left”, while mobile robots work on quantitative level and can only interpret instructions with quantitative data, such as “145.0 meters ahead, then make a 37.5 degree turning, ...”. There is apparently a gap between a human operator and a mobile robot if they want to communicate with each other. Much research has been focusing on applying mathematical well-founded qualitative spatial calculi and models to represent and reason about spatial environments (e.g. Ligozat and Renz (2004), Schultz, et al (2006), Wolter and Lee (2010), Kurfess, et al (2011)). Adding to this body of literature, using qualitative spatial knowledge as an intermediate layer for the intuitive human-robot communication has been viewed as the foundation of our work.

Furthermore, providing a sequence of route instructions is a rather complex process for the human operator, since spatially-related communication problems could easily occur if spatial objects are wrongly localized or a certain instruction is wrongly given due to a certain spatial situation, e.g., a coffee machine cannot be found after taking a right turn, or a room to be passed is not on the left as expected (Reason (1990) and Bugmann (2004)). Therefore, an effective mechanism is needed for the mobile robot and the human operator to collaboratively identify the problems and negotiate possible solutions with each other.

Thus, in order to bridge the interaction gap between the human operator’s qualitative spatial mental model and the mobile robot’s quantitative representation, as well as supporting the high-level collaborative negotiation of spatially-related communication problems, we proposed a...
qualitative spatial knowledge based four-level conceptual model: the Qualitative Spatial Beliefs Model (QSBM). This model was first proposed in (Shi and Krieg-Brückner 2008), and then extended and implemented with a computational framework (Jian, et al 2009) and a set of high-level conceptual strategies to support collaborative human-robot spatial navigation (Shi, et al 2010). Two conceptual strategies were evaluated and compared in (Jian, et al 2010). With the further development of the conceptual model based computational framework and the integration into a practical interactive system for a mobile robot, the current paper reports on a new high-level conceptual strategy for resolving more spatially-related human-robot communication problems, as well as an empirical study, which was conducted to test the current system with the focus of evaluating the new conceptual strategy and its comparison with the previous strategies.

The remainder of the paper is organized as follows. Section 2 presents the qualitative spatial knowledge based four-level conceptual model and one of its application dependent instance with conceptual strategies to solve the spatially related communication problems. Section 3 introduces the computational framework that implements the conceptual model. Section 4 then describes the empirical study to evaluate a practical interactive system regarding the model-based conceptual strategies and Section 5 discusses the results of the study. Finally, Section 6 concludes the paper and gives an outline to our future work.

2 A Qualitative Spatial Knowledge based Four-Level Conceptual Model

2.1 The Overview of the General Model

According to the perspective of human operators, spatial environments are not represented with quantitative data as a mobile robot does, but with conceptual objects or places and their qualitative spatial relations. Accordingly, Qualitative Spatial Beliefs Model (QSBM), a qualitative spatial knowledge based conceptual model is developed to model a mobile robot’s beliefs for supporting more intuitive communication with human operators. Figure 1 illustrates the general QSBM with a four-level structure.

Figure 1: The QSR-based four-level conceptual model: Qualitative Spatial Beliefs Model (QSBM)

The basic level is the QSR Model level, which contains the most basic theoretical foundation of the QSBM model: qualitative spatial calculi for different application requirements, such as Double-Cross Calculus (Freksa, 1992), Cardinal Directions (Frank, 1991), 9+ Intersection (Kurata, 2008), etc.

Based on the chosen qualitative spatial calculus, a basic conceptual model can be constructed and serves as the fundamental conceptual level. This level only contains qualitative spatial information and the basic calculating and reasoning mechanism with respect to the connection between the chosen calculus and the navigation environment. It can be seen as a black box holding a conceptual qualitative spatial knowledge based representation of a spatial environment with two basic functions: Qualify for qualifying quantitative information into qualitative relations, and CalculateRelation for calculating additional qualitative spatial relations with qualitative spatial relations between objects using calculus-based qualitative spatial reasoning.

