Marlon Dumas
Manfred Reichert
Ming-Chien Shan (Eds.)

LNCS 5240

Business Process Management

6th International Conference, BPM 2008
Milan, Italy, September 2008
Proceedings

Springer

Marlon Dumas   Manfred Reichert
Ming-Chien Shan (Eds.)

Business Process Management

6th International Conference, BPM 2008
Milan, Italy, September 2-4, 2008
Proceedings

Springer

Volume Editors

Marlon Dumas
University of Tartu
J Liivi 2, 50409 Tartu, Estonia
E-mail: marlon.dumas@ut.ee

Manfred Reichert
University of Twente
Department of Computer Science
Information Systems Group
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: m.u.reichert@cs.utwente.nl

Ming-Chien Shan
SAP Labs
LLC 3475 Deer Creek Road, Palo Alto, CA 94304, USA
E-mail: ming-chien.shan@sap.com

Library of Congress Control Number: 2008933578

CR Subject Classification (1998): H.3.5, H.4.1, H.5.3, K.4, K.6, J.1

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

ISSN 0302-9743
ISBN-10 3-540-85757-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-85757-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12465750 06/3180 5 4 3 2 1 0

Preface

BPM 2008 is the sixth international conference in a series that provides the most distinguished specialized forum for researchers and practitioners in business process management (BPM). The conference has a record of attracting innovative research of the highest quality related to all aspects of BPM including theory, frameworks, methods, techniques, architectures, standards, and empirical findings.

BPM 2008 was held in Milan, Italy, on September 2–4, 2008, and was organized by the Information Systems Research Group of the Department of Electronics and Information of the Politecnico di Milano. The present volume contains the research, industry, and prototype demonstration papers accepted for presentation at the conference.

This year, we received 154 full paper submissions. These submissions came from authors located in 36 different countries, geographically distributed as follows: 101 submissions originated from Europe, 19 from Australia, 16 from Asia, 14 from America, and 4 from Africa. As in previous years the paper selection process was extremely competitive. After a thorough refereeing process in which every paper was reviewed by between 3 and 5 program committee members, only 23 of the 154 submissions were accepted, leading to an acceptance rate just below 15%. Among the 23 accepted papers, there are 20 research papers and 3 industry papers.

In addition to these 23 papers, 3 invited keynote presentations were delivered by Paul Harmon (Executive Editor and Founder, BPTrends, USA), Michael Rosemann (Queensland University of Technology, Australia), and Peter Dadam (University of Ulm, Germany). We are very grateful to the keynote speakers for their contributions.

In conjunction with the main conference, nine international workshops took place the day before the conference. These workshops have fostered the exchange of ideas and experiences between active BPM researchers, and stimulated discussions on new and emerging issues in line with the conference topics. The proceedings with the papers of all workshops will be published in a separate volume of Springer's *Lecture Notes in Business Information Processing* series. Finally, the present volume contains 6 prototype demonstration papers that were selected out of 15 demo submissions by the demo chairs and the reviewing committee they appointed.

We owe special thanks to all senior and regular members of the Program Committee of BPM 2008 as well as their sub-referees for their work. We are also very grateful to the numerous people who were involved in the organization of the BPM conference and its satellite events. In particular, we would like to thank the General Co-chairs – Barbara Pernici and Fabio Casati – as well as Danilo Ardagna for his outstanding support as Organization Chair of BPM 2008. We would also like to thank Massimo Mecella and Jian Yang (Workshop Co-chairs), Malu Castellanos and Andreas Wombacher (Demo Co-chairs), Vincenzo d'Andrea and Heiko Ludwig (Tutorial / Panel Co-chairs), and the many other colleagues who contributed to the success of BPM 2008. Finally, we thank the conference sponsors for their support in making BPM 2008 another successful event in the series.

June 2008

Marlon Dumas
Manfred Reichert
Ming-Chien Shan

Conference Organization

General Chairs

Fabio Casati, University of Trento, Italy
Barbara Pernici, Politecnico di Milano, Italy

Program Chairs

Marlon Dumas, University of Tartu, Estonia & Queensland University of Technology, Australia
Manfred Reichert, University of Ulm, Germany

Industry Chair

Ming-Chien Shan, SAP Labs, Palo Alto, USA

Senior Program Committee

Wil van der Aalst, The Netherlands
Boualem Benatallah, Australia
Peter Dadam, Germany
Jörg Desel, Germany
Schahram Dustdar, Austria
Johann Eder, Austria
Gregor Engels, Germany
Claude Godart, France
Kees van Hee, The Netherlands
Arthur ter Hofstede, Australia
Stefan Jablonski, Germany
Frank Leymann, Germany
Barbara Pernici, Italy
Michael Rosemann, Australia
Jianwen Su, USA
Mathias Weske, Germany

Program Committee

Alistair Barros, Australia
Djamal Benslimane, France
M. Brian Blake, USA

Shawn Bowers, USA
Chris Bussler, USA
Jorge Cardoso, Portugal
Jinjun Chen, Australia
Valeria De Antonellis, Italy
Asuman Dogac, Turkey
Maria Grazia Fugini, Italy
Harald Gall, Switzerland
Dimitrios Georgakopoulos, USA
Peter Green, Australia
Paul Grefen, The Netherlands
Manfred Hauswirth, Ireland
Marta Indulska, Australia
Gerti Kappel, Austria
Ekkart Kindler, Denmark
Jana Koehler, Switzerland
Akhil Kumar, USA
Heiko Ludwig, USA
Zongwei Luo, Hong Kong
Peri Loucopoulos, UK
Axel Martens, USA
Jan Mendling, Australia
Mirjam Minor, Germany
Michael zur Muehlen, USA
Mike Papazoglou, The Netherlands
Cesare Pautasso, Switzerland
Olivier Perrin, France
Marco Pistore, Italy
Calton Pu, USA
Frank Puhlmann, Germany
Jolita Ralyte, Switzerland
Hajo Reijers, The Netherlands
Wolfgang Reisig, Germany
Stefanie Rinderle, Germany
Domenico Sacca, Italy
Mohand Said-Hacid, France
Shazia Sadiq, Australia
Wasim Sadiq, Australia
Heiko Schuldt, Switzerland
Carine Souveyet, France
Stefan Tai, USA
Farouk Toumani, France
Kunal Verma, USA
Barbara Weber, Austria
Andreas Wombacher, Switzerland

Local Organization Chair

Danilo Ardagna, Politecnico di Milano, Italy

External Reviewers

Sudhir Agarwal
Ali Aït-Bachir
Piergiorgio Bertoli
Sami Bhiri
Devis Bianchini
Ralph Bobrik
Lianne Bodenstaff
Khouloud Boukadi
Gert Brettlecker
Jan Calta
Remco Dijkman
Rik Eshuis
Dirk Fahland
Sergio Flesca
Francesco Folino
Nadine Froehlich
Walid Gaaloul
Christian Gierds
Karthik Gomadam
Genady Grabarnik
Armin Haller
Jon Heales
Thomas Hettel
Anke Hutzschenreuter
Raman Kazhamiakin
Mariana Kessler Bortoluzzi
Jens Kolb
Oliver Kopp
Jochen Kuester
Marcello La Rosa
Gokce Laleci
Steffen Lamparter
Jim Alain Laredo
Tammo van Lessen
Chen Li
Rong Liu
Guo-Hua Liu
Niels Lohmann
Stefan Luckner
Linh Thao Ly
Peter Massuthe
Ana Karla Medeiros
Michele Melchiori
Thorsten Moeller
Ganna Monakova
Hamid Motahari-Nezhad
Michael Mrissa
Kreshnik Musaraj
Dominic Müller
Paul O'Brien
Olivia Oanea
Alper Okcan
Mehmet Olduz
Woosoek Park
Jarungjit Parnjai
Luigi Pontieri
Francesco Pupo
Michael Rabinovich
Jan Recker
Alastair Robb
Terry Rowlands
Ksenia Ryndina
Yacine Sam
Simon Scerri
Marian Scuturici
Samir Sebahi
Dzmitry Shaparau
Zhongnan Shen
Anna Sibirtseva
Ali Anil Sinaci
Giandomenico Spezzano
Christian Stahl
Steve Strauch
Yehia Taher
Michele Trainotti
Fulya Tuncer
Laura Voicu
Konrad Voigt

Hagen Volzer	Daniel Wuttke
Jochem Vonk	Gabriele Zacco
Gabriela Vulcu	Maciej Zaremba
Daniela Weinberg	Zhangbing Zhou
Jan Martijn van der Werf	Christian Zirpins
Karsten Wolf	

Table of Contents

Business Process Management: Today and Tomorrow

Paul Harmon

BPTrends, USA
pharmon@sbcglobal.net

Companies have been striving to improve their business processes for decades, but, in the past few years, the emergence of a variety of new software technologies and the relentless competitive pressures on large companies to outsource and to develop a worldwide presence has taken the interest in business processes to a new level of intensity. In this talk we consider some of the roots of today's interest in business process management (BPM), the growing resources available to those who want to undertake business process change, the emerging BPM systems that seem destined to transform businesses in the next decade, and the implications this transformation will have for those who work in the new generation of process-oriented organizations.

Understanding and Impacting the Practice of Business Process Management

Michael Rosemann

Queensland University of Technology, Australia
m.rosemann@qut.edu.au

This presentation will explore how BPM research can seamlessly combine the academic requirement of rigor with the aim to impact the practice of Business Process Management. After a brief introduction into the research agendas as they are perceived by different BPM communities, two research projects will be discussed that illustrate how empirically-informed quantitative and qualitative research, combined with design science, can lead to outcomes that BPM practitioners are willing to adopt. The first project studies the practice of process modeling using Information Systems theory, and demonstrates how a better understanding of this practice can inform the design of modeling notations and methods. The second project studies the adoption of process management within organizations, and leads to models of how organizations can incrementally transition to greater levels of BPM maturity. The presentation will conclude with recommendations for how the BPM research and practitioner communities can increasingly benefit from each other.

The Future of BPM: Flying with the Eagles or Scratching with the Chickens?

Peter Dadam

Institute of Databases and Information Systems, Ulm University, Germany
peter.dadam@uni-ulm.de

Service-oriented architectures, business process management (BPM) systems, and BPM in general receive a lot of attention these days and the number of articles which describe the benefits and great potential of these technologies has significantly increased. It is something like a second wave after the first (and short) workflow hype in the middle of the 90's. However, the contemporary hype in newspapers and IT magazines does not really reflect reality. In fact, much more companies are still thinking about whether and in which form they shall introduce these technologies rather than concretely performing projects in these fields. And many companies which have started respective projects are still in the phase of designing and implementing (web) services or in evaluating SOA platforms and repositories of different vendors; i.e., they are still not bringing (larger) processes into production. Nevertheless, expectations are very high: Everything will become easier and more flexible, implementation of cross-organizational processes will become business as usual, and process management systems will enable new kinds of process-aware applications which have to be performed manually today. In fact, BPM has a great potential. However, to realize this potential in practice, we have to face much more the challenges of the real world, we have to learn more seriously from how business processes are executed today, and we have to understand how actors deal with exceptional situations. It is not hard to predict what will happen with the current BPM hype if users discover that they cannot do much more with these technologies than with previous ones or, even worse, that they can do less. And no organization will accept to become inflexible. – It is partially up to us, whether BPM will become a big and sustainable success or whether it will share the fate of many other hypes (like Computer Integrated Manufacturing at the end of the 80's). This talk will present real-world examples from different domains to illustrate where we jump too short. It will use the ADEPT project [1,2] to show how stimulating it can be also from a research point of view to face the reality as it is.

References

1. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. J. of Intelligent Information Systems 10(2), 93–129 (1998)
2. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive Process Management with ADEPT2. In: Proc. ICDE 2005, pp. 1113–1114 (2005)

Applying Patterns during Business Process Modeling*

Thomas Gschwind[1], Jana Koehler[1], and Janette Wong[2]

[1] IBM Zurich Research Laboratory,
Switzerland
www.zurich.ibm.com/csc/bit
[2] IBM Software Group, Canada

Abstract. Although the business process community has put a major emphasis on patterns, notably the famous workflow patterns, only limited support for using patterns in today's business process modeling tools can be found. While the basic workflow patterns for control flow are available in almost every business process modeling tool, there is no support for the user in correctly applying these simple patterns leading to many incorrectly modeled business processes. Only limited support for pattern compounds can be found in some tools, there is no active support for selecting patterns that are applicable in some user-determined context, tools do not give feedback to the user if applying a pattern can lead to a modeling error, nor do they trace the sequence of applied patterns during the editing process.

In this paper, we describe an extension of a business process modeling tool with patterns to provide these capabilities. We distinguish three scenarios of pattern application and discuss a set of pattern compounds that are based on the basic workflow patterns for control flow. We present an approach where business users receive help in understanding the context and consequences of applying a pattern.

1 Introduction

There is wide agreement that patterns can accelerate the process of designing a solution and reduce modeling time, while at the same time they enable an organization to more easily adopt best practices [1,2,3]. Patterns enable participants of a community to communicate more effectively, with greater conciseness and less ambiguity. Furthermore, process patterns are considered as an effective means to bridge the Business IT gap. Bridging this gap is more critical than ever because IT advances have escalated the rate of development of new business functions and operations [2].

Despite the common belief in the importance of patterns, only limited support for using patterns in today's business process modeling tools can be found. While the basic workflow patterns for control flow [4] are available in most business process modeling tools and the YAWL system [5] provides all workflow patterns, applying even a basic pattern is under the full responsibility of the user. It is thus not surprising that most modeling errors result from incorrect combinations of the *exclusive choice*, *parallel split*, *simple merge*, and *synchronization* patterns [6].

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

In this paper, we discuss flexible pattern support where users can apply patterns to unstructured process models, they obtain active support in selecting patterns that are applicable in some user-determined context, the tool gives feedback to the user if applying a pattern can lead to a modeling error and it traces the sequence of applied patterns during the editing process. We focus on the basic workflow patterns for control flow, because of their frequent usage during business process modeling and discuss a set of pattern compounds that can be built from them. We present an infrastructure that automates parts of the pattern application process. The infrastructure analyzes the consequences of applying a pattern with respect to the soundness of the resulting process model and enables only those patterns that are correctly applicable in a given context, which we describe using the category of the *process fragments* to which the pattern is applied. Information about the fragment category is obtained from the *process structure tree* that results from parsing the workflow graph underlying the process model [7,8].

We show how patterns can be integrated in a modeling tool such that they enable business users to move away from a drawing tool with drag-and-drop editing capabilities to a true business-level process modeling tool that allows users to arrive at models of higher quality with less effort. Although we only consider control-flow patterns, we see our contribution as an important prerequisite for extending powerful pattern support to more concrete business process patterns that describe best practices, because these patterns will usually contain control-flow information as one essential part of the pattern description [3]. We do not yet introduce a domain-specific vocabulary for the control-flow patterns, but we argue that it is necessary to do so in the future to make the patterns more easily usable by business users.

The paper is organized as follows: In Section 2 we revisit the basic workflow patterns for control flow and define three scenarios for the application of control-flow patterns during an iterative process modeling approach: (1) refinement of a *single* control-flow edge by applying a block-oriented pattern compound, (2) application of a pattern compound to a *pair* of selected control-flow edges, (3) application of a basic pattern to a *set* of selected control-flow edges. Sections 3, 4, and 5 present the three scenarios of pattern application in more detail. Section 3 also provides details on our infrastructure that is based on the process structure tree [7] and that enables us to extend the application of patterns to unstructured process models. Section 6 summarizes initial experiences with an implementation of the three pattern scenarios in a commercial business process modeling tool. The section also discusses the interplay of process patterns, process refactoring operations, and process model transformation. Section 7 gives an overview on the state of the art in business process patterns, while Section 8 concludes the paper.

2 The Workflow Patterns Revisited

When talking about business process patterns, many business process experts refer to the famous workflow patterns [4] that have their origin in comparing the runtime constructs available in existing workflow engines. Figure 1 shows the most widely used subset of the control-flow patterns. We selected these patterns to build active pattern support into a business process modeling tool.

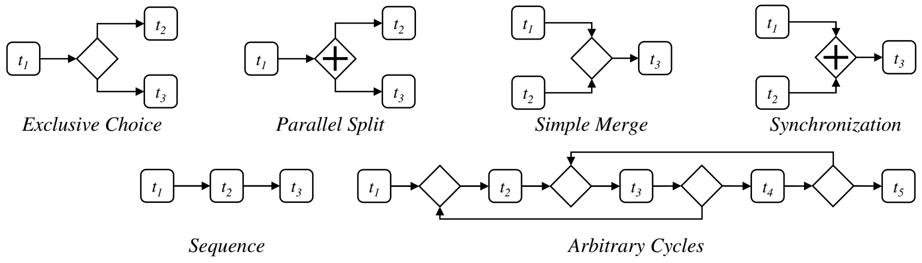


Fig. 1. The five basic workflow patterns *exclusive choice*, *parallel split*, *simple merge*, *synchronization* and *sequence*. In addition, the *arbitrary cycles* pattern as the most frequently occurring pattern for iteration [4].

The patterns as shown in Figure 1 are of course available in most business process modeling tools in the form of gateway icons that business users can drag and drop on a canvas and connect to other modeling elements. Unfortunately, this availability of the patterns in today's modeling tools is insufficient to enable users to successfully reuse proven solutions to recurring problems. The workflow patterns are too fine-grained and not sufficiently enriched with information on the context and consequences to represent a reusable solution. A possible alternative, as for example implemented in the ADEPT2 system [9,10], is to offer block-structured *pattern compounds* and *change patterns* that allow users to model structured workflows by an editing process where processes are sound by construction.

In this paper, we are especially interested in pattern-support for the editing of unstructured process models where the soundness of these models is not guaranteed by construction. We developed a pattern-based modeling prototype by extending the commercial modeling tool IBM WebSphere Business Modeler with *pattern compounds* that we built from the basic control-flow patterns. Our special emphasis is on *pattern sequences*, i.e., how a model unfolds pattern by pattern and how a user creates an unstructured model by applying patterns in an iterative and tool-supported modeling process.

The process models that we consider are generalizations of workflow graphs that permit multiple start and end events. Following [11] we define them as follows:

A business process model is a directed graph $G = (N, E)$ where each node $n \in N$ is either a start or end event, an activity, or a gateway with the gateways partitioned into the types exclusive choice, parallel split, simple merge, and synchronization, satisfying the following conditions

1. there is at least one start event and at least one end event; each start event has no incoming edges and exactly one outgoing edge, whereas each end event has exactly one incoming edge but no outgoing edges,
2. the exclusive choice and parallel split have exactly one incoming edge and two or more outgoing edges, whereas the simple merge and synchronization have two or more incoming edges and exactly one outgoing edge; each activity has exactly one incoming and exactly one outgoing edge,
3. the graph is connected and each node $n \in N$ is on a path from a start to an end event.

We are adopting BPMN notation to draw the process models and pattern structures. This means that we use a diamond to depict a gateway and in the case of a parallel split or synchronization, a plus sign is added to the diamond. Activities are depicted with rounded corner rectangles, while a start event is depicted with an empty circle and an end event is depicted with a thick circle.

Table 1 gives an overview of three *pattern application scenarios* that we discuss in this paper. Each scenario is applicable to process models that are still unfinished, i.e., they may not fully comply to the definition above.

Table 1. Overview of pattern application scenarios

Sc.	**selected process elements**	**applied pattern compound**	**source**	**target**
1	single edge	well-formed sound block	sound	sound
2	pair of edges	gateway-guarded control flow	sound	sound/unsound
3	set of edges	gateway	sound/unsound	sound/unsound

In Scenario 1, a user selects a *single edge* in a model. This edge is replaced by a pattern compound that represents a well-formed process fragment. The user has four choices of pattern compounds that he can apply: *sequence*, *parallel compound*, *alternative compound*, and *cyclic compound*. This form of pattern application (sometimes also denoted as transition refinement) is always possible in our tool. When applied to a sound process model or fragment thereof, it preserves the soundness, i.e., it cannot introduce any modeling errors. Section 3 discusses this scenario in more detail.

By *soundness* of a process model, we mean the absence of *deadlocks* and *lack of synchronization*. In other words, no situation occurs where some part of the process is waiting indefinitely for another part of the process and no part of the process executes more often than intended because of two tokens that occur on the same edge. A formal account of soundness would go beyond the scope of this paper, but can be found in [12,7].

In Scenario 2, a user selects a *pair of edges* in the model to which he can add a new gateway-guarded control flow. Two pattern compounds are available to the user which we denote as *alternative branch* and *parallel branch*. This scenario allows the user to also create arbitrary cycles. The pattern application is always possible, but an unshielded application can introduce new modeling errors, i.e., a process or fragment with a sound underlying workflow graph can become unsound. We describe this scenario in Section 4 and discuss how potential soundness problems can be discovered and prevented.

In Scenario 3, the user selects a *set of edges* to redirect existing control flow such that it starts or ends in a newly introduced gateway. In this scenario, the basic control-flow patterns are directly available to the user. They can be applied to any process model or fragment thereof and either maintain soundness, yield an unsound model or correct an unsound model into a sound one. In Section 5 we describe an infrastructure that alerts the user of these situations and thereby extends the limited support for basic workflow patterns that is available today.

Our scenarios differ by the user-triggered selection of modeling elements and by the class of process models that the user can create with the patterns that are available for each selection. The focus on the selection of modeling elements is important to help business users understand how to apply a pattern. Furthermore, it provides them with a simple and systematic description of the context of a selected pattern in the form of the surrounding process fragment, and the consequences in terms of soundness of the resulting model, while the modeling tool exploits this information to automate the pattern application. We believe that a higher degree of automation is essential because we are addressing non-technical users in contrast to software developers who traditionally apply software patterns in a mostly manual process.

3 Scenario 1: Applying Patterns to a Single Edge

Our first scenario has been widely studied by the workflow community, e.g., as a form of transition refinement [13]. We introduce it here in order to review some essential prerequisites for structured workflow modeling that we then gradually relax in Scenarios 2 and 3. Scenario 1 provides the user with the most simple form of application of a pattern where he can select a single control-flow edge to further refine the business process model. Instead of selecting a single edge, the user can also select a single activity in the process model. In this case, our tool assumes that with this selection, the single outgoing control-flow edge of this activity is selected, i.e., the pattern is applied following the activity in the control flow.[1]

In this scenario, we provide users with *pattern compounds* that represent a well-formed and sound block-structured fragment of a process. These pattern compounds have been studied within the context of structured workflows [14,15,16,17] and are also available in ADEPT2 [9,10]. Four types of block-structured pattern compounds are available to the user:

– *sequence*: a totally ordered set of connected activities,
– *parallel compound*: a parallel split followed by a synchronization that are connected by two or more branches containing one or more activities[2],
– *alternative compound*: an exclusive choice followed by a simple merge,
– *cyclic compound:* a simple merge followed by an exclusive choice.

Figure 2 illustrates this mode of pattern application, which restricts the user to model structured workflows, but which are guaranteed to be sound by construction. The initial sequence of activities in this example can be either created manually or by using the *sequence* pattern. Alternatively, we offer an *auto-link transformation* where the user only places the activities that he wants to be part of the initial sequence in an approximate horizontal arrangement. Then he invokes the auto-link transformation that takes a set of horizontally arranged activities and produces a fully connected sequential process model including a start and an end event.

Aalst [14] and Kiepuszewski et al. [15] showed that only a subset of all workflow graphs can be generated when using block-structured process fragments. However,

[1] We do not consider here the refinement of a single activity into a subprocess, which is a completely different scenario.

[2] The number of branches and the names of activities can be provided as parameters when invoking the pattern.

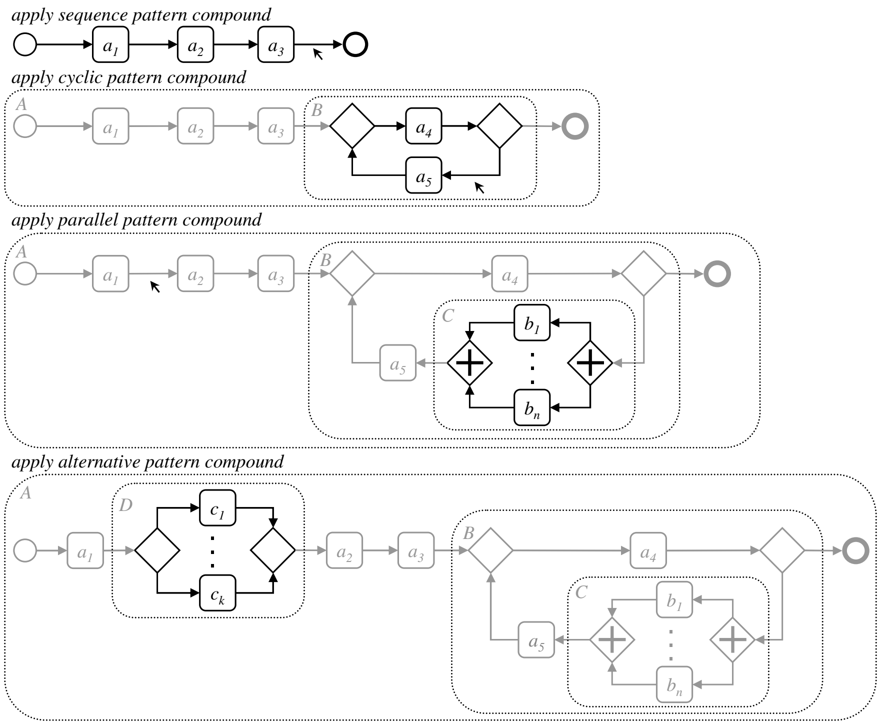

apply sequence pattern compound
a1
a2
a3
apply cyclic pattern compound
A
B
a4
a5
apply parallel pattern compound
C
b1
bn
apply alternative pattern compound
D
c1
ck

Fig. 2. Sound refinement of a process model by applying block-structured pattern compounds to a single edge

modeling block-structured processes is practically relevant for two reasons: First, these models are more comprehensible to human users [18]. Secondly, they can be directly mapped to structured process execution languages such as BPEL and thus make it much easier to go from business to IT.

In order to trace the successive application of patterns, i.e., the pattern sequence, and to determine the context under which a pattern can be correctly applied, we use the *process structure tree* (PST) [7,8] and the notion of *category* of a fragment. In Scenario 1, illustrated by Figure 2, the PST is used to trace the successive application of patterns by the user. Scenarios 2 and 3 will illustrate how the PST together with the category notion can be used to help a user correctly apply patterns to unstructured process models.

The PST results from parsing the workflow graph of the process model. It represents a unique decomposition of a workflow graph into canonical *SESE fragments*, which are either disjoint or fully nested in each other. A SESE fragment is a non-empty subgraph of the workflow graph that is bordered by a single-entry and a single-exit (SESE) edge [7] or node [8]. The dotted rounded-corner rectangles in Figure 2 show the SESE edge fragments of the example. Note that only maximal sequences are canonical fragments, e.g., the sequence containing activity a_1 followed by fragment D is not a canonical fragment, because fragments a_2, a_3, and B are part of the same sequence. The PST can be computed in linear time. It is unique and modular, i.e., a local change of the workflow graph only causes a local change of the decomposition. It is as fine-grained as possible when using SESE node fragments [8].

To determine whether a pattern can be correctly applied, the category of a fragment is important, which is defined by syntactic properties of the underlying workflow graph. Well-structured, acyclic concurrent, unstructured alternative, and complex fragments were proposed in [7]. Other categorizations can be defined instead, i.e., we define and use our own categories *sequence*, *alternative branching* (an arbitrary number of XOR-splits and XOR-joins that must be cycle-free), *parallel branching* (an arbitrary number of AND-splits and AND-joins that must be cycle-free), and *cyclic alternative branching*, which is an alternative branching that is not cycle-free. Fragments in these categories are known to be sound. Figure 3 illustrates the categories *parallel branching* and *cyclic alternative branching* with two unstructured example models.

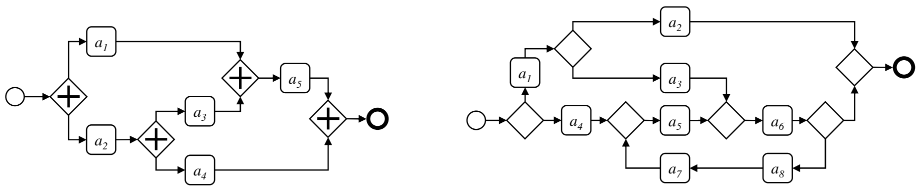


Fig. 3. Fragment categories parallel branching (left) and cyclic alternative branching (right)

Figure 4 shows the PST for the example of Figure 2 based on SESE edge fragments.

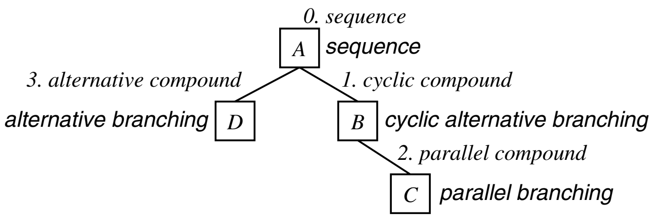


Fig. 4. The process structure tree

The nodes of the PST, which represent the fragments, are annotated with the category of the fragment. The edges are annotated with a number and pattern name providing us with the history of pattern application, i.e., the pattern sequence that the user applied in this example: 0. sequence, 1. cyclic compound, 2. parallel compound, 3. alternative compound. Applying patterns in Scenario 1 adds new fragments to the tree. When applying the patterns of Scenario 2 and 3, fragments and their category can change locally in the PST, e.g., a fragment can also disappear.

4 Scenario 2: Applying Patterns to a Pair of Edges

As refining process models with block-structured patterns is very limiting for many business modeling scenarios, we now consider a first generalization where the user selects an ordered pair of edges (s, t). The first selected edge s is considered the *source* of the new flow and the second selected edge t is considered its *target*. The user can select any two edges as source and target edges.[3]

In this scenario, we support two pattern compounds *alternative branch* and *parallel branch* that the user can apply to establish a new control flow between the source and the target. We provide the pattern compounds in the form of gateway-guarded control-flow edges:

– *alternative branch*: an exclusive choice with a single outgoing edge leading to a simple merge,
– *parallel branch*: a parallel split with a single outgoing edge leading to a synchronization.

Figures 5 and 6 illustrate a typical example of this pattern application scenario. In Figure 5 we see part of a mortgage approval process with two alternative branches. If the customer is not creditworthy, a rejection is sent by the bank and the application by the customer is closed. If the customer is creditworthy, a mortgage offer is sent, the documents are completed, and an account is set up for paying out the mortgage and the mortgage is registered.

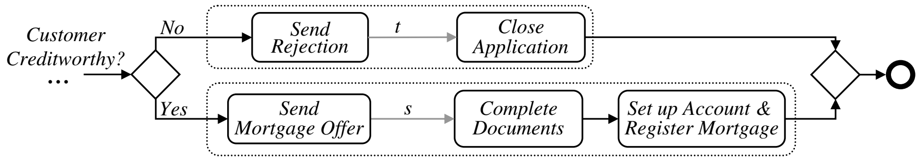


Fig. 5. Example of a simple mortgage approval process

When taking a closer look at this process model, we notice that it assumes that the customer accepts the mortgage offered by the bank. However, this may not always be the case. If the offer is rejected by the customer, the bank employee should contact the customer to find out why and then also close the application. To achieve this change in the process model, the user selects the edges s and t as the source and target and applies the alternative branch pattern. The result can be seen in Figure 6. In a parameterized version of this pattern, a list of activities can be provided that is placed on the newly added branch.

In this example, the user transforms a structured model into an unstructured, but sound model. If the user had applied the parallel branch pattern compound to Figure 5, a deadlock error would have been introduced. In order to prevent such situations, knowledge of the fragment categories maintained within the PST is essential when patterns

[3] Alternatively, a user can select two activities where the tool takes the outgoing edge of the first activity as the source and the incoming edge of the second activity as the target.

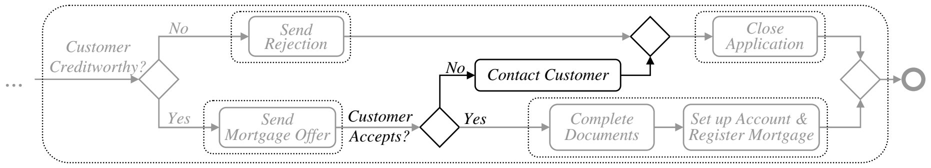


Fig. 6. Adding an alternative branch pattern to two selected edges

are applied to a pair of edges. In this example, the user selects two edges that belong to two different fragments of type sequence that each comprise one of the decision branches for the *Customer Creditworthy?* decision. The pattern application destroys these fragments. They are replaced by four smaller fragments of type sequence—two on each branch. Their parent fragment, which spawns the process fragment of Figure 5, remains unchanged and also preserves its type *acyclic sequential branching*.

The tool guides the user in applying the pattern compounds by analyzing the SESE fragments that contain the selected edges. If the selected pair of edges is a pair of entry/exit edges of a SESE fragment, all pattern compounds that we discussed for Scenarios 1 and 2 are applicable independently of the category of the fragment. If the user selects a pair of edges where at least one of the edges is not an entry or exit edge of a fragment, an analysis of the SESE fragments surrounding the selected edges needs to be performed. The tool analyzes the SESE fragments containing the source and target edge and all those SESE fragments that enclose these fragments up to the smallest SESE fragment that contains both edges. All fragments have to be of the same category, which decides if a parallel or alternative branch can be applied to the edges, see Table 2.