The application level consists of a set of most atomic application-dependent update rules, which correspond to all the possible user-uttered route instructions to a mobile robot in collaborative spatial navigation. For instance, the update rule Reorientation can refer to the instruction “turn left”, Redirection can interpret “take the next junction on the left”. Feature-based Motion concerns instructions with features of objects or landmarks, such as “go around the big laboratory” (see (Gondorf and Jian, 2011)), and Learning-based Motion represents those instructions requiring the robot to augment its conceptual knowledge by learning new landmarks or disambiguating landmarks, such as “the third office is the directory’s office, pass by it”, etc. Each update rule is used to update the state of the spatial representation on the conceptual level with respect to its formal definition based on a chosen calculus and the related qualitative spatial reasoning on the QSR Model level.

On the strategy level, high-level conceptual strategies are developed to assist in interpreting a sequence of route instructions and if possible, resolve the spatially-related communication problems during the collaborative spatial navigation. Basically, each conceptual strategy defines its own mechanism for appropriately choosing and applying atomic update rules on the application level.

In general, the QSBM is a conceptual model for applying qualitative spatial knowledge to represent a spatial environment, qualitative spatial reasoning to define a set of application-dependent update rules to update the conceptual representation, and conceptual strategies to manage the atomic update rules to support high-level spatially-related human-robot communication. With the flexibility and expandability provided by the multi-level structure, further application scenarios can be supported by using different qualitative calculi on the QSR model level, more application-dependent actions can also be added on the application level, or new high-level strategies can also be implemented to resolve more communication problems, while each of these changes/extensions requires only limited adaptation on the other levels in QSBM.

Specifically, since qualitative spatial calculi at the QSR Model Level are well studied, formal details about the other three levels of an instance of QSBM will be given in the rest of the chapter.

2.2 A DCC-based QSBM

Considering the current requirement of the collaborative spatial navigation scenarios, double-cross calculus (DCC) is selected as the basic QSR model and a DCC-based...
QSBM is developed (Shi, et al 2010) and introduced according to the conceptual, application and strategy level as follows.

2.2.1 The Conceptual Level

In mobile robot navigation, one of the most important basic models is called Route Graph (Werner, et al 2000). Route graphs are a special class of graphs, with graph nodes representing conceptual places at geographical positions regarding a quantitative reference system, and graph edges or route segments, each of which is directed. Conventional route graphs cannot only be used as quantitative representation of spatial environments for mobile robots’ navigation, they also capture the topological knowledge of space from human perspective and therefore have the potential of intermediate layers for human operators. However, the gap between qualitative representation of conventional route graphs and qualitative knowledge based mental construct of human operators remains a problem preventing a more direct interaction.

**Figure 2:** a) one part of a conventional route graph; b) the orientation frame of Double Cross Calculus with 15 qualitative spatial relations.

On the other hand, Double Cross Calculus (DCC (Freksa, 1992)) divides the 2-dimensional space with a directed segment into disjoint grids (see Fig. 2 b)), which defines 15 meaningful qualitative spatial relations. Thus, a DCC model can be used as a local navigation map from an egocentric perspective and support the interaction with human operators in a local navigation scenario.

By combining the structure of a conventional route graph and the DCC model, the conceptual route graph (CRG) is developed (Shi and Krieg-Brückner, 2008). A CRG inherits the topological structure from a conventional route graph, where quantitative information is completely replaced by the DCC relations between graph nodes and route segments. Formally, a CRG of a spatial environment is defined by a tuple of four elements: (M, P, V, R):

- M is a set of landmark-place-pairs in the environment, specifying the locations of all the landmarks at places in P, such as an {office: x1}, or a {kitchen: x2}.
- P is a set of topological places, or the graph node in a CRG, such as x1 or x2.
- V is a set of vectors, each of which is directed from one place to another place, such as AB.
- R is a set of relation-pairs, which specify the DCC relations between places and vectors. A relation pair is written as <AB, LeftFront, x1>, meaning that x1 is in the LeftFront grid of AB.

Therefore, the CRG for the simple spatial environment illustrated in Fig. 2 b) is represented as:

```plaintext
<crg, pos = AB>
```

And a state of a DCC-based QSBM model, which is stored as a mobile robot’s internal representation about current spatial situation, can then be represented for example as:

```plaintext
<crg, pos = AB>
```

This means that the mobile robot is now located at place A and looking at the direction of place B, with an office on the LeftFront position and kitchen at the Back.