Table 2. Soundness of pattern application based on fragment category

Fragment category	adding cycle allowed?	parallel branch	alternative branch
sequence	yes	×	√
sequence	no	√	×
cyclic alternative branching	yes	×	√
parallel branching	no	√	×

Rows 1 and 2 show that if the fragment categories are a simple sequence of activities, both branch patterns are applicable to a selected pair of edges. However, only the alternative branch pattern is permitted to add a cycle to the process model (row 1). Adding a parallel branch such that a cycle is introduced would lead to a deadlock and is thus not permitted (row 2). The (acyclic) alternative branching is a special case of the cyclic alternative branching and is thus subsumed by row 3. Adding an alternative branch to a parallel branching fragment is not permitted, because it would introduce a lack of synchronization error (row 4).

If the selected edges do not satisfy the conditions with respect to the fragment categories or if the process model is known to be unsound, patterns can still be applied in our current prototype, because we do not want to constrain the user too much in using the pattern-based editing capability. However, a warning, but no further guidance is given to the user. Note that it is not possible to eliminate a deadlock or lack of synchronization error by only applying one of the six pattern compounds that we discussed so far.

5 Scenario 3: Applying Patterns to a Set of Edges

In our last scenario, we consider the most general situation where the user has selected a set of two or more edges or nodes. This means, the selections possible in Scenario 2 can occur here as well, but we consider application scenarios for the basic patterns *parallel split*, *synchronization*, *exclusive choice*, and *simple merge*. When applying the basic patterns to unsound fragments, it is possible for the user to correct modeling errors.

We want to support users selecting nodes in addition to selecting edges because in the midst of editing a process model, it is common to encounter nodes without connecting edges yet and applying patterns to nodes can be very useful to complete the editing. On the other hand, it is not usual to have dangling edges without nodes, because business process modeling tools make users add nodes first and then allow them to connect the nodes with edges. We support three situations if nodes are selected:

1. all selected nodes have incoming edges, but no outgoing edges,
2. all selected nodes have outgoing edges, but no incoming edges,
3. all nodes are fully disconnected, i.e., have neither incoming nor outgoing edges.

In situation (1), a new outgoing edge is added to each node and the synchronization or simple merge pattern is enabled depending on the fragment type returned for the selection. In situation (2), a new incoming edge is added to each node and the parallel split or exclusive choice pattern is enabled. In situation (3), all four basic patterns are enabled and depending on the selection of a pattern by the user, either a new outgoing or incoming edge is added to each node.

Currently, we impose very restrictive constraints when applying the basic patterns to a selection of nodes or edges. For example, a synchronization pattern can be added if a parallel split is found in the process model from which all selected nodes or edges can be reached without encountering other non-AND gateways along the path. Similar conditions can be formulated for the other three basic patterns.

Figure 7 illustrates an example situation. In case that the user only selects activities a_3, a_4, and a_5 in the process model shown in the left, the exclusive choice is found that can only be correctly matched with a simple merge. If in addition, a_1 is selected as well, the parallel split is found, but on three of the four paths, the exclusive choice is encountered. In such a situation, more than one pattern must be applied as is shown in the right of the figure, which requires refactoring techniques that are subject of our ongoing work [19]. The same challenges occur when the user selects a set of edges. Again, we constrain the pattern application as described above. In addition, we have to consider the nodes that are connected by the selected edges.

Figure 8 illustrates an example process where the user wants to introduce two join points in the process flow. With his first selection e_1, e_2, e_3, the user wants to join the three parallel flows to allow the doctor to talk to the patient after having worked out the

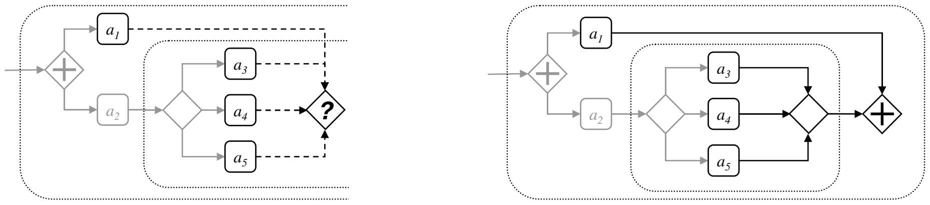


Fig. 7. Applying a single basic pattern to merge or synchronize all selected nodes/edges is not possible without introducing an error into the process model. Instead, two basic patterns must be applied.

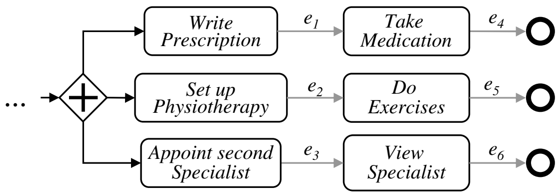


Fig. 8. A medical example process with two edge selections e_1, e_2, e_3 and e_4, e_5, e_6

patient's prescriptions. With his second selection e_4, e_5, e_6, the user wants to join the individual process ends and allow the doctor to review the treatment results with the patient.

Figure 9 shows the result of applying two synchronization patterns to the example, each parameterized with an activity that follows the added AND-Join.

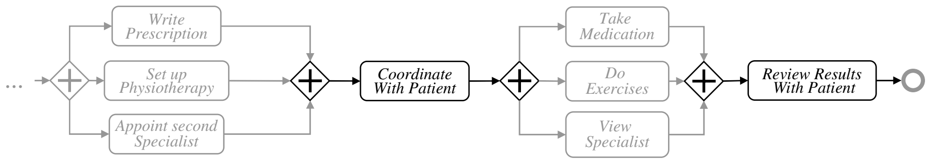


Fig. 9. Selecting a set of edges for joining leads to a fusion of nodes into a single node when they are identical. In the case of different nodes, applying a pattern to join branches also requires applying another pattern to split the branches again, because the nodes cannot be merged.

Applying the synchronization pattern to the first selection e_1, e_2, e_3 requires the introduction of an additional AND-Split following the AND-Join resulting from the pattern, because the subsequent activities cannot be merged. In the case of gateways, a merging is sometimes possible, but it may not always be desired by the user. We take a conservative approach so far and do not merge gateways. When applying the synchronization pattern to the second selection e_4, e_5, e_6, the end events can be merged into a single node and no additional gateway is needed. After having applied the two patterns, the process can be further improved by for example applying the cyclic pattern compound to iterate the prescription and treatment for the patient if necessary.

6 Implementation and Validation of Pattern-Based Editing

Figure 10 shows a screen capture of our prototype implementation where we added pattern support in the form of additional plug-ins to IBM WebSphere Business Modeler, which is an Eclipse-based commercial business process modeling tool. As not all process models can be generated with the patterns and pattern compounds that we described in this paper, we also provide the user with refactoring and transformation operations in addition to the normal editing capabilities.

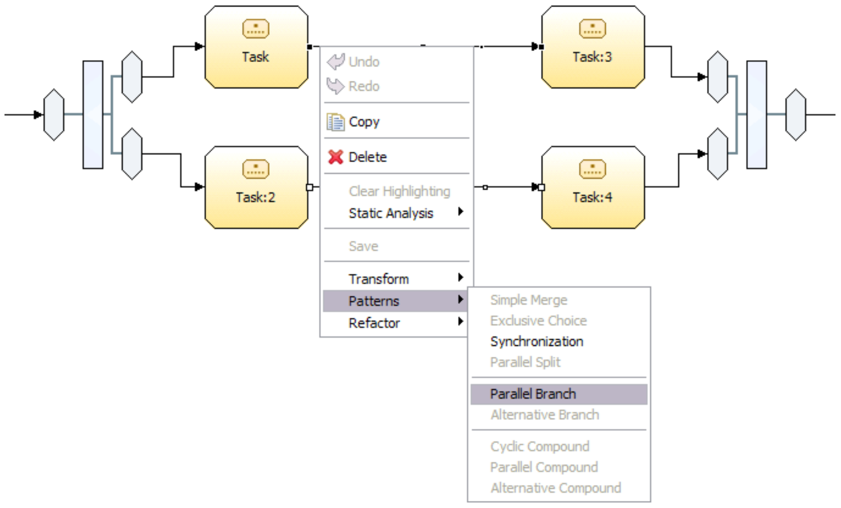


Fig. 10. Context-sensitive pattern availability when selecting a pair of edges within a parallel branching fragment

Our set of currently implemented transformations has been described in [20] and includes a first prototype of a transformation that introduces data flow. We plan to extend this set of transformations by adopting and extending transformations that have been described in [14,16,17,21]. Several of these transformations require a tool to verify the soundness of the process model to which they are applied or that they create. With the linkage of our transformations and patterns to the PST and its fragments, we have laid the foundation to perform these checks much faster. In many practical cases a process model only contains simply structured fragments where soundness can be decided in linear time. For the general case, we currently develop a complete soundness checker that we can can invoke on complex fragments. The feedback to the user about potential modeling errors that can be introduced into a model when applying a pattern is clearly valuable to increase the quality of the process models.

One can easily demonstrate that by applying patterns, transformations, and refactoring operations on a business process model, many time-consuming editing operations can be replaced by a single click [20]. A first adoption of the plug-ins by IBM consultants showed that on average about 10% of the modeling time can be saved with up to 70% of the pure editing time. About 50% of all users who installed the plug-ins use them frequently in their daily work. Approx. 10% find them very easy to use, while two-thirds said that they need practice. 90% of all users confirmed that the plug-ins help them in improving the quality of their models.

Adding pattern-based support for data flow was the most frequently requested extension of the plug-ins. When applied to a single edge, patterns can inherit the data flow from the single edge. If several edges are selected that carry different data items, many possible ways to resolve such a situation exist that we currently explore. Our approach can also be extended to activities that have multiple incoming and outgoing edges, but then requires different disambiguation techniques to determine the edges to which a pattern must be applied in case the user selects one or more activities. As such a disambiguation is not always possible, slightly modified patterns with more constrained application conditions must be developed.

7 Related Work

A growing divide in the patterns world is discussed that opens between the pattern experts who continue to document patterns and the pattern users who are rarely aware of relevant patterns and understand how to leverage and apply them [22]. Only little adoption of patterns by practitioners is observed leading to a rather low impact of the pattern experts on the expected pattern users. The reason for the low adoption of patterns is located in the difficulty to find, contextualize, and compose patterns. "To use a cooking analogy, what they find is a list of ingredients when what they really want is a recipe" [22]. This observation is more than true for the business user working with a business process modeling tool today.

In order to enable users to adopt and actively use patterns, tools must allow users to build applications by progressively applying patterns. However, how and if patterns can be built into tools is a hot debate [23]. Following the pioneering work by Gamma et al. [24], patterns must be thoroughly described by the commonly recurring *problem*, the *context* and *consequences* of applying the pattern, and the *solution* provided by the pattern itself. Understanding the context and consequences related to a specific pattern is a very important human-centered task. Tools that help users in achieving this task must provide active support to select patterns and apply them in composition steps towards creating a complete solution for a particular scenario. The challenge is that "tools that work with patterns would have to be able to semantically understand your design as well as the pattern's trade-offs" [23]. By linking pattern application to the process structure tree, its fragment categories and their soundness, we have built an initial semantic understanding into our business process modeling tool.

Process patterns are found at three levels of abstraction [3]: (1) *abstract process patterns* that capture generic process structures applicable to any process, (2) *general process patterns* that capture proven process elements applicable across different domains, (3) *concrete process patterns* that capture hands-on experience and best practices for recurring business functions in a specific domain. The most prominent example of abstract patterns are probably the famous workflow patterns for control flow [4], which have also been complemented by patterns for data flow and resources. Examples of general process patterns are discussed in [25,26]. A famous collection of concrete process patterns is [27]. Further examples of concrete process patterns are discussed in [28,29,30]. All three levels of process patterns can be built into a business process modeling tool. While it is argued that many abstract patterns can provide significant opportunities for reuse [3]—hence our initial focus on the basic control-flow patterns—it is also emphasized that patterns should be presented in the domain vocabulary of the business user [31] for easier recognition and application, which we have not addressed yet.

Three building blocks of a process pattern-based approach are proposed that must be built into a tool [3]: (1) a *pattern inventory*, (2) support for *pattern selection* and (3) *pattern realization*. Only the ADEPT2 system [9,10], with which we coincide on Scenario 1, seems to implement solutions for the pattern inventory and the selection and realization phases. Some abstract use cases have also been formulated: they comprise the listing, insertion, connection, visualization, and removal of patterns [30] of which we address the context-sensitive listing and the correct insertion of a pattern in a process model in this paper. Palettes that group patterns for specific purposes are discussed in [29,30] and a concrete design of such a palette is shown in [20].

A significant part of research is devoted to pattern languages. An example of such a pattern language for processes implemented in a service-oriented architecture is discussed in [25], while [32,33] describe a visual pattern language for the representation and enforcement of quality constraints in process models. UML-based metamodels and pattern languages are proposed in [34,35,28]. An application of domain-specific modeling languages for the IT-oriented refinement of business processes is discussed in [36]. In this paper, we do not focus on a specific representation of patterns in some language or metamodel, but present an initial collection of patterns for specific scenarios that we found useful for business users.

Unfortunately, very few practical recommendations for the reuse of process patterns are given to business users. Havey [37] emphasizes the need for high quality, but gives only two very simplistic and not so easy to follow recommendations: keep a process model to a size that it fits on a single page (if necessary by using subprocesses) and model in a coarse-grained way, i.e., focus on the main process activities. Our active guidance of the user in applying a pattern helps us to go beyond approaches of syntax-based editing [38] that constrain editing operations by syntactic properties of a model, because we focus on a linkage to a semantic analysis addressing soundness.

8 Conclusion

In this paper, we present three different scenarios of pattern application in a business process modeling tool. For each scenario, we discuss a set of patterns and pattern compounds that are linked to an effective structural and semantic analysis of the business process model based on its process structure tree in order to guide the user in applying a pattern. This analysis helps business users in understanding the context and consequences of applying a pattern and enables the tool to actively support a user during pattern selection and application. Future work will focus on developing a comprehensive set of patterns, refactoring operations and model transformations for the most frequent use cases in business process modeling including a refinement of process models with data flow.

References

1. Buschmann, F., Henney, K., Schmidt, D.: Past, present and future trends in software patterns. IEEE Software 24(7/8), 31–37 (2007)
2. Medicke, J., McDavid, D.: Patterns for business process modeling. Business Integration Journal 1, 32–35 (2004)
3. Tran, H., Coulette, B., Thuy, D.: Broadening the use of process patterns for modeling processes. In: Proc. SEKE, Knowledge Systems Institute Graduate School, pp. 57–62 (2007)
4. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
5. van der Aalst, W., ter Hofstede, A.: Yawl: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
6. Koehler, J., Vanhatalo, J.: Process anti-patterns: How to avoid the common traps of business process modeling. IBM WebSphere Developer Technical Journal 10(2+4) (2007)
7. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models though SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
8. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) Proc. BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008)
9. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with ADEPT2. In: 21st Int. Conference on Data Engineering, pp. 1113–1114. IEEE, Los Alamitos (2005)
10. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
11. Sadiq, W., Orlowska, M.: Analyzing process models using graph reduction techniques. Information Systems 25(2), 117–134 (2000)
12. van der Aalst, W., Hirnschall, A., Verbeek, H.: An alternative way to analyze workflow graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)
13. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)
14. van der Aalst, W.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
15. Kiepuszewski, B., ter Hofstede, A., Bussler, C.: On structured workflow modeling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
16. Sadiq, W.: On business process model transformations. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 267–280. Springer, Heidelberg (2000)
17. Eder, J., Gruber, W., Pichler, H.: Transforming workflow graphs. In: Proc. INTEROP-ESA, pp. 203–216. Springer, Heidelberg (2006)
18. Mendling, J., Reijers, H., Cardoso, J.: What makes process models understandable? In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 48–63. Springer, Heidelberg (2007)

19. Vanhatalo, J., Völzer, J., Moser, S., Leymann, F.: Automatic workflow graph refactoring and completion (submitted for publication)
20. Koehler, J., Gschwind, T., Küster, J., Pautasso, C., Ryndina, K., Vanhatalo, J., Völzer, H.: Combining quality assurance and model transformations in business-driven development. In: Proc. AGTIVE. LNCS, vol. 5088. Springer, Heidelberg (2008)
21. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science 270(1-2), 125–203 (2002)
22. Manolescu, D., Kozaczynski, W., Miller, A., Hogg, J.: The growing divide in the patterns world. IEEE Software 24(4), 61–67 (2007)
23. Kircher, M., Völter, M.: Software patterns. IEEE Software 24(7/8), 28–30 (2007)
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1994)
25. Zdun, U., Dustdar, S.: Model-driven and pattern-based integration of process-driven SOA models. Int. Journal of Business Process Integration and Management 2(2), 109–119 (2007)
26. Barros, O.: Business information system design based on process patterns and frameworks. BPTrends 9, 1–5 (2004)
27. Malone, T., Crowston, K., Herman, G.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge (2003)
28. Tran, H., Coulette, B., Thuy, D.: A UML-based process meta-model integrating a rigorous process patterns definition. In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 429–434. Springer, Heidelberg (2006)
29. Thom, L., Iochpe, C., Reichert, M.: Workflow patterns for business process modeling. In: 8th Workshop on Business Process Modeling, Development, and Support in conjunction with CAISE 2007 (2007)
30. Thom, L., Lau, J., Iochpe, C., Mendling, J.: Extending business process modeling tools with workflow pattern reuse. In: Proc. ICEIS 2006. LNBIP, vol. 3, pp. 447–452. Springer, Heidelberg (2007)
31. Rising, L.: Understanding the power of abstraction in patterns. IEEE Software 24(7/8), 46–51 (2007)
32. Förster, A., Engels, G., Schattkowsky, T., Straeten, R.: A pattern-driven development process for quality standard-conforming business process models. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), pp. 135–142. IEEE, Los Alamitos (2006)
33. Förster, A., Engels, G., Schattkowsky, T., Straeten, R.: Verification of business process quality constraints based on visual process patterns. In: Proc. TASE, pp. 197–208. IEEE, Los Alamitos (2007)
34. Störrle, H.: Describing process patterns with UML. In: Ambriola, V. (ed.) EWSPT 2001. LNCS, vol. 2077, pp. 173–181. Springer, Heidelberg (2001)
35. Hagen, M., Gruhn, V.: Process patterns - a means to describe processes in a flexible way. In: Proc. ProSim (2004), http://prosim.pdx.edu/prosim2004
36. Brahe, S., Bordbar, B.: A pattern-based approach to business process modeling and implementation in web services. In: Georgakopoulos, D., Ritter, N., Benatallah, B., Zirpins, C., Feuerlicht, G., Schoenherr, M., Motahari-Nezhad, H.R. (eds.) ICSOC 2006. LNCS, vol. 4652, pp. 166–177. Springer, Heidelberg (2007)
37. Havey, M.: Essential Business Process Modeling. O'Reilly, Sebastopol (2005)
38. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2. World Scientific, Singapore (1999)

Modularity in Process Models: Review and Effects

H.A. Reijers[1] and J. Mendling[2]

[1] Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.a.reijers@tue.nl
[2] Queensland University of Technology
Level 5, 126 Margaret Street, Brisbane QLD 4000, Australia
j.mendling@qut.edu.au

Abstract. The use of subprocesses in large process models is an important step in modeling practice to handle complexity. While there are several advantages attributed to such a modular design, including ease of reuse, scalability, and enhanced understanding, the lack of precise guidelines turns out to be a major impediment for applying modularity in a systematic way. In this paper we approach this area of research from a critical perspective. Our first contribution is a review of existing approaches to process model modularity. This review shows that aside from some limited insights, a systematic and grounded approach to finding the optimal modularization of a process model is missing. Therefore, we turned to modular process models from practice to study their merits. In particular, we set up an experiment involving professional process modelers and tested the effect of modularization on understanding. Our second contribution, stemming from this experiment, is that modularity appears to pay off. We discuss some of the limitations of our research and implications for future design-oriented approaches.

1 Introduction

Modularity is the design principle of having a complex system composed from smaller subsystems that can be managed independently yet function together as a whole [19]. Such subsystems – or *modules* – can be decomposed in a similar vein. In many domains, modularity is a key principle to deal with the design and production of increasingly complex technology. For example, it has been argued that the computer industry has dramatically increased its rate of innovation by adopting modular design [5]. Modules can also be found in business process models, where they are commonly referred to as *subprocesses*. Most popular process modeling techniques support this concept, e.g. flowcharts, IDEF0 diagrams, UML Activity Diagrams, EPCs, BPMN, and YAWL.

Various advantages are attributed to the use of subprocesses in process models. At build-time, subprocesses support a *modeling style* of stepwise task refinement, stimulate *reuse* of process models, and potentially *speed up* the (concurrent) development of the overall process model [2,23]. At run-time, when a process model is enacted, subprocesses allow for *scaling* advantages: Each subprocess, for example, may be executed on a different workflow server [23]. Finally, when a process model is used to facilitate the understanding of complex business processes among various stakeholders, subprocesses are supposed to ease the *understanding* of the model [15,36].

However, the way that modularity is currently utilized in modeling practice raises some questions about the actual benefits. First of all, there are no objective criteria for applying granularity. Accordingly, there is no absolute guideline if a particular subprocess should be on level X or $X + 1$ in the model hierarchy [13]. Neither is there a unique way to modularize a process model [13]. As a consequence, modularity is often introduced in an ad-hoc fashion. Furthermore, there are clearly drawbacks when the process logic is fragmented across models. In particular, it "becomes confusing, less visible, and tracking its paths is tiring" [12] if a subprocess is decomposed in further subprocesses. The fact that the semantic check in `ARIS Toolset` mainly addresses consistency issues between events in the subprocess and around the refined function illustrates the seriousness of this problem. Finally, even if modularization is useful for maintenance purposes, it is questionable whether advantages materialize in practice as many organizations fail to keep their models up to date.

The greater research challenge we see here is to provide explicit guidance for using modularization in process models. But, this would be a dubious undertaking at the present state of the art: We simply do not have the evidence whether modules in process models pay off. Therefore, this paper is concerned with establishing an empirical foundation as a necessary preparation for a design-oriented approach to the subject. We start this investigation from a critical review of existing approaches to introduce modularity in process models.

Our null hypothesis is that *modularization does not increase process model understanding*, and we introduce an experimental design to challenge it. In this approach, we worked together with a group of professional process modelers to evaluate a set of professional process models. The controlled variable in the design is whether subprocesses are used or not; the response variable is the degree of understanding that the subjects display with respect to the models. Note that we focus on the *understanding* of a process model as the major point of evaluation. Our motivation is that in most business applications, the primary purpose of a process model is to act as a means of communication [25,31]. As Van der Aalst and Van Hee put it when discussing the introduction of subprocesses in a process model "[..] the most critical consideration is that the process be understood by the people carrying out the work. If this is not the case, the result can be a difficult-to-manage process."

Against this background, the structure of this paper is as follows. In the next section, we will give a broader background for the concept of modularity, in particular with respect to process modeling. In Section 3, we will present our research method, after which the results of the experiment we carried out are given in Section 4. Before we come to a conclusion in Section 6, we discuss our findings and their limitations in Section 5.

2 A Review of Modularity and Process Modeling

2.1 Concepts and Terms

A first issue that should be considered here is that the terms *modularity*, *decomposability*, and *hierarchy* are sometimes used interchangeably. However, according to [19], a modular system is not automatically *decomposable*, since one can break a system into modules whose workings remain highly interdependent with the internal workings of other modules. Furthermore, as Parnas points out in his seminal paper on "information hiding", a modular system is not necessarily *hierarchical* [32]. That would be the case if the "uses" relation between modules gives a partial ordering, which is not always so. One can easily imagine, for example, a software program where software modules mutually call each other. These subtleties also hold in the context of process models. In most practical cases, however, a modular process model will probably be hierarchical too although perhaps not decomposable, i.e. its subprocesses may still be highly interdependent. In this paper we consider the more general phenomenon of "modularity" as the main point of interest.

2.2 Modularization in Systems

In many settings, "the real issue is normally not to be modular but *how* to be modular" [19]. But at the same time, modular systems are much more difficult to design than comparable interconnected systems [5]. Beyond that, problems with incomplete or imperfect modularization tend to appear only when the modules come together and work poorly as an integrated whole. It has been argued that many of the most attractive and durable systems are developed through an "unselfconscious" design process [4]. In this mode, the design rules that are used are not explicit; inconsistencies and interdependencies are revealed by trial and error only. However, it is by no means obvious that unselfconscious design must always, or even usually, result in modularity [19].

Quality criteria to consciously decompose a system into modules have been discussed by Wand and Weber on a general level [41,43]. The authors identify five criteria. The first three are absolute criteria that are either met or not and focus on the content of the modular model, not its structure. *Minimality* requires that there is no redundant state information in the modular model. In data models this basically matches normalization requirements. *Determinism* requires that a state change is clearly identified to be triggered by an internal or an external event. If that is not the case the behavior of a module can only be understood by knowing the state of another subsystem. *Losslessness* demands that emergent properties are not lost in a modularization. Furthermore, the two criteria coupling and cohesion should be optimized, cf. [45]. *Coupling* should be minimal such that the sum of inputs of each subsystem is less or equal to the sum of inputs in any other modularization. *Cohesion* should be maximal such that all output affected by input variables are contained in the same set, and adding another output does not extend the set of input variables on which they depend.

Wand and Weber's criteria had a strong influence on the object-oriented design metrics proposed by Chidamber and Kemerer [10]. The usefulness of the five criteria has been demonstrated for UML class diagrams and state charts in an experimental setting [8]. Yet, an application in the area of process modeling, either by designing good decomposition operations or by testing their suitability, is missing.

2.3 Modularization in Process Models

The area of related research in the context of process models is huge, covering works on *process modularization*, e.g. [3,7,42], *process inheritance*, e.g. [6,26], and *reduction rules*, e.g. [14,34,44]. Since the latter two categories are mainly utilized in process model analysis, we will focus on the first category. Furthermore, we do not consider modular design of process-aware information systems such as in [16,24]. In the context of process model modularization, three aspects can be distinguished: modularization operations, modularization prerequisites, and modularization selection.

Modularization Operations: The idea that *basic operators* should facilitate modularization was already proposed in the 1980s for data flow diagrams [3]. Refinement operations have also been defined for Workflow Nets [1]. Also, some modeling approaches impose the use of block structures of nested control primitives, which favor the creation of decomposable modules, as in e.g. BPEL. Recently, the ability to extract a subprocess from a process model has been described as a change pattern for process-aware information systems [42]. This pattern must be implemented reflecting the syntactic requirements of the modeling language. In ARIS there are two ways to extract a subprocess: by modularization (refining function with subprocess) and by segmentation (cutting a model in different parts) [13]. Both these options are tailored to yield syntactically correct EPCs.

Modularization Prerequisites: There are some recommendations regarding *when* a process model should be considered for modularization. Some of the practitioners books state that modularization should be introduced in a model with more than 5–15 [18] or 5–7 activities [36], yet without giving any support for this rule. Recently, it has been recommended based on empirical findings that process models with more than 50 elements should be decomposed [28]. Depending on the process modeling language the amount of activities can vary for 50 elements, e.g. EPCs use connectors for routing and events to separate functions while YAWL essentially only uses tasks. Still, up to now no objective criteria has been proposed for identifying which subprocess should be on which level in the model hierarchy [13].

Modularization Selection: There are some guidelines on how to *select* parts of process models for modularization. Good candidates for subprocesses are fragments of a model that are components with a single input and a single output control flow arc [22,7,39]. Furthermore, long and thin process models should be preferred to square models [13, p.278]. This argument points to the potential of metrics to guide the modularization. The idea here would be to use quality metrics like the ones proposed in [28,29] to assess which modularization should be preferred. An application of metrics to compare design alternatives is reported in [38]. Yet, there is no dedicated approach to guide modularization based on metrics.

Overall, the main focus of research on process modularization is of a conceptual nature. Clearly, there are no objective and explicit guidelines that modelers in practice can rely on. The aim of our research as reported in the following sections is to contribute to a better understanding of the effects of modularization as a stepping stone towards such guidelines.

3 Research Design

In the previous sections we discussed that the ad-hoc way in which modularity is currently introduced in modeling practice raises doubts about its benefits. In this section, we will explain our design to test the following null hypothesis:

`H0`: Use of modularization does not improve understanding of a process model.

There are several challenges in testing the presumed absence of a relation between modularity and understanding, in particular in pursuing results that have a potential to be generalizable on the one hand while applying methodological rigor on the other. In particular, it would be unsatisfactory to rigorously test the effects of modularity in small, toy-like process models, as any effect would possibly be hard to spot anyway. To achieve a realistic setting for our research, we set up a collaboration with Pallas Athena Solutions[1] in the Netherlands, a specialized provider of BPM services. This company provided us with real-life models as study objects. Furthermore, their process modelers participated in our investigation. As will be explained in this section, we applied an *experimental* design to achieve sufficient control over the dependent variable (modularization) and to allow a meaningful evaluation of the response variable (understanding) from our hypothesis.

In lack of specific literature on empirical research with respect to modular process modeling, we build on approaches and classifications used in the field of software experimentation [17,33]. In particular, we use an experimental design that is comparable to what was applied in a recent study to evaluate various types of BPM technology [30]. To test the hypothesis we carried out a so-called *single factor experiment*. In general, this design is suitable to investigate the effects of one *factor* on a common *response variable*. This design also allows to analyze variations of a factor: The *factor levels*. The *response variable* is determined when the participants of the experiment – the *subjects* – apply the factor or factor levels to a particular *object*. The overall approach in our experiment is visualized in Figure 1. We will address the most important elements in our design in more detail now.

Objects. The basic *objects* that were evaluated by the participants, were two process models from practice. The models were used in the experiment both in

[1] See http://www.pallas-athena.com

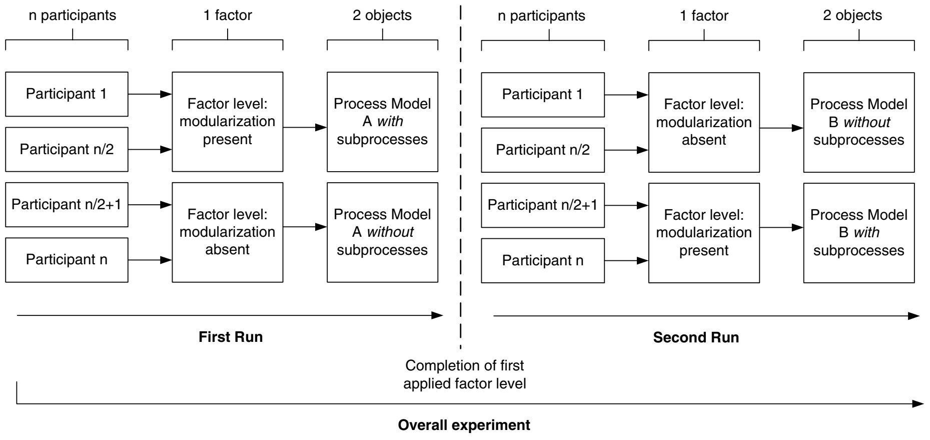

n participants
1 factor
2 objects
Participant 1
Participant n/2
Participant n/2+1
Participant n
Factor level: modularization present
Factor level: modularization absent
Process Model A *with* subprocesses
Process Model A *without* subprocesses
First Run
n participants
1 factor
2 objects
Participant 1
Participant n/2
Participant n/2+1
Participant n
Factor level: modularization absent
Factor level: modularization present
Process Model B *without* subprocesses
Process Model B *with* subprocesses
Second Run
Completion of first applied factor level
Overall experiment

Fig. 1. Experiment design

their original form – displaying modularity – and in their flattened version where modularity is completely removed: All dependencies between model elements are then on the same level of abstraction. Note that for any particular process model the absence or presence of modularity does not affect the business logic.

Both process models were selected from a little over 80 process models that were created and delivered by the consultancy company for its clients. We focused our search for suitable objects using three criteria: (1) presence of modularity, (2) size, and (3) access to the original creators of the models. The process models we looked for needed to display modularity, as consciously applied by the modeler to deal with the complexity of a large model. We only considered models of more than 100 tasks, which can be considered as *very large* using the process size classification provided in [9]. Our line of reasoning here is that if modularity does not help to understand very large models, it will not help to distinctively understand smaller models either. Finally, we needed access to the modelers of the model to validate questions on the content of the model.

From our search, four candidate models emerged. One of these models was specifically developed for automated enactment. It was not further considered because understanding is generally not a prime issue with this modeling purpose. Of the remaining three, which were all developed for the support of stakeholders in a process improvement project, the two process models were selected that were most similar to each other in terms of process size, number of subprocesses, and modularity depth. Both models had been modeled with the `Protos` tool [40]. The flattened versions of the process models can be seen[2] in Figure 2, so that the reader can get an impression about their structure and size. Note that we are not allowed to disclose the content of the models.