2.2.2 The Application Level

In order to support the application scenarios of human-robot collaborative spatial navigation, a set of route instructions such as “turn left”, “take the next junction on the right”, “pass by the office on the left”, etc., should be interpreted by the mobile robot. According to the formal definition of the DCC-based CRG on the conceptual level, a set of low-level update rules regarding the most common route instructions for mobile robots are developed on the application level and used to update the state of the DCC-based QSBM, i.e., the state of a mobile robot about spatial environment.

Each update rule is specified with the following three elements:

- a name (followed by RULE), which identifies a class of most common route instructions,
- a set of preconditions (followed by PRE), under which this update rule can be applied, and
- an effect (followed by EFF), describing how the state of the DCC-based QSBM is updated after applying the update rule.

As examples, the update rules for reorientation and directed motion are presented as follows:

- **Reorientation** refers to the simplest route instructions, which change the current orientation of a robot, such as “Turn left”, “Turn right” and “Turn around”. In general, the precondition is whether a robot can find a CRG vector satisfying two conditions: 1. it is originated from the current place and 2. It is targeted at a place that has the desired spatial relation with the current position; the effect is that the robot position is updated as that found CRG vector, formally described as:

  ```plaintext
  RULE: Reorientation
  PRE: pos = P0, \( \exists P, P_1, V \in P, P_1, \text{dir}, P_1 > \)
  EFF: pos = P1
  ```

Concretely, the rules indicates that the robot is currently at the place \( P_0 \) and faces the place \( P_1 \) (\( P_0, P_1 \) is a CRG vector), if there exists a CRG vector \( P_0, P_1 \) with a targeting place \( P_2 \) such that the spatial relation of \( P_2 \) with respect to the route
segment $P_0P_1$ (i.e. the current position) is the desired direction $dir$ to turn, i.e., $<P_0P_1, dir, P_2>$, then the current position will be updated as $P_2P_3$ after applying this update rule.

- **Directed Motion** defines the class of the route instructions that usually contain a motion action and a turning action changing the direction of the continuing motion, such as “take the next junction on the right”. These instructions usually involve with a landmark (e.g. the “junction”), until which the robot should go, and a direction (e.g. on the “right”), towards which the robot should turn. For example, in general, for the route instruction “take the next corridor on the right”, the first corridor on the right from the robot’s current position needs to be identified first. Thus, the update rule for directed motions with the first landmark and a turning direction is specified as:

\[
\text{RULE: DirectedMotionWithFstLandmarkAndDir}
\]

\[
\text{PRE: pos} = P_0P_1, \\
\forall P_2, P_3 \in \mathcal{E}V \ ( \langle P_2, P_3, dir, P_2 \rangle \land \langle P_0P_1, Front, P_0 \rangle) \\
\text{EFF: pos} = P_2P_3
\]

In this rule, $l$ is the targeted landmark and $dir$ is the direction to turn to; The first precondition specifies that the robot should find a CRG vector $P_0P_1$, such that the targeted landmark is located at $P_2$, the spatial relation between $P_3$ and the segment $P_0P_1$ is the desired direction $dir$ and $P_2$ is in front of the robot’s current position; The second precondition limits that $P_2$ is the first place referring to the given landmark at the given direction, instead of an arbitrary one; this condition is satisfied if there exists a place $P_3$ with the same feature as $P_2$, $P_3$ must be ahead of $P_0P_1$ from the current perspective. The effect is that, the robot position is updated to $P_2P_3$ after applying this rule. Similarly, other variants of directed motions, such as “go straight ahead”, “go right” or “take the second left” can be specified with similar update rules accordingly.

### 2.2.3 The Strategy level

With the update rules defined on the application level, single route instructions can be interpreted. However, in human robot collaborative navigation, human operators usually give a sequence of route instructions to the mobile robot. In this case, if a certain route instruction is wrongly given, spatially related communication problems could easily occur, because taking the wrong route instruction could cause problems of interpretation of the subsequent route instructions, which could result in failure of the entire interpretation or even lead to a completely unexpected route.

In order to resolve these problems, a set of high-level conceptual strategies are developed on the strategy level, which apply the low-level update rules accordingly and appropriately according to different principles and methods. Among them, the two most important conceptual strategies are briefly introduced as follows.

#### 2.2.4 Reasoning with Backtracking

With the qualitative spatial reasoning on the QSR model level, the preconditions of update rules on the application level can easily be checked, this is in fact the most straightforward way to see if a sequence of route instructions can be interpreted. However, there are often situations where the failure of the interpretation of some instructions is caused by a previously incorrect instruction, e.g. see the situation in Fig. 3. The robot is located at the thick red arrow and the instructions are: “go straight ahead, then go left, and then turn right, and go until the kitchen on the right.” A simple check fails on interpreting the fourth instruction “go until kitchen on the right”, because there is no kitchen ahead after taking a right turn as the previous instruction. However, by taking one step backwards, if the third instruction is changed from right to left, then the last instruction can also be interpreted properly.

**Figure 3: An example of a wrong instruction**

Thus, the strategy “Reasoning with Backtracking” (abbr. RwB) interprets the route instructions as the straightforward way does, checking every precondition as usual. Yet after applying each update rule, the state of the updated QSBM is also saved in an interpretation history. Once one instruction cannot be interpreted, the previous state of the QSBM can be reloaded as the current state and possible suggestion can be made based on the previous instruction, such as “turn left” instead of “turn right” in the example in Fig. 3. As a result, the checking of the preconditions of the remaining route instructions can be resumed based on the suggested route instruction, and a possible route matching the entire sequence of route instructions can be found.

The RwB strategy has been evaluated and compared with other conceptual strategies and the positive results were reported in (Jian, et al 2010).

#### 2.2.5 QSR-Value Tuples based Searching

During the development and integration of the QSBM model into an interactive system to be used by a mobile robot, a new class of spatially-related communication problems is identified. Fig. 4 illustrates one example of these problems.

**Figure 4: An example of a “missing” instruction**
In this example, the robot is located at the thick red arrow and the instructions are “go straight, then left, then go until the kitchen on the right”. From the perspective of the human operator, the kitchen is located directly on the right side, and therefore the operator simply ignores a turning point that is in the conceptual representation but not in his/her mental representation. However, after taking a right turn, the last instruction “go until kitchen on the right” cannot be interpreted, because there is no continuing possibility as shown in Fig. 4.

These problems cannot be solved by the RwB strategy, because the RwB strategy can only provide suggestions if there exists a wrong route instruction, while in these situations one route instruction is missing. Thus, the strategy “QSR-Value Tuples based Searching” (abbr. QSRVT) was developed. For each outgoing direction of each turning node in a conceptual route graph during the interpretation, a QSR weighted value tuple is defined as:

\[
\text{QSR-\text{v}} = \sum_{i=1}^{\text{current}} \text{mr}_i \times \text{sr}_i
\]

where \(\text{route}\) is the currently taken route, \(\text{instructions}\) is the set of all the along this route interpreted instructions, and \(\text{qsr}\_\text{v}\) is the cumulative value calculated by

where \(\text{mr}_i\) is the matching rate by comparing the desired qualitative spatial direction with the current route direction while interpreting the \(i\)-th instruction, \(\text{sr}_i\) is the success rate of interpreting the \(i\)-th route instruction, and \(\text{current}\) is the index of the current route instruction.

The QSRVT strategy first initializes an empty set of QSR-value tuples at the starting position of the robot. This set of QSR-value tuples is then automatically updated and expanded by the searching agents of the QSRVT strategy, while they are travelling along all paths (according to the branching of each turning node) in the QSBM. Finally, a full set of QSR-value tuples is generated and the QSR-value tuple with the highest QSR-weighted value is either the best possible solution for interpreting the route instructions or contains the most relevant information to provide possible suggestion to resolve the spatially-related communication problems.

As an example, Fig. 5 briefly illustrates how the QSRVT strategy solves the problem in Fig. 4.

![Figure 5: A simple process of the QSRVT strategy](image)

Figure 5: A simple process of the QSRVT strategy

After interpreting the first two instructions “go straight” and “go left”, the searching comes to the turning node x3. There are two possible directions going out of x3 and accordingly two more QSR-value tuples are added. In this situation, the last instruction cannot be interpreted with the left going route while it can be interpreted with the right going one. Therefore, the instructions are interpreted with the route x0->x1->x2->x3->x5, since the QSR-value tuple has the highest value 2.5.