Model A describes the procedure in use by an organization that is responsible for handing out driver's licences. The process in question deals with clients that

[2] For larger images, see http://www.reijers.com/models.pdf

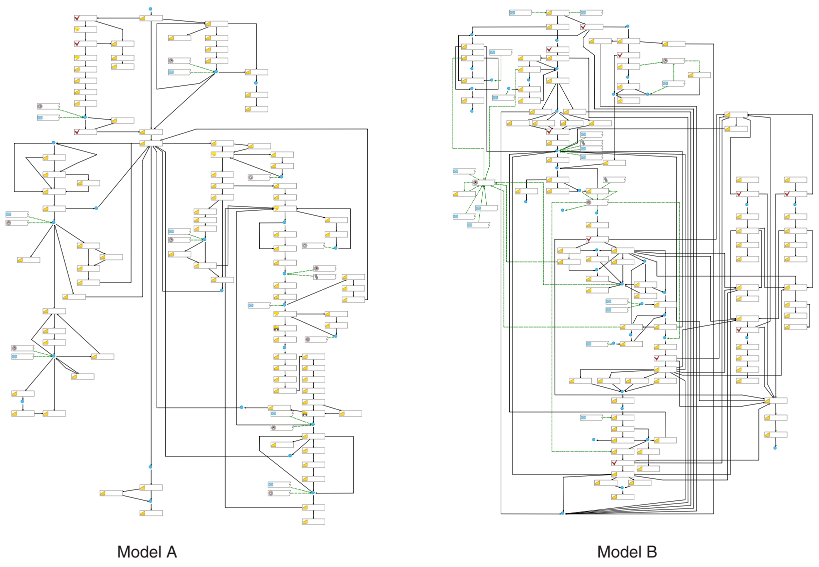


Fig. 2. Flattened versions of the used process models

cannot directly obtain their driver's license because of physical or psychological disabilities that can influence their driving. Model B captures how a certain category of unemployed citizens is coached and receives advice in finding a job. Note that labels in Figure 2 have been removed to protect the confidentiality of the involved organizations; the subjects in our experiment saw the entire model in full (including the labels).

Factor and factor levels. In our experiment, the *use of modularity* is the considered *factor*, with factor levels "present" and "absent". Note that we deliberately collected real process models from practice already exhibiting modularity and derived flattened versions from it, instead of doing it the other way around. In this way, we could build on a real-life application of modularity.

Response variable. The *response variable* in our experiment is the level of understanding that the respondents display with respect to the process models, both in their modularized and flattened form. To measure the response variable, a specific set of questions was developed for each of the two models to be answered by the subjects. We used the percentage of correctly answered questions given by a subject as measure for his or her level of understanding of the particular model. This approach is similar to the one we applied in a previous study into model understandability [29]. An example question for model A is: "If an AA-investigation is required, then a number of alternative settlements is possible. How many of these settlements exist?". For model B an example question is: "If a client does not appear on an appointment, is it always so that a new appointment is scheduled?". Note that the question sets are different for each of the models because both their content and structure differs. The questions were formulated in Dutch, the same language used by the creator of the modeler to name model elements, and also being the native language for all subjects. The model-specific questions were preceded by a general introduction to the experiment, some specific background information for each of the models, and a number of general questions with respect to the subject's background. As will be explained later, we used the latter information for comparison purposes (see Section 4.2).

Subjects. The participants in this experiment were 28 experienced consultants from Pallas Athena Solutions. They were randomly assigned to the two groups used in our set-up (*block design*). Each group was presented two models: One model that displayed modularity and the other model in the flattened version. This way each participant received two *different processes* – models A and B – and two *different styles* – modular and flattened. Participation in the experiment was voluntary; the single reward offered for participation was access to the research results.

Instrumentation. The experiment was then carried out in the following way. The groups of subjects were provided with the process models *on paper*, together with the questions; an alternative would have been to show the models on a computer display, e.g. using the software that was used to create the models. The involved consultancy company indicated that paper is a common form to interact with their clients. Recall that the original versions of the models were divided into subprocesses by their respective authors. These models could therefore be presented to the respondents as a set of A4-sized papers, one for the main process and one for each subprocess. The alternative, flattened model versions were presented on A3 paper format, making task labels clearly legible.

Prior to the actual experimentation, all questions and correct answers were discussed with the creators of the models. They approved of these and validated that the question sets were a proper way to test understanding of the models. Then, five graduate students from Eindhoven University of Technology were involved in a pre-test. This led to the reformulation of 10 questions to remove ambiguities and the removal of 3 questions. The latter was explicitly required to keep the experiment within a reasonable time frame. For each model, 12 questions were included in the final version of the experiment.

Data collection and analysis. During the experiment, the subjects were asked to spend at most 25 minutes per model for answering its related questions. This limit was imposed to keep the time spent on the entire questionnaire under one hour and to prevent an imbalance in time spent on each model. Both at the start and at the end of answering a set of questions for each model, subjects were asked to write down the current time to allow for exact comparisons.

For our data analysis, well-established statistical methods and standard metrics are applied, as provided by the software package `STATGRAPHICS XV.II`.

From the description the elements in this section, it follows that the experiment is *balanced*, which means that all factor levels are used by all participants of the experiment. In general, such an approach enables repeated measurements and the collection of more precise data as every subject generates data for every treated factor level. As can be seen in Figure 1, we went through two runs, so this experiment displays a *repeated measurement*. But in contrast to the approach in [30], two objects instead of one were used (process models A and B) to repeat the experiment in a second run. This setup prevents confronting the same group of subjects to the same model more than once. In this way, we could avoid learning effects to take place while still varying the factor levels.

4 Results

In this section, we will first present our main analysis results, after which we will explore some alternative explanations for these to decide on our hypothesis.

4.1 Main Results

Our main analysis for each model focuses on the comparison between the group performance in terms of correctly answered questions for its modularized and flattened version. In other words, does it matter whether someone sees a modularized or a flattened version of a process model? As explained, we calculated for each of the subjects the percentage of correct answers given for each model to make this comparison. Recall that each subject saw a modular model for one process and a flattened model for the other. The values are shown in Table 1.

Table 1. Average percentages of correct answers for the model variants

	Flattened	**Modular**
Model A	38.54%	42.36%
Model B	37.50%	58.33%

As can be seen from this table, for both models the modular version generates a *higher* average percentage of correct answers, which suggests a better understandability. To determine whether the differences are statistically significant, it is important to select the proper statistical test. Therefore, we first explored for each of the models the distribution of correct answers for each of its variants, i.e. the modular and flattened version. Because the standardized skewness and standardized kurtosis are within the range of -2 to +2, for each model the correctly answered questions can be assumed to be normally distributed. Additionally, F-tests indicated that with a 95% confidence the standard deviations of the samples for each of the models are also the same. These two conditions justify the application of Student's t-test [37].

Application of the t-test results in a P-value for each comparison; a P-value lower than 0.05 signals a significant difference when assuming a 95% confidence level. The results are then as follows:

Table 2. Group comparison

Factor	Factor levels	P-value
Domain knowledge	Knowledgeable with the process context or not	0.386
Company experience	Actual number of years within company	0.411
Field experience	Actual number of years working as process consultant	0.726
Education	University degree or not	0.453
Job type	Business consultant or technical consultant	1.000
Modeling amount	Estimated number of process models created	0.504
Modeling size	Estimated average size of process models created (nodes)	0.764
Time overall	Actual time spent on entire experiment	0.948
Time A	Actual time spent on model A in the experiment	0.641
Time B	Actual time spent on model B in the experiment	0.417

– For model A, there is *no difference* between the modular and the flattened version in terms of the average percentage of correctly answered questions (P= 0.562).
– For model B, there is a *significant difference* between the modular and the flattened version in terms of the average percentage of correctly answered questions (P= 0.001).

The difference for model B seems to support rejection of `H0`. However, we must first explore whether alternative explanations exist to properly decide on the acceptance or rejection of this hypothesis.

4.2 Supporting Results

The main *alternative* explanation for the difference for model B is that the group that produced better results for the modular version is simply different from the group that looked at the flattened version. Recall from Section 3 that our experiment is characterized by a *block design*, i.e. subjects are randomly assigned to the two experimental groups. If the groups are different with respect to a characteristic that may influence their ability to understand process models, then this would not allow us to reject `H0` – despite the noted statistical difference. A second, *alternative* explanation would be that one group of respondents simply spent more time than the other on answering the corresponding questions.

To assess these alternative explanations, we analyzed the characteristics as shown in Table 2. Each entry in the table lists an investigated factor, the considered factor levels, and the P-value resulting from a statistical test. Note that we applied a standard t-test to determine a statistical difference between the groups with respect to each factor, unless its basic requirements were not met with respect to the assumed normal distribution and variance equality. In the latter case, we used the non-parametric Mann-Whitney W test to compare the medians across both groups [37].

All P-values in this table are far greater than 0.05, so none of the investigated factors signals anything close to a statistical difference between the groups at a 95% confidence level. Therefore, in lack of knowledge on other plausible influences, we must *reject* hypothesis `H0`. We conclude that *modularity appears to have a positive connection with process understanding.*

5 Discussion

We single out two questions that emerge from considering the results from the previous section:

1. Why does modularity matter for understanding model B, but not for A?
2. What is the explanation for modularity influencing the understanding of model B?

In this section, we will first address these questions and then discuss some limitations of our experiment.

5.1 Model Differences

We recall that we selected models A and B from a wide range of models, keen on satisfying a number of requirements (see Section 4). From the four models that met these, models A and B were *most* similar, notably with respect to the number of tasks they contain and their depth. To determine why modularity plays a bigger role in understanding model B, we carried out a further analysis of both models by using the metrics shown in Table 3. At the top of the table, some basic metrics are given, followed by metrics that have been proposed as indicators for process model complexity in general, and at the bottom some metrics that are explicitly proposed for assessing modular process models.

Two metrics display values that differ more than a factor 2 between the models under consideration, i.e. `Subprocesses` and `FanIn-Out`. According to [20], the relatively high value of the latter metric for model B (33.42) would suggest a poorer structuring of model B compared to model A, which would make it more difficult to use. However, an additional test to determine whether a difference exists in model understandability between the modular version of model A and the modular version of model B does not show a higher average percentage of correct answers for the former. In lack of other empirical support for the use of this metric, the relatively high number of subprocesses (20) in model B seems more relevant: It suggests that the difference between the modular and flattened version of this model is more distinct than for model A.

For the remaining factors, models A and B display quite similar characteristics, even though model B is the slightly larger one. There is no general trend that suggests that one model is considerably more complex than the other and none of the metrics display substantial and meaningful differences other than the number of subprocesses. So, the most reasonable answer to the question why modularity has an impact on understanding model B but not on model A is

Table 3. Complexity metrics

Metric	Description	Source	Model A	Model B
Tasks	Total number of tasks	–	105	120
Nodes	Total number of nodes	–	130	175
Arcs	Total number of arcs	–	171	248
Subproc	Total number of subprocess in original model	–	**9**	**20**
To	Average number of outgoing arcs from transitions (tasks)	[21]	0.81	1.03
Po	Average number of outgoing arcs from places (milestones)	[21]	3.42	2.24
CycN	McCabe's cyclomatic number (adjusted for Petri nets)	[21]	43	75
Connect	Number of arcs divided by the number of nodes	[27]	1.32	1.42
Density	Number of arcs divided by the maximal number of arcs	[27]	0.020	0.016
ConDeg	Average number of input and output arcs per routing element	[27]	1.10	1.21
Fan-In	Average number of modules calling a module	[20]	1.25	2.26
Fan-Out	Average number of modules called by a module	[20]	1.5	2.26
Fanin-Out	Average $((\text{Fan-In}) * (\text{Fan-Out}))^2$ per module	[20]	**3.63**	**33.42**
Depth	Degree of nesting within the process model	[27]	3	3

that B's original version displayed *a much higher degree of modularization* than model A, which eased its understanding.

5.2 The Influence of Modularity

In search for an explanation of *how* modularity increases model understanding, we re-examined the questions we used in our experiment. Recall that these questions were validated by the original creators of the model (see Section 3): The questions were considered to be to the point, reasonable, and a good way to test someone's understanding of the model.

In the *ex post* analysis of our results, we pursued the idea that by using a modular model perhaps one type of question would be answered better than another. In particular, we categorized our questions as being of a *local* or *global* type. The answer for a local question can be found within the confinements of a single subprocess in the modular version, where the examination of more subprocesses is required to answer a global question. As it turned out, model B contained 2 global questions and 10 local questions. In a comparison between the group that used the modular model and the group that used the flattened model, the following results emerged:

– Too few global questions were used to determine whether there is a difference in terms of the average percentage of correctly answered questions between using the modular or the flattened version of model B .
– For local questions, there is a *significant difference* in terms of the average percentage of correctly answered questions between the modular and the flattened version of model B (P=0.002).

From this analysis, we cautiously infer that modularity may be helpful for understanding a process model because it *shields the reader from unnecessary information.* Where the reader of flattened model always sees the entire context, the reader of the modular version is confronted with precisely the right set of information when the proper subprocess is selected. In this sense, it resembles the positive effect of Parnas' "information hiding" concept [32]: Programmers are most effective if shielded from, rather than exposed to the details of construction of system parts other than their own.

Whether there is also an opposite effect, i.e. the correct answer for a global question would be easier to find with a flattened model, could not be established for model B. However, it does not seem too likely; an analysis of the results for model A did not show such an effect.

5.3 Limitations

Only a small number of 28 subjects were involved in this experiment and only 2 process models were considered. Both aspects are threats to the *internal validity* of this experiment, i.e. whether our claims about the measurements are correct. But these small numbers result from our choices to (1) involve experienced process modelers and (2) process models from industrial practice. Most experienced modelers from the company already participated and the confidential models could not be shown outside the company. Also, to keep the experiment's duration under one hour – a pragmatic upper bound to avoid non-response – it was not feasible to use, for example, more models. The choice for professional modelers and real models clearly positively affects the *external validity* of our study, i.e. the potential to generalize our findings. Therefore, our experiment shows how "internal and external validity can be negatively related" [11].

Another aspect is the choice for displaying the process models *on paper*. It is by no means certain that similar findings would result from an experiment where models are shown on a *computer display*. In the latter mode, "information hiding" is achievable in other ways than by applying modularity. For example, the `Protos` tool that was used to create the models allows to zoom in on part of a process model, which is another form of shielding away irrelevant data.

Finally, the lay-out of a process model may be a factor that influences understandability, as we hypothesized before in [29]. As a limited understanding of this effect exists at this point, we are restrained in properly controlling this variable. We used the same modeling elements, the same top-down modeling direction, and roughly a similar breadth and width for both models on paper to limit this effect – if any (see Figure 2).

6 Conclusion

On the basis of the controlled experiment we described in this paper, the main conclusion of this paper must be that modularity in a process model (through use of subprocesses) appears to have a positive connection with its understandability. However, this effect manifests itself in *large* models if modularity is applied to a sufficiently *high* extent and particularly seems to support comprehension that requires insight into *local* parts of the model.

These results should be considered within the limitations of the experiment we described, but in principle favor further efforts into the development of more explicit design guidance towards modularizing process models. As we noted, this is a major gap in our knowledge on process modeling. From the review of process modularization approaches that we presented in the paper, we identified several attractive ingredients for such an approach. In particular, Wand and Weber's quality criteria have already been succesfully applied for other types of models and the use of metrics to guide process modularization seems a fruitful direction. Our future work is aimed at the development of such guidance and metrics.

Aside from this research agenda, we hope that publications like [8,30,35], and this paper as well, may serve as an inspiration for further integrating methodologies from behavorial science in the design-science approaches common to the BPM field. This could be particularly helpful to provide explicit support for both the necessity and the utility of the models, algorithms, systems, and other artifacts that BPM scholars are concerned with.

Acknowledgements

We would like to thank Bela Mutschler, Barbara Weber, Maarten Boote, and all participants from Pallas Athena Solutions for their contribution to this research.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
2. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT press, Cambridge (2002)
3. Adler, M.: An algebra for data flow diagram process decomposition. IEEE Transactions on Software Engineering 14(2), 169–183 (1988)
4. Alexander, C.: Notes on the Synthesis of Form. Harvard University Press (1970)
5. Baldwin, C.Y., Clark, K.B.: Managing Modularity. Harvard Business Review 75(5), 84–93 (1997)

6. Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)
7. Basu, A., Blanning, R.W.: Synthesis and Decomposition of Processes in Organizations. Information Systems Research 14(4), 337–355 (2003)
8. Burton-Jones, A., Meso, P.: How good are these UML diagrams? An empirical test of the Wand and Weber good decomposition model. In: Applegate, L., Galliers, R., DeGross, J.I. (eds.) Proceedings of ICIS, pp. 101–114 (2002)
9. Cardoso, J.: Poseidon: A Framework to Assist Web Process Design Based on Business Cases. Int. Journal of Cooperative Information Systems 15(1), 23–55 (2006)
10. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)
11. Cook, T.D., Shadish, W.R., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin (2002)
12. Damij, N.: Business Process Modelling Using Diagrammatic and Tabular Techniques. Business Process Management Journal 13(1), 70–90 (2007)
13. Davis, R.: Business Process Modelling With Aris: A Practical Guide (2001)
14. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
15. Dong, M., Chen, F.F.: Petri Net-Based Workflow Modelling and Analysis of the Integrated Manufacturing Business Processes. The International Journal of Advanced Manufacturing Technology 26(9), 1163–1172 (2005)
16. Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. In: Proceedings of the International Working Conference on Dynamic Modelling and Information Systems (1994)
17. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Dordrecht (2001)
18. Kock Jr., N.F.: Product Flow, Breadth and Complexity of Business Processes: An Empirical Study of 15 Business Processes in Three Organizations. Business Process Re-engineering & Management Journal 2(2), 8–22 (1996)
19. Langlois, R.N.: Modularity in Technology and Organization. Journal of Economic Behavior and Organization 49(1), 19–37 (2002)
20. Laue, R., Gruhn, V.: Complexity metrics for business process models. In: Abramowicz, W., Mayr, H.C. (eds.) Proceedings of BIS 2006. Lecture Notes in Informatics, vol. 85, pp. 1–12 (2006)
21. Lee, G.S., Yoon, J.M.: An Empirical Study on Complexity Metrics of Petri Nets. Microelectronics and reliability 32(9), 1215–1221 (1992)
22. Leymann, F.: Workflows Make Objects Really Useful. EMISA Forum 6(1), 90–99 (1996), http://sunsite.informatik.rwth-aachen.de/Societies/GI-EMISA/forum/content_96_1/Emisa_1_96_S90-99.pdf
23. Leymann, F., Roller, D.: Workflow-based Applications. IBM Systems Journal 36(1), 102–123 (1997)
24. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. Prentice Hall, Englewood Cliffs (2000)
25. Lindsay, A., Downs, D., Lunn, K.: Business Processes: Attempts to Find a Definition. Information and Software Technology 45(15), 1015–1019 (2003)
26. Malone, T.W., Crowston, K., Lee, J., Pentland, B.: Tools for Inventing Organizations: Toward a Handbook for Organizational Processes. Management Science 45(3), 425–443 (1999)
27. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)

28. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the occurrence of errors in process models based on metrics. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 113–130. Springer, Heidelberg (2007)
29. Mendling, J., Reijers, H.A., Cardoso, J.: What makes process models understandable? In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 48–63. Springer, Heidelberg (2007)
30. Mutschler, B., Weber, B., Reichert, M.U.: Workflow management versus case handling: Results from a controlled software experiment. In: Liebrock, L.M. (ed.) Proceedings of the ACM Symposium on Applied Computing, vol. I, pp. 82–89 (2008)
31. Ould, M.A.: Business Processes: Modelling and Analysis for Re-engineering and Improvement. Wiley, Chichester (1995)
32. Parnas, D.: On the Criteria for Decomposing Systems into Modules. Communications of the ACM 15(12), 1053–1058 (1972)
33. Prechelt, L.: Kontrollierte Experimente in der Softwaretechnik: Potenzial und Methodik. Springer, Heidelberg (2001)
34. Sadiq, W., Orlowska, M.E.: Analyzing Process Models using Graph Reduction Techniques. Information Systems 25(2), 117–134 (2000)
35. Sarshar, K., Loos, P.: Comparing the control-flow of EPC and Petri nets from the end-user perspective. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 434–439. Springer, Heidelberg (2005)
36. Sharp, A., McDermott, P.: Workflow Modeling: Tools for Process Improvement and Application Development. Artech House (2001)
37. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. CRC Press, Boca Raton (2004)
38. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Evaluating workflow process designs using cohesion and coupling metrics. Computers in Industry (2008)
39. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
40. Verbeek, H.M.W., van Hattem, M., Reijers, H.A., de Munk, W.: Protos 7.0: Simulation made accessible. In: Ciardo, G., Darondeau, P. (eds.) Proceedings of the 24th International Conference on Application and Theory of Petri Nets, pp. 465–474. Springer, Heidelberg (2005)
41. Wand, Y., Weber, R.: On the Deep Structure of Information Systems. Information Systems Journal 5, 203–223 (1995)
42. Weber, B., Rinderle, S., Reichert, M.U.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
43. Weber, R.: Ontological Foundations of Information Systems. Coopers & Lybrand and the Accounting Association of Australia and New Zealand, Melbourne (1997)
44. Wynn, M.T., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Reduction rules for YAWL workflow nets with cancellation regions and or-joins. BPMCenter Report BPM-06-24, BPMcenter.org (2006)
45. Yourdon, E., Constantine, L.L.: Structured Design. Prentice Hall, Englewood Cliffs (1979)

Model Driven Business Transformation – An Experience Report

Juliane Siegeris[1] and Oliver Grasl[2,*]

[1] Gematik, Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH, Friedrichstrasse 136, 10117 Berlin
juliane.siegeris@gematik.de
[2] Transentis management consulting GmbH, Kranzplatz 5-6, 65193 Wiesbaden
oliver.grasl@transentis.com

Abstract. This report details the experience made using BPMN as the process modeling notation for a large-scale modeling effort that formed the heart of a business transformation project. It illustrates the practical limitations encountered in using BPMN and how they were overcome by using UML to extend BPMN. The automated document generation approach used to generate user-friendly process documentation from the BPMN model and the instruments used to drive the business transformation project forward are explained.

1 Introduction

Newly founded, rapidly growing companies face the challenge of making the transition from the ad-hoc processes needed in the start-up phase to the clearly defined responsibilities and the high-performance, repeatable processes needed to sustain the company in an aggressive market place.

This case study reports from a business transformation project at the gematik, a German public-private partnership that is responsible for the specification and implementation of the german health insurance chip card due to be introduced in 2008. The gematik's main "product" is a bundle of specification documents that form the basis for the chip card and the IT infrastructure needed to support it. These specifications are used by independent vendors to implement the various parts of the system.

The company currently (Spring 2008) employs around 180 people in ten departments. Due to the rapid growth of the company a transformation process was started mid-2007 to establish clear responsibilities and defined workflows to ensure efficient definition of high-quality specifications at reasonable cost. At the heart of the transformation is a new matrix organization aligned to the major business processes the company supports.

[*] The authors would like to thank the board of directors and head of quality management at gematik for their support and encouragement during the writing of this paper.

To ensure employee buy-in it was decided early on to involve employees of each new department in the analysis and modeling of the workflows, under coordination of a process architecture team. All in all the process modeling team has 20 members with different backgrounds.

At a very early stage the team decided to use BPMN 1.0 for workflow modeling, as implemented in the repository based modeling tool Enterprise Architect 7.0. The main reasons for choosing BPMN were that BPMN has been standardized by the OMG, is supported by a wide variety of tools, and is readily understood both by users with an IT-background as well as users with a business background. Workflows modeled in BPMN can also be used as a basis for process simulation and automated process execution using technologies such as BPEL4WS—though this is not a goal of the initial modeling project, it is expected that process simulation and automated process execution will be important in the future. The tool Enterprise Architect was chosen because gematik already used this tool to develop its IT specifications—therefore tool support and maintenance know-how was already available within the company.

1.1 Major Challenges

The process modeling team encountered many challenges on the way. This paper focuses on three challenges that concern BPMN and its use in a large, heterogeneous modeling team:

– Practical limitations of Modeling with BPMN. While BPMN is very well suited to modeling workflows, the modeling team encountered a number of limitations modeling organizational responsibilities within the workflows and the complex artifact structure supported by the business processes.
– Using BPMN models as a basis for process documentation. BPMN process models cannot meet the many requirements employees have for the process documentation they need on a daily basis to support their work. In particular easy, role-based navigation and cross-references between roles, processes and artifacts are not directly utilizable though they are (implicitly) modeled in BPMN workflows.
– Organizing large-scale modeling efforts. BPMN is a modeling notation and not a method. The process architecture team defined and utilized a number of instruments that were needed to ensure creation of consistent and valid models.

1.2 Outline of the Paper

This paper details the solutions the process architecture team found in response to the major challenges:

Customizing BPMN. For the description of workflow aspects the rich set of BPMN modeling possibilities was restricted. To overcome its limitations BPMN was extended with language constructs based on UML. These constructs are used to model high-level process architectures, organizational structures and artifact landscapes.

Using BPMN models to generate process documentation. A mechanism was implemented to generate a complete, intranet-based web portal from the BPMN model.

Practical implementation. A number of practical instruments were implemented to fully leverage the work of the process architecture team and to ease the transformation process.

2 Customizing BPMN

The objective of the modeling project was to describe the processes and their context. The blue print had identified 10 core business processes aggregating 129 processes. The required level of detail for a process description varied from an abstract, "black-box" process description just defining the context to very fine granular descriptions e.g. system operations. The contextual information defines the organizational unit owning the process, the roles responsible for the execution of the involved activities, and the specific documents used or produced within the process.

These objectives pose many requirements to the chosen modeling technique. It should support structuring of the processes at different levels of abstraction combined with the possibility to drill-down into a single process, thus moving from an abstract level to the details.

Apart from the workflow itself the description of further aspects, such as organizational and information related aspects, should be supported, thus allowing the modeler to define who works on an activity and which items are produced within an activity.

The decision was made in favor of BPMN, the new OMG standard for business process modeling [2]. BPMN amalgamates best practices within the business modeling community. However, its focus is on the description of activities and their control flow dependencies. A variety of modeling elements can be combined in many different ways, thus providing much freedom in defining the actual process flow. Regarding BPMN's support for other aspects, such as the information and resource perspectives, BPMN's performance is not so good. Although the BPMN provides some concepts for their description, the modeling of an organizational structure, an extensive role concept or a distinct document landscape is beyond BPMN's means, c.f. [2,9,7].

Defining rules for the use of BPMN within the gematik transformation process, these shortcomings were also felt by the gematik process architecture team. To exploit the BPMN for the business transformation project two lines of customization were followed:

Adaption of native BPMN constructs. To define the actual process flow and the interfaces between processes, and to provide an adequate drill down mechanism, the rich set of BPMN modeling elements and their combination possibilities had to be adapted.

Extension of BPMN via UML. The organizational structure, the artifact landscape but also the high-level process structure is modeled using UML-concepts. Here the interface between the provided BPMN-concepts and the separately used UML-models had to be defined.

The necessary customizations are described in more detail in the following sections:

1. Process architecture
2. Organization structure
3. Artifact landscape

Each section is described in identical format: First the gematik requirements are given. Then the chosen BPMN-concepts are listed and the linkage to the used UML mechanisms is described. Finally examples are provided.

2.1 Process Architecture

The new structure of the gematik is process-oriented. Starting form their needs for resources and with the proviso that interfaces should be reduced the new structure focuses on the core processes essential for the value creation of the company. The departments are defined along the core processes, ensuring that each process is owned by exactly one department. The core processes are made up of processes and their interfaces. Only at the level of processes the actual workflow is defined.

BPMN's highest level of abstraction is the workflow. A concept to describe process layers at a higher level is not provided. To reflect the structure of the process landscape the UML-package concept is used.

Core process. Each core process respectively department is modeled by an UML package. These packages are denoted with a special stereotype "core process". In the repository these packages are contained in a super package called "departments". The department package is one of the two main packages of the repository. The second top-level package "cross-process information" contains information relevant to more than one core process such as process interfaces, artifacts and roles.

Process. A core process aggregates a set of processes. Processes are depicted by business process elements[1]. Obligatory information on this level is a precise and unambiguous name and a short textual summary of the process, outlining the process goal, the critical success factors and the risks. The summary is provided using the notes-field of the model element. To reflect dependencies between the processes, they can be grouped (model element group or package) or connected using control-flow arcs or message flow arcs.

[1] In the BPMN Adopted Specification [2] there is no distinct notation for business process. The distinct activity type "business process" originates from the modeling set provided by the chosen modeling tool.

Workflow. The processes are refined describing the actual workflow. The required level of detail for a process description varies from quite abstract descriptions just defining the context to very fine granular descriptions of e.g. system operations.

For the description of the workflow a subset of the BPMN modeling elements was selected. In general all basic flow elements (events, activity, gateways) can be used. For the drill down the BPMN-hierarchy concept is used. Different abstraction levels are supported by the two activity types "sub-process" (which can be refined) and "task" (which is atomic).

On the first level the workflow description is complete, but not detailed. On this level possible start events (process triggers) and possible end events, as well as involved roles and interfaces to other processes are reflected. Participating parties (modeled by BPMN pools and lanes) can be described both using a white- or black-box approach. For white box descriptions, the organizational unit, involved roles and process steps are visible. A black box description only discloses the involved organizational unit—such elements must have a white-box description elsewhere in the model. It is clear that the part of the process owner is always modeled white-box. A further requirement is, that the elements of the process owner are all connected, i.e. they are all on a path from a start event to an end event.

A white-box description of a third party (not the process owner) can be refined. Concerned organizational units can decide to provide their own (enhanced) process description, thus over riding the pre-defined interface. Requirement for the re-definition is that the initially provided modeling elements (and their dependencies) are part of the redefined process. An example for a workflow containing white and black-box descriptions is provided in Figure 1.

Here, the part of the "Expert" is modeled white box. This means that the predefined tasks "comment document" and "send review comments" (and their order) are obligatory for every refinement. The process of the author is modeled black box. Except for the defined interface[2], no restrictions are made to their workflow.

Sub-process refinement. Any sub-process can be refined into sub-workflows. All modeling constructs available to process workflows are also available to sub-process workflows. The only restriction is that the sub workflow always starts with one start- and ends with one end event.

Note. The hierarchy must be strictly met. Link model elements from different levels via flows is prohibited—elements are linked to their parent element by containment and can be connected to elements at the same level via flows or messages. Moreover, every flow element should be part of a desired execution path. In fact, mapping the requirements to possible correctness criteria [6], only relaxed soundness[4,5] is applicable. Well-structuredness [1,3,8] is not required.

[2] An interface between different organizational units is always modeled using a message flow pointing to a message event.

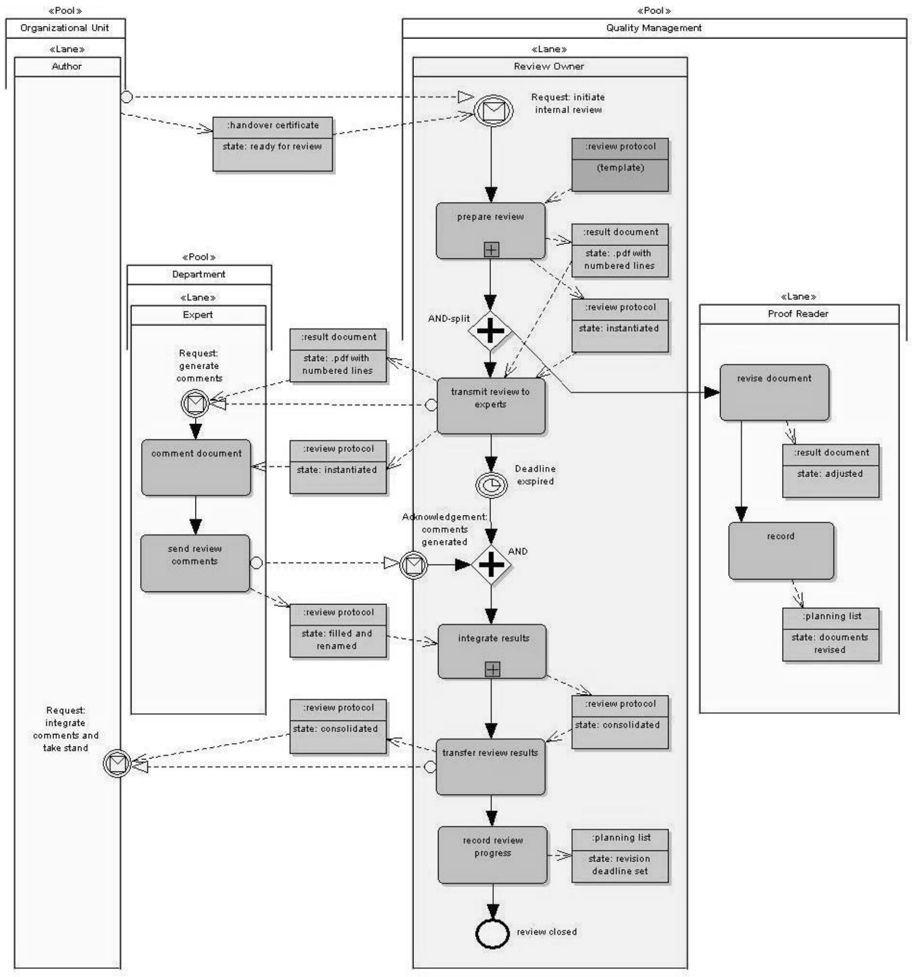


Fig. 1. The workflow within the review process

Soundness [1] cannot be guaranteed, because of the OR-gateway, which is allowed to reflect optional flow.