### 3 A Conceptual Model based Computational Framework

Based on the introduced QSBM, including update rules and the high-level conceptual strategies, we developed SimSpace, a conceptual model based computational framework for supporting the implementation of QSBM into a practical interactive system to be used by a mobile robot.

#### 3.1 General Architecture

According to the Model-View-Controller architecture (originally from (Burbeck 1987)), the general architecture of SimSpace consists of a Model component **Spatial Envioronment**, an optional View Component **SSGUI** and a Controller **SpaceManager**.

**Spatial Environment** maintains the current state of the QSBM instance, i.e., the conceptual route graph and the hypothesis of the robot position in the CRG, as well as the optional quantitative spatial environment (QuanSE) for quantitative data and the optional feature map (FM) component containing the conceptual information.

**SSGUI** is the graphical user interface of SimSpace. It is an optional component and is only used if the SimSpace system is started as a stand-alone application. It visualizes the spatial environment with quantitative and conceptual descriptions, interacts with a human user who is giving the natural language route instructions, and communicates with the Space Manager for interpretation of incoming route instructions as well as outgoing system responses.

**Space Manager** is the central processing component of SimSpace, it consists of the following five functional components:

- Basic Creator creates a spatial environment instance with quantitative and conceptual data according to the quantitative map data, if given.
- Concept Manager manages an ontology database of the conceptual knowledge, such as names of landmarks or persons, how they are conceptually related, etc. It is used to interpret the conceptual terms in the natural language route instructions.
• QSBM Manager connects with QSBM Reasoner and generates a QSBM instance according to a qualitative spatial calculus on the QSR model level and a quantitative environment if given, manipulates and updates an empty or existing QSBM instance with the application dependent update rules on the application level, and saves the updated QSBM instance into a XML-based specification with .crg file extension, if needed.

• High-Level Planner implements the high-level conceptual strategies to apply appropriate update rules to interpret route instructions and resolve spatially-related communication problems.

3.2 The Interpretation of Route Instructions in SimSpace

The SimSpace system can interpret a sequence of human route instructions in the following steps:

• The sequence of route instructions is firstly parsed into a list of predefined semantic representations.

• According to the activated high-level conceptual strategy, each semantic representation is assigned with an applicable low-level update rule.

For each low-level update rule, its preconditions are instantiated. Taking the sample instruction “go until the kitchen on the right” in the previous section, the update rule GoUntilRight is applied and by substituting the current robot position with the CRG vector AB and the location of the kitchen is found as Pkit, the second precondition is instantiated to:

$$\exists P, P_1 \in V. (\text{kitchen} : P_{\text{kit}}) \land \langle AB, \text{RightFront}, P_1 \rangle \land \langle AB, \text{RightBack}, P_1 \rangle \land \langle AB, \text{Front}, P_1 \rangle \land \langle AB, \text{Front}, P_1 \rangle$$

Then with the support of the SparQ toolkit, the instantiated preconditions are checked against the current state of the QSBM.

If the current state matches the instantiated precondition, the current robot position is updated to $$P_2 P_3$$ and a message object containing the success information is returned.

If the current state provides e.g. the relations:

$$(\text{kitchen} : P_{\text{kit}}) \land \langle AB, \text{LeftFront}, P_{\text{kit}} \rangle$$

This means, the kitchen is located on the left side from the perspective of the robot and therefore, $$\langle AB, \text{RightFront}, P_{\text{kit}} \rangle$$ in the precondition cannot be satisfied. In this case, SimSpace creates a corresponding message which contains necessary information for indicating the failure of the interpretation and/or generating suggestion.

• According to the conceptual strategy and the returned message, either the interpretation continues if possible, or strategy dependent process is performed (e.g. in the RwB or QSRVT strategy), or appropriate responses or suggestions are made and presented back to the human user.

On the one hand, SimSpace can be used as a stand-alone evaluation platform for visualizing spatial environments, generating corresponding QSBM instances and testing the interpretation of natural language route instructions. On the other hand, it can also be used as a well encapsulated module and integrated into an interactive system to be used by a mobile robot to assist in the interaction with human operators.