2.2 Organization Structure

Tasks are executed by responsible actors (mostly persons). Responsibilities are summarized in role(description)s which are assigned to positions within an organization. In our context[3] this relationship was condensed to roles that belong

[3] We only consider the description of processes and not their instantiation at run time. So we do not need to reflect the relationship between roles and concrete actors.

to organizational units. Several roles can be involved within one process. One distinguished role is said to be the process owner.

The BPMN provides model elements to organize and categorize activities. These are groups, lanes and pools. Pools are used to represent participants of the process. A lane is a sub-partition of a pool.

In the gematik process landscape roles are modeled using the model element lane. Organizational units are modeled using the model element pool. Every task has to be assigned to a role, that is the corresponding model element (activity or sub process) is contained in a lane. Every role is assigned to an organizational unit. Therefore every lane is contained in a pool.

The organizational structure of the gematik and the set of possible roles has been modeled separately using UML-class diagrams. The class diagrams describing the organizational structure do not only contain class representations for concrete organizational units (e.g. "Test-Department", "Quality Management") but also for abstract generalizations, e.g. classes "Organizational Unit", and "Department".

Associated to the organizational units are the available roles. As the set of classes, the roles are divided into generic and internal roles. The first describes roles that can be applied to people in potentially all organizational units. Examples for such roles are "Author", "Expert", "Project Manager", "Head of Department", "Process Responsible" or "Employee". The internal roles are specific to a certain department. Examples for the latter are "Employee of Quality Management Department", "Review Owner", or "Head of Test Department".

Figure 2 shows an extract of the metamodel describing the gematik's organizational structure. It is important to note that every role or organizational unit contains a detailed description which is aligned to the authorized definition in the organization manual.

In order to guarantee consistency between the BPMN process models and the UML-class diagrams the following convention must be followed: Every lane and pool within the process model (BPMN) has to be linked to one class (either generic or specific) of the organization model (UML). The compliance with the defined rule is checked with the help of a validation script. Regularly invoked it checks whether the roles defined in the gematik organization model (UML classes) are well-defined. It then verifies that every lane used in the process model has its counterpart within the organization model. If so, a link is set between the two elements. Otherwise the mismatch is reported.

Figure 1 shows the process model describing the internal review procedure. It involves up to three different organizational units, and four different roles. The "Review Owner" (process owner) and the "Proof Reader" are specific roles in the quality management department. Therefore the corresponding lanes are contained in a pool representing this specific department. The "Author" of the reviewed specification can come from either a department or from a project. The class (respectively pool) generalizing both is the "Organizational Unit". The technical reviewing (role "Expert") is a task that belongs to the responsibilities

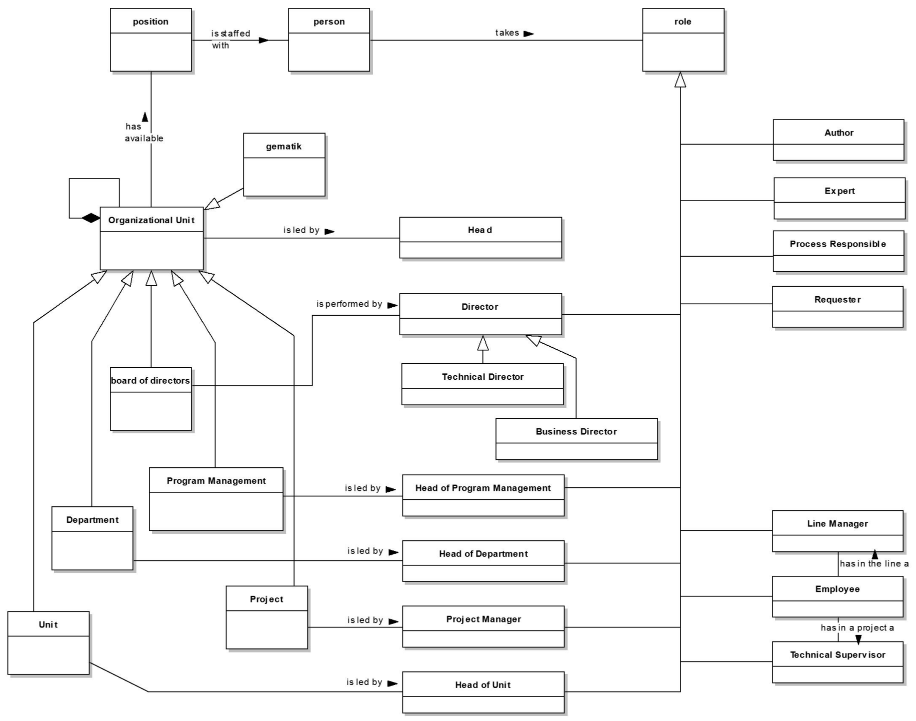


Fig. 2. Organization model (extract): The generic organizational structure

of the line structure. The corresponding role is therefore associated only with departments.

2.3 Artifact Landscape

Artifacts, in particular specification documents, play a prominent role in the gematik context. These documents constitute the actual "product" of the company. Most of the processes are centered around their development, respectively improvement, and their publication. Correspondingly, the artifacts used and produced within the processes must be modeled within the process descriptions. Two aspects are modeled: the document type and the progression of a single document instance.

The only elements BPMN provides to depict information related aspects, are the model element "data object" and specific connectors to depict data flow. Detailed data and information models are beyond the scope of BPMN.

To overcome this shortcoming, we followed a similar approach as sketched for the organizational modeling. The gematik artifact landscape had been described with an UML-class diagram. It contains a generalized class "gematik artifact" with common attributes, such as state, status, scope, storage type, and storage place. Derived classes are e.g. "document", "form", "document template",

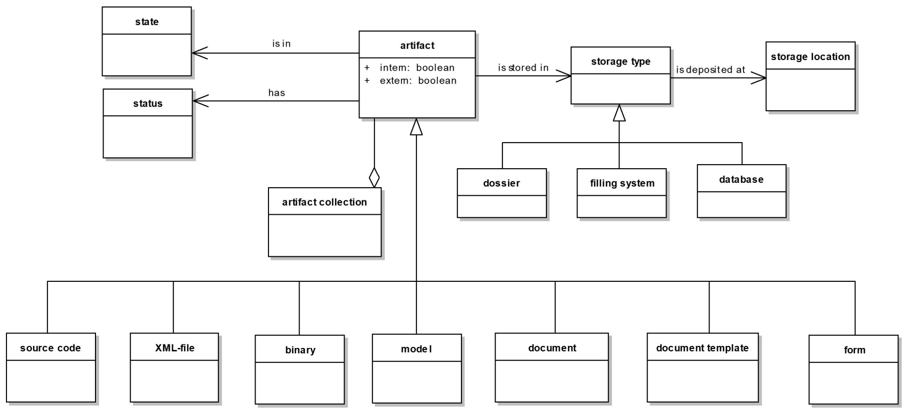


Fig. 3. Information model: The gematik's artifact landscape

"source code", "binary", "model" and an aggregation, the "artifact collection". Figure 3 shows the information model describing the gematik artifact landscape.

In order to use the different artifact types within the process models, the UML-diagram was transformed into an UML-profile with corresponding stereotypes. This profile was imported into the EA, enhancing the set of possible stereotypes for the model element Artifact.

Artifacts are built and stored independently from the process models in a super-ordinate package "Artifacts". Creating an artifact, its type has to be determined, applying one of the pre-defined stereotypes. Every artifact contains a textual description outlining its specific purpose. Further information about the artifact can be given using the attributes provided. Corresponding tagged values can be set to denote e.g. the state, the status, the scope and its storage location.

In the process descriptions, instances of the artifact are linked. The handling of an artifact within a process can be documented using (respectively overwriting) again the mentioned attributes.

Examples for the use of artifacts are provided in the Review Process in Figure 1. Here the "review protocol" is the main document. The process is triggered using the interface "Request: initiate internal review" and transferring a certificate referring to the artifacts to be reviewed. During the first activity "prepare review" the frame of the review is fixed. This includes determining a deadline and the circle of reviewers. This information is noted in the review protocol (state:instantiated). In parallel, the documents to be reviewed are converted into a line numbered pdf-format. For reporting purposes a record is created in the internal planning list. In the next step, the review protocol and the pdf-documents are transferred to the reviewers (interface "Request: generate comments"). They use the protocol to include their comments. The completed protocol is renamed and sent back to the QM-department (interface "Acknowledgment: comments generated").

The review owner waits for the deadline to be reached and than collects the review results. The different comments are integrated into one form and reordered. Redundant comments are combined. A deadline for the revision is set. The consolidated protocol and the revision deadline are transmitted back to the author (interface "Request: integrate the comments and respond"). The process is closed by recording the revision deadline in the planning list. In addition to review coordination, the QM-department also performs formal revision of the documents. This task is accomplished by the role "Proof Reader".

3 Using BPMN Models to Generate Process Documentation

An important pre-requisite for successful organizational transformation is that the new organizational structure and the redesigned workflows are understood, implemented and found useful by all employees concerned.

While BPMN models are useful for analyzing and redesigning workflows, the process architecture team quickly realized that a BPMN model per se is neither an ideal instrument for communicating the essence of newly designed process landscapes nor suited as process documentation to support day-to-day work:

- BPMN models are centered around processes and workflows, not roles and artifacts.
- Modeling tools can only be used easily by experienced users
- Comprehending and navigating complex models is difficult without guidance
- Useful textual annotations are hidden in the notes of the model and not directly visible

After some discussion the following high-level requirements for the process documentation were elicited from the members of the process team:

- The documentation must provide quick and easy access to the information needed
- The documentation must provide a role-based view, answering the questions "Which processes am I involved in?" and "Which artifacts do I have to produce?"
- The documentation must provide a process-based view, answering the question "Who else is involved in this process?" and "How will we work together to produce the required results?"
- The documentation must provide an artifact-based view, answering the questions "Which inputs must I use to create this artifact?" and "Which templates can I use to create this artifact?" and "To whom must I deliver the results?"

A quick check of these requirements showed that most, if not all, of the information needed for the process documentation was actually already contained in the model, or could be provided by the model through simple extensions. Therefore the process team realized that the main issue in creating the process documentation lay not in writing wholly new information, but in extracting the information from the model, presenting it in new form, and making it readily accessible.

3.1 An Intranet-Based, Fully Generated Process Portal

In consequence the decision was made to create process documentation which could be fully generated from the process model. The solution to the ease of use and quick access requirement was to create a process portal based on web-technology, implementing the following features:

– The process portal offers a process-based, role-based and artifact-based access to the process documentation. For example this means that an employee can start at the role "quality manager" and immediately find all processes a quality manager is involved in and all artifacts a "quality manager" creates. If she clicks on one of these artifacts, e.g. "audit protocol", she will find all processes this artifact is involved in.
– The process portal offers direct access to artifacts of everyday importance such as document templates, contact information for process owners, and guidelines.
– The process portal is the binding source for information on organization charts, process descriptions and role descriptions authorized by top management.
– The process portal is fully generated from the process model.

3.2 The Approach Taken in Developing the Process Portal

Due to high time pressure the process portal was created in parallel to the process model. The following approach was adopted.

1. Create a mock-up.
2. Choose an initial set of processes.
3. Refine the meta-model.
4. Align initial process model to meta-model.
5. Implement portal generator.
6. Evaluate process portal prototype.
7. Align process model to refined meta-model.
8. Generate first process portal.

Create a mock-up. The mock-up is used to ensure early end-user involvement, gain buy-in from management, and agree on the layout and style of the process portal. The mock-up (and later the process portal) is created using XML and XSLT technology. In this way the presentation of the process portal (defined in XSLT) is independent of the content (defined in XML). The portal generator then just needs to generate XML and is not concerned with the presentation itself. The presentation can later be refined by web designers to maximize ease of use and the appeal of the graphical design.

The process portal is illustrated in figure 4.

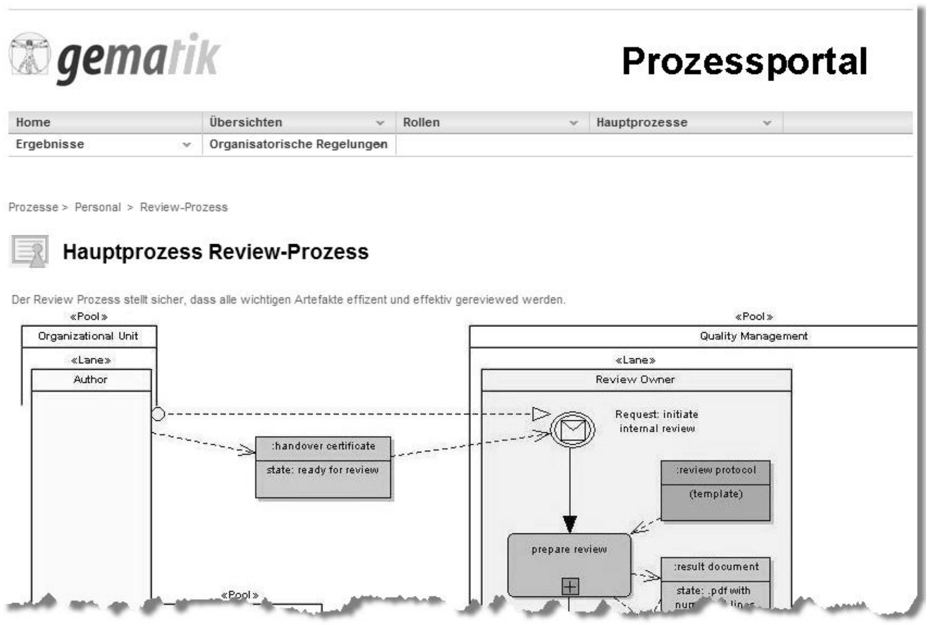

Der Review Prozess stellt sicher, dass alle wichtigen Artefakte effizent und effektiv gereviewed werden.

Fig. 4. The review process within the process portal

Choose an initial set of processes. This step is necessary to ensure that development of the portal generator is not impaired by changes made to the processes or the structure of the model. It is sensible to choose a small set of processes initially that have already been released (and are therefore complete).

Refine the meta-model. Once the mock up is completed the meta-model must be reviewed to check if all information to be displayed in the process portal is considered in the meta-model.

Align initial process model to meta-model. Once the meta-model has been completed the initial process model must be aligned to the meta-model. In particular it must be ensured that all model elements are used correctly and all necessary relationships are defined.

Implement portal generator. The portal generator is implemented using Java technology. It first traverses the model and generates XML files for each element corresponding to the major views in the model (e.g. one XML file for each process, role and artifact). In a second pass through the model it generates information necessary for the menu structure of the process portal.

Evaluate process portal prototype. At this stage, the process portal prototype contained complete process information for those processes included in the initial process model. Small changes to the handling of the process portal and the information displayed were requested. The portal generator, the process portal presentation layer and the meta-model were finalized on this basis.

Align process model to refined meta-model. Once the meta-model is complete, the main process model must be aligned to the new meta-model to ensure the process model can be used as a basis for the portal generator.

Generate first complete process portal. Finally a process portal based on all released processes is generated.

4 Practical Implementation

The set of processes to be modeled is extensive. In the initial blue print 129 processes were identified. The modeling effort is being conducted by all departments to ensure early buy-in of gematik employees. This means, many people with different backgrounds are involved in the modeling. It is clear, that the modelers have to be coordinated to gain a consistent process landscape.

Consistency is a prerequisite to use the process portal as a solid basis for the communication of workflows and responsibilities across the company. Consistency can only be achieved if the description of the individual processes are at similar levels of abstraction, use similar language, and provide the same look and feel. Furthermore, the processes must fit together building an integrated whole. In order to ensure the model can be used for automatic generation of the process portal, the model must be compliant to the metamodel.

At the level of BPMN models this goal can only be reached if all processes reflect the same level of granularity, use the same set of modeling concepts and conventions and reference artifacts and roles from a common framework.

4.1 The Process Architecture Team and the Modeling Guideline

In order to implement the proposed process landscape, an internal team was commissioned and authorized to coordinate the modeling effort. In the beginning the process team consisted of two process architects and a tool expert. Later on, the team was strengthened by 17 modelers, one of each unit.

In the run-up to the modeling at large, the core team developed the modeling framework. This included the decision for the modeling language, the tools to be used, the setup of a common modeling repository and the modeling of the process architecture based on the initial blue print. The core task at this stage was to develop a common set of modeling guidelines. These were then used to train the modelers.

The modeling guidelines are laid down in a specific document, which is the core instrument in coordinating the modelers. The document contains a short introduction to the relevant subset of BPMN modeling elements and the rules for their application in the gematik context.

The guidelines support the modeling, determining what has to be described and which conventions are to be followed. They define the criteria that have to be fulfilled to gain formal approval of the process descriptions. To maximize support the modeler, these criteria are summarized in the appendix of the modeling guidelines.