4 An Empirical Study

In order to evaluate the qualitative knowledge based conceptual model and its implementation into a practical interactive system regarding the two different high-level conceptual strategies: reasoning with backtracking and QSR-value-tuples based searching, an empirical study was conducted.

4.1 Participants

Altogether 18 university students, with no background knowledge on cognitive science and therefore considered as novice users, participated in the study, in which 9 of them were interacting with the system using the strategy reasoning with backtracking, while the other 9 were testing the system with QSR-value-tuples based searching.

4.2 Stimuli and Apparatus

All stimuli were the same for each participant during the interaction process, e.g., visual stimuli were presented on a graphical user interface on a laptop displaying a map of an indoor environment with named landmarks, a robot avatar showing the current position of the robot, a possibly highlighted route along which the robot is going, and the clearly emphasized text of system response with respect to participants’ instructions (see Fig. 7); audio stimuli of the system response were also generated as complementary feedback and played via the external speaker of the same laptop at a well-perceivable volume.

![Figure 7: The graphical user interface with all visual stimuli](image)
The same map of a floor plan of an indoor environment with the same virtual landmarks within this environment was used throughout the study.

The interactive system was a networked software system consisting of two laptops: one laptop, called the system laptop, hold the actual interactive system, which included the graphical user interface, interaction manager, speech synthesizer and the spatial knowledge processing component SimSpace that implemented the qualitative knowledge based conceptual model and the conceptual strategies; the other laptop, called the speech recognizer laptop, run a graphical interface, which was only operated by a human investigator and used to transfer the spoken natural language instructions to the system laptop via wireless network. The time for inputting natural language instructions is significantly shortened with a well-designed group of function-buttons on the speech recognizer laptop, so that only two seconds on average were needed for transferring utterances to the system laptop. As a result, the whole system was simulated as if each participant was giving instructions to the system using spoken natural language directly.

All participants were accompanied by the same investigator, who gave the introduction to the study and the system at the beginning, and input the natural language instructions of each participant into the speech recognizer laptop during the task performing through pressing the function buttons. An internal automatic logging program of the system was used to collect interaction data such as dialogue turns, utterances, event time, and so on, while the standard audio recorder of windows recorded the whole dialogic interaction process.

Two questionnaires were conducted. The first one is called spatial ability questionnaire, which includes questions regarding abilities of describing routes to others, inquiring ways from others and using map in everyday life. This questionnaire aims to get the subjective assessment of the spatial ability with the interactive system. Both questionnaires were based on 5-point Likert scale.

### 4.3 Procedure

For each test a participant had to undergo four steps:

1. **Self-assessment:** the participant was asked to fill the spatial ability questionnaire.
2. **Introduction:** the participant was given a brief introduction to the system and the following test runs, which included how to interact with the system and what to expect during the interaction.
3. **Interaction:** each participant was given five different tasks, each of which contains a starting position and a goal position. Only spoken language instructions were used to tell the robot go from the starting position to the goal position. In order to collect more data and to produce more problem situations, for each task the participant had to describe two different routes or utter two different descriptions. Each task was ended, if either the goal position was reached, or the participant gave up trying.
4. **Evaluation:** after interacting with the system, the participant was asked to fill in the evaluation questionnaire.

### 5 Results and Discussion

According to the general view of the well accepted evaluation framework Paradise (Walker, et al 1997), the performance of an interactive system can be measured via the effectiveness, the efficiency and the user satisfaction. Thus, we have performed the analysis of the data from the interactive system on the two conceptual strategies with respect to these three aspects.

Even with the relatively small group of the participants (9 persons in each group), the authors believed that the comparison of the presented empirical results between the two groups can be considered representative, since the grouping was performed in a random manner, and furthermore, the results of the self-assessment of the spatial ability are similar between the two groups with the values of 53.2 and 51.9 on average.

#### 5.1 Regarding the Effectiveness

The study was conducted with a Wizard of Oz setting without an automatic speech recognizer, therefore, the effectiveness of the interactive system could only depend on whether the subtasks were successfully performed, namely, whether the navigation goals were reached or not. 10 Goals were supposed to be reached by each participant. With 9 participants for one strategy, the number of reached goals are counted and summarized in table 1.

<table>
<thead>
<tr>
<th></th>
<th>RwB</th>
<th>QSRVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reached Goals (percentage)</td>
<td>85 (94.4%)</td>
<td>90 (100%)</td>
</tr>
</tbody>
</table>

Table 1: Effectiveness with RwB and QSRVT

For both strategies, the effectiveness of performing navigation tasks with the interactive system is very good. The participants using the RwB strategy reached 85 goals out of 90, while the ones using the QSRVT strategy reached all the goals.

#### 5.2 Regarding the Efficiency

In order to find out how efficiently each participant was assisted with the interactive system using the two different strategies, the automatically logged data were analyzed according to the average elapsed time and interaction turns for each task. The results are summarized in table 2.

<table>
<thead>
<tr>
<th></th>
<th>RwB</th>
<th>QSRVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Elapsed Time (s)</td>
<td>87.37</td>
<td>48.12</td>
</tr>
<tr>
<td>Average Interaction Turns</td>
<td>7.14</td>
<td>0.013</td>
</tr>
</tbody>
</table>

Table 2: Data concerning efficiency for each participant and each task

From a general perspective for task performing, a very good efficiency is shown with 87.37 seconds and 7.14
interaction turns on average for each task with the RwB strategy, since this also includes some very long system responses, some of which even needed over 20 seconds to be played. The standard deviation of 33.13 for elapsed time is however a bit high, this is mainly due to one certain participant who confused the left/right relations too often and used over 150 seconds on average to finish one task, which, however, is not common for the other participants.

Moreover, the performance efficiency with the QSRVT strategy is much better: each participant only used 48.12 seconds and 4.07 turns on average to finish one task. The p-values of 0.007 and 0.013 also indicate that the participants with the QSRVT strategy could perform tasks significantly more efficiently than those with the RwB strategy.

5.3 Regarding the User Satisfaction

Regarding the user satisfaction about the interactive system, the subjective data of the evaluation questionnaire filled by each participant after task performing were analysed and summarized in table 3.

<table>
<thead>
<tr>
<th></th>
<th>RwB</th>
<th>QSRVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Response</td>
<td>3.36</td>
<td>4.0</td>
</tr>
<tr>
<td>General Support</td>
<td>3.94</td>
<td>4.25</td>
</tr>
<tr>
<td>Future use</td>
<td>3.72</td>
<td>3.94</td>
</tr>
<tr>
<td>Total</td>
<td>3.68</td>
<td>4.06</td>
</tr>
<tr>
<td>Total / Skill</td>
<td>0.07</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Table 3: Data concerning user satisfaction

The overall user satisfaction of the interactive system with the RwB strategy for each participant is considered at a satisfactory level with the total average value of 3.68 and standard deviation 0.63. Specifically, they found the system response sufficiently understandable with the value 3.36, felt supported by the system with the value of 3.94 and they would recommend the system with the value of 3.72. The standard deviations of 0.79 for system response and 0.80 for general support are a bit higher, this is because of the special situations where the RwB strategy encounters with missing instructions and therefore the system could not provide very useful information about the communication problems.

Meanwhile, the user satisfaction of the system with the QSRVT strategy was improved from every perspective, 4.0 for the system response, 4.25 for the general support, 3.94 for the future use and all together 4.06.

With the data from the self-assessment questionnaire, a skill value is calculated and shows how confident each participant considers him- or herself to be with spatially-related tasks. The ratios of the total satisfaction degree and the skill value of 0.07 and 0.08 also roughly indicate that, the QSRVT strategy better assists the participants also in a more or less subjective manner than the RwB strategy does.

6 Conclusion and Future Work

In this paper we reported our work on using conceptual model to support human robot collaborative navigation, focusing on the following three important aspects:

- the design and development of a qualitative spatial knowledge based multi-level conceptual model for human robot interaction,
- the implementation of the conceptual model and the model-based high-level conceptual strategies within a general computational framework, and
- the evaluation of an interactive system built on the conceptual model, framework and strategies.

The positive empirical results validated our effort on developing and implementing the proposed conceptual model and framework. It was also shown that, the model based high-level conceptual strategies, especially the strategy of QSR-value tuple based searching can assist the mobile robot to clarify more spatially-related communication problems and better support the human-robot collaborative navigation.

The presented work served as a fundamental step towards building robust, effective, efficient, user-friendly models, frameworks and interactive systems in spatially-related applications. The integration of the conceptual model, framework and strategies into a real mobile robot for spatial navigation with untrained human operators is being conducted. For the strategy of QSR-value tuple based searching, learning-based QSR-value updating is being investigated. We are also considering adding other qualitative spatial calculi into the QSR model level to support further application, such as object localization within complex buildings. Human-robot collaborative exploration in unknown or partially-known spatial environments is also a work package to be pursued.

Acknowledgement

We gratefully acknowledge the support of the German research foundation (Deutsche Forschungsgemeinschaft, DFG) through the Collaborative Research Center SFB/TR8 spatial cognition and the University of Bremen.

7 References


Hois, J., Ross, R.J., Kelleher, J.D., and Bateman, J.A. (eds).


Author Index

Dalpiaz, Fabiano, 3
Dam, Hoa Khanh, 55
Etzlstorfer, Juergen, 19
Ferrarotti, Flavio, 71
Ghose, Aditya, 55
Grossmann, Georg, iii, viii
Henderson-Sellers, Brian, 49
Jian, Cui, 81
Kapsammer, Elisabeth, 19
Kusel, Angelika, 19
Le, Van Tran Bao, 71
Link, Sebastian, 71
Mafazi, Shamila, 39
Mayer, Wolfgang, 39
Mylopoulos, John, 3
Nakatani, Takako, 65
Ochimizu, Koichiro, 11
Rogers, Ben, 49
Saeki, Saeki, iii, viii
Santiputri, Metta, 55
Saraswati, Ayu, 55
Schütz, Christoph, 29
Schoenboeck, Johannes, 19
Schrefl, Michael, 29
Schwinger, Wieland, 19
Shi, Hui, 81
Souza, Vitor E. Silva, 3
Stumptner, Markus, 39
Tsumaki, Toshihiko, 65
Wimmer, Manuel, 19
Wischenbart, Martin, 19
Listed below are some of the latest volumes published in the ACS Series Conferences in Research and Practice in Information Technology. The full text of most papers (in either PDF or Postscript format) is available at the series website http://crpit.com.

Volume 124 - Database Technologies 2012

Volume 125 - Information Security 2012

Volume 126 - User Interfaces 2012
Edited by Baoqin Chen, Flinders University, Australia and Ross T. Smith, University of South Australia, Australia. January 2012. 978-1-921770-07-4.

Volume 127 - Parallel and Distributed Computing 2012

Volume 128 - Theory of Computing 2012
Edited by Juliana Monte, University of Sydney, Australia. January 2012. 978-1-921770-09-8.

Volume 129 - Health Informatics and Knowledge Management 2012

Volume 130 - Conceptual Modelling 2012

Volume 133 - Australian System Safety Conference 2011

Volume 134 - Data Mining and Analytics 2012
Edited by Zhaoping, Department of Immigration and Citizenship, Australia, Juyong Li, University of South Australia, Paul J. Kennedy, University of Technology, Sydney, Australia and Peter Christen, Australian National University, Australia. December 2012. 978-1-921770-14-2.

Volume 135 - Computer Science 2013
Edited by Bruce Thomas, University of South Australia, Australia. January 2013. 978-1-921770-20-3.

Volume 136 - Computing Education 2013
Edited by Angela Carboni, Monash University, Australia and Jacqueline Whalley, AUT University, New Zealand. January 2013. 978-1-921770-21-0.

Volume 137 - Database Technologies 2013
Edited by Hua Wang, University of Southern Queensland, Australia and Rui Zhang, University of Melbourne, Australia. January 2013. 978-1-921770-22-7.

Volume 138 - Information Security 2013

Volume 142 - Health Informatics and Knowledge Management 2013
Edited by Kathleen Gray, University of Melbourne, Australia and Andy Keane, University of South Australia, Australia. January 2013. 978-1-921770-26-5.

Volume 143 - Conceptual Modelling 2013

Volume 144 - The Web 2013
Edited by Helen Ashman, University of South Australia, Australia, Quan Z. Sheng, University of Adelaide, Australia and Andrew Trottman, University of Otago, New Zealand. January 2013. 978-1-921770-15-9.