The rules defined in the modeling guideline implicitly determine a metamodel. Later on this metamodel was made explicit and formalized. On the one hand the metamodel is an instrument the architecture team can use to validate the modeling guidelines, on the other hand it serves as a basis for automatic validation of the model (via validation scripts) and for the automatic generation of the process documentation (via generating scripts).

4.2 Further Instruments Utilized by the Process Architecture Team

The process architecture team used a number of other instruments to improve communication within the modeling team.

Wiki. The wiki was set up to support the modelers during their modeling efforts. It is used to provide tips and tricks in using the modeling tool. Moreover, it allows quick response to questions. Changes to the modeling guideline are also communicated via the wiki.

Training. In the beginning the modelers were trained individually or in groups. Joint modeling of prototypical process proved to be particularly effective: Working in pairs using the modeling tool the modelers could gain hands-on experience of the tool's user interface and the modeling guidelines.

Regular process meetings. The entire modeling team meets weekly. The meeting is used to communicate the processes throughout the company, coordinate handovers between the processes and to exchange encountered problems and best practices.

Approval process. An approval process has been established to validate both the process architecture and the description of the individual processes. It involves validation of the process content by the head of the department owning the process, and validation of compliance to the process architecture and modeling guidelines by the process architect. In case the process contains handovers to processes from other departments, theses interfaces have to be approved by the corresponding department. Cross organizational processes need final approval by the board of directors.

5 Conclusions

The project this paper is reporting from is on-going. So far 60% of the processes identified have been modelled in detail.

BPMN was found to be well-suited to the requirements encountered in this business transformation project. The limitations concerning BPMN illustrated here were successfully overcome using UML extensions. No further problems are expected in this area.

The next months will show whether the process portal will be accepted by gematik employees and used as an instrument to ease daily work. The resonance has been very positive so far. The practical instruments used by the process architecture team have proved to be very effective, they will be refined further as the business transformation project continues. It is clear that the business transformation will not be finished once the process modeling effort is completed. The next major challenge will be to ensure the processes are accepted by all stakeholders and are executed in daily practice. The experience made during process execution will be discussed at meetings of the process owners and feed back into the process model and thus the process portal at regular interval.

References

1. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
2. Business Process Modeling Notation (BPMN 1.0): OMG Final Adopted Specification (2006)
3. Chrzastowski-Wachtel, P., Benatallah, B., Hamadi, R., O'Dell, M., Susanto, A.: A Top-Down Petri Net-Based Approach for Dynamic Workflow Modeling. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 336–353. Springer, Heidelberg (2003)
4. Dehnert, J., Rittgen, P.: Relaxed Soundness of Business Processes. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 157–170. Springer, Heidelberg (2001)
5. Dehnert, J., van der Aalst, W.M.P.: Bridging the Gap Between Business Models and Workflow Specifications. Int. Journal of Cooperative Information Systems (IJCIS) 13(3), 289–332 (2004)
6. Dehnert, J., Zimmermann, A.: On the Suitability of Correctness Criteria for Business Process Models. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 386–391. Springer, Heidelberg (2005)
7. Recker, J., Indulska, M., Rosemann, M., Green, P.: How good is BPMN really? Insights from Theory and Practice. In: Ljungberg, J., Andersson, M. (eds.) 14th European Conference on Information Systems, Goeteborg, Sweden (2006)
8. Verbeek, E.: Verification of WF-nets. PhD thesis, TU Eindhoven (2004)
9. Wohed, P.P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russell, N.: On the suitability of BPMN for business process modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 161–176. Springer, Heidelberg (2006)

Supporting Flexible Processes through Recommendations Based on History

Helen Schonenberg, Barbara Weber, Boudewijn van Dongen, and Wil van der Aalst

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.h.schonenberg, b.f.v.dongen, w.m.p.v.d.aalst}@tue.nl
Department of Computer Science, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at

Abstract. In today's fast changing business environment flexible Process Aware Information Systems (PAISs) are required to allow companies to rapidly adjust their business processes to changes in the environment. However, increasing flexibility in large PAISs usually leads to less guidance for its users and consequently requires more experienced users. To allow for flexible systems with a high degree of support, intelligent user assistance is required. In this paper we propose a recommendation service, which, when used in combination with flexible PAISs, can support end users during process execution by giving recommendations on possible next steps. Recommendations are generated based on similar past process executions by considering the specific optimization goals. In this paper we also evaluate the proposed recommendation service, by means of experiments.

1 Introduction

In todays fast changing business environment, flexible Process Aware Information Systems (PAISs) are required to allow companies to rapidly adjust their business processes to changes in the environment [7]. PAISs offer promising perspectives and there are several paradigms, e.g., adaptive process management [13], case handling systems [16] and declarative processes [11,12] (for an overview see [18,20,3]).

In general, in flexible PAIS it occurs frequently that users working on a case, i.e., a process instance, have the option to decide between several activities that are enabled for that case. However, for all flexibility approaches, the user support provided by the PAIS decreases with increasing flexibility (cf. Fig. 1), since more options are available, requiring users to have in-depth knowledge about the processes they are working on. Traditionally, this problem is solved by educating users (e.g., by making them more aware of the context in which a case is executed), or by restricting the PAIS by introducing more and more constraints on the order of activities and thus sacrificing flexibility. Both options, however, are not satisfactory and limit the practical application of flexible PAISs.

In this paper, we present an approach for intelligent user assistance which allows PAISs to overcome this problem and to provide a better balance between flexibility and support. We use event logs of PAISs to gain insights into the process being supported without involving a process analyst and we propose a tooling framework to provide continuously improving support for users of flexible PAISs. At the basis of our approach lie so-called recommendations. A recommendation provides information to a user about how he should proceed with a partial case (i.e., a case that was started but not completed yet), to achieve a certain goal (e.g., minimizing cycle time, or maximizing profit). In this paper we discuss several methods for calculating log-based recommendations. In addition, we describe the implementation of our approach as recommendation service and its evaluation. The remainder of this paper is structured as follows. In Section 2, we present the requirements and an overview of the recommendation service. Then, in Section 3 we define a log-based recommendation service. In Section 4, we describe the experiment we conducted to evaluate whether recommendations indeed help to achieve a particular goal. Finally, we discuss related work in Section 5 and provide conclusions in Section 6.

2 Overview

Fig. 2 illustrates the envisioned support of users of flexible PAISs through a recommendation service. In general, each business process to be supported is described as process model in the respective PAIS. We consider both imperative and declarative process models. In fact, our approach is most useful when the process model provides the user a lot freedom to manoeuvre, i.e., multiple activities are enabled during execution of a case. At run-time, cases are created and executed considering the constraints imposed by the process model. In addition, the PAIS records information about executed activities in event logs. Typically, event logs contain information about start and completion of activities, their ordering, resources which executed them and the case they belong to [1].

As illustrated in Fig. 2, the recommendation service is initiated by a request from the user for recommendations on possible next activities to execute. In this request, the user sends the recommendation service information about the partially executed case, i.e., (1) the currently enabled activities, and (2) the history of executed activities, which we call the partial trace. Information about the partial trace is required because the decision which activities to perform next for a particular case usually depends on the activities already performed for this case. In addition, only enabled activities are considered to ensure that no recommendations are made that violate the constraints imposed by the process model. The recommendation service then provides the PAIS a recommendation result, i.e., an ordering of recommendations where each recommendation refers to one activity and some quality attributes (e.g., expected outcome) explaining the recommendation. Recommendations are ordered such that the first recommendation

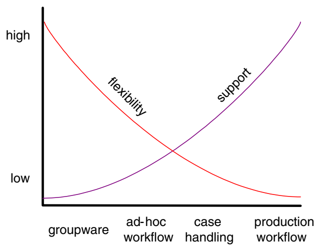


Fig. 1. PAIS trade-offs [5]

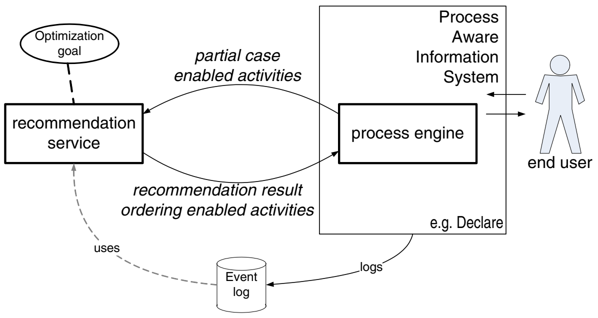


Fig. 2. An overview of the recommendation service

in the list is most likely to help the user achieving his goal, i.e., optimizing a certain target, such as profit, cost, or cycle time. Different users can have different targets, resulting in different recommendations.

As an example we describe a fictive process of applying for a building permit at a town hall. Initially, the employee has to do several tasks; (A) bill registration fee (B) register the application details, (C) initiate permission procedure, (D) announce the application in local newspaper, and (E) inform applicant. The employee can decide in which order to execute these tasks. Ideally, the employee finishes these as soon as possible. All tasks have a fixed duration, however, tasks B and C use the same database application and if B is directly followed by C, then the combined duration of the tasks is much shorter, since there is no closing-time for B an not set-up time for C, moreover C can use the data provided by B, without data re-entry. The recommendation service can guide employees to execute in the faster order of tasks.

In this simple example the use of recommendations seems to be an overkill as the user only has to select among a limited set of options. In the presence of real life flexible processes, with increasing complexity there are so many options for users, that user support becomes fundamental. At the same time, giving recommendations based on extracted knowledge from execution logs can provide knowledge that was not available during the design of the process.

3 Log-Based Recommendation Service

In this section, we present a concrete definition of a log-based recommendation service for providing users with recommendations on next possible activities to execute. Recommendations for an enabled activity provide predictive information about the user goal, based on observations from the past, i.e., fully completed traces accompanied by their target value (e.g., cost, cycle time, or profit), that have been stored in an event log. The log-based recommendation service requires the presence of an event log that contains such information about cases that have been executed for a certain process.

3.1 Preliminaries

Let A be a set of activities. A^* denotes a set of finite sequences over A. A trace $\sigma \in A^*$ is a finite sequence of activities, where $|\sigma| = n$ is the length of the sequence. Sequences are denoted as $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ and we denote $\forall_{1 \leq i \leq n}\ \sigma(i) = a_i$.

On traces, we define the standard set of operators.

Definition 1 (Trace operators). *Let* $\sigma : \{1, \ldots, n\} \to A$ *and* $\sigma' : \{1, \ldots, m\} \to B$ *be traces with* $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ *and* $\sigma' = \langle b_1, b_2, \ldots, b_m \rangle$.

Prefix	$\sigma \leq \sigma' \iff n \leq m \wedge \forall_{1 \leq i \leq n}\ a_i = b_i$
Concatenation	$\sigma ^\frown \sigma' = \langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m \rangle$
Membership	$a \in \sigma \iff \exists_{1 \leq i \leq n}\ a_i = a$
Parikh vector	$par(\sigma)(a) = \#_{0 \leq i \leq n}\ a_i = a.$

The Parikh vector $par(\sigma)(a)$ denotes the number of occurrences of a in a trace σ, e.g., $par(\langle a, b, c, a, b, c, d \rangle)(a) = 2$.

For multi-sets (bags), we introduce standard notation to denote the universe of multi-sets over a given set. Let S be a set, then the universe of multi-sets over S is denoted by $\mathcal{B}(S)$, with $X \in \mathcal{B}(S)$, denoted as $X : S \to \mathbb{N}$ is a multi-set, where for all $s \in S$ holds that $X(s)$ denotes the number of occurrences of s in $X(s)$. We will use $[\![a, b^2, c^3]\!]$ to denote the multi-set of one a, two b's and three c's as a shorthand for the multi-set $X \in \mathcal{B}(A)$ where $A = \{a, b, c\}$, $X(a) = 1, X(b) = 2, X(c) = 3$. Furthermore, multi-set operators such as for for union $\uplus$, intersection $\Cap$, and submulti-set $\sqsubseteq$, $\sqsubset$ are defined in a straightforward way and can handle a mixture of sets and multi-sets.

Definition 2 (Event log). *Let A be a set of activities. An event log $L \in \mathcal{B}(A^*)$ is a multi-set of traces referring to the activities in A.*

Recall that each recommendation contains predictive information regarding the user goal. For now, we assume that this goal can be captured by a function on a trace, i.e., each trace σ in an event log has a target value (e.g., cost, cycle time, or profit) attached to it.

Definition 3 (Target Function). *Let A be a set of activities and $\sigma \in A^*$ a sequence of activities. We define $\tau(\sigma) \in \mathbb{R}^+$ to represent the target value of the sequence σ.*

Note that τ is not *a function, as similar sequences might have different values attached to them. However, τ is total, i.e. it provides a value for all sequences.*

3.2 Recommendations

A recommendation is initiated by a recommendation request, which consists of a partial trace and a set of enabled activities. Formally, we define a recommendation request as follows.

Definition 4 (Recommendation request). *Let A be a set of activities and $\rho \in A^*$ a partial trace. Furthermore, let $E \subseteq A$ be a set of* enabled activities. *We call $r = (\rho, E)$ a recommendation request.*

An activity accompanied by predictive information regarding the user goal is called a recommendation. For each enabled activity, we determine the expected target value when doing this activity (do), and the expected target value for alternatives of the enabled activity, i.e., other enabled activities ($dont$). Precise definitions of do and $dont$ are given in Definitions 10 and 11. A recommendation result is an ordering over recommendations.

Definition 5 (Recommendations). *Let A be a set of activities and $L \in \mathcal{B}(A^*)$ an event log over A. Furthermore, let (ρ, E) be a recommendation request with $E \subseteq A$, $|E| = n$ and $e \in E$ an enabled activity.*

- *$\big(e, do(e, \rho, L), dont(e, \rho, L)\big) \in E \times \mathbb{R} \times \mathbb{R}$ is a recommendation. We use $\mathfrak{R}$ to denote the universe of recommendations.*
- *A recommendation result $\mathcal{R} = \langle \big(e_1, do(e_1, \rho, L), dont(e_1, \rho, L)\big), \big(e_2, do(e_2, \rho, L), dont(e_2, \rho, L)\big), \ldots, \big(e_n, do(e_n, \rho, L), dont(e_n, \rho, L)\big)\rangle$ is a sequence of recommendations, such that $\mathcal{R} \in \mathfrak{R}^*$ and $\forall_{1 \leq i < j \leq n}\ e_i \neq e_j$.*

The nature of the ordering over recommendations is kept abstract, however, we provide a possible ordering for a recommendation result in Example 1, Section 3.6. In the next section we describe how recommendations are generated by the recommendation service based on an existing event log L.

3.3 Trace Abstraction

When generating log-based recommendations only those traces from the event log should be considered, which are relevant for determining the predictive information of an enabled activity. From those traces the ones with a high degree of matching with the partial trace execution should be weighted higher than those with small or no match.

To determine which log traces are relevant to provide recommendations for a given partial trace and to weight them according to their degree of matching we need suitable comparison mechanisms for traces. Our recommendation service provides three different trace abstractions based on which traces can be compared, namely, prefix, set and multi-set abstraction. The prefix abstraction basically allows for a direct comparison between the partial trace and a log trace. In practice such a direct comparison is not always relevant, e.g., when the ordering, or frequency of activities is not important. Therefore we provide with set and multi-set two additional abstractions. They are independent of the domain context, e.g., they do not assume the process to be a procurement process or an invoice handling process [17].

Definition 6 (Trace abstraction). *Let A be a set of activities, $L \in \mathcal{B}(A^*)$ be an event log and $\sigma \in L$ be a trace. $\sigma_p = \sigma$ denotes the prefix abstraction of σ, $\sigma_s = \{a \mid a \in \sigma\}$ denotes the set abstraction of σ and $\sigma_m = par(\sigma)$ denotes the multi-set abstraction of σ, i.e., for all $a \in \sigma$ holds that $\sigma_m(a) = par(\sigma)(a)$.*

In Section 3.4 we explain how we determine which log traces are relevant for obtaining predictive information of an enabled activity. In Section 3.5 we describe how we calculate the weighting of log traces.

3.4 Support

The relevance of log traces for a recommendation is determined on basis of support. Typically, traces that are relevant are those that support the enabled activity for which the recommendation is computed. What support exactly means here, depends on the trace abstraction used.

For the prefix abstraction, we say that a log trace σ supports enabled activity e, if and only if e occurs in σ at the same index as in the partial trace ρ, when this activity is executed. For set abstraction, we consider a log trace σ to support the enabled activity e whenever activity e has been observed at least once in the log trace. To support an enabled activity e in multi-set abstraction of trace σ, the the frequency of activity e in the partial trace ρ must be less than the frequency in the log trace σ, i.e., by executing e after ρ, the total number of e's does not exceed the number of e's in σ.

Definition 7 (Activity support functions). *Let A be a set of activities, $\rho, \sigma \in A^*$ and enabled activity $e \in A$. We use the predicate $s(\rho, \sigma, e)$ to state that log trace σ supports the execution of e after partial trace ρ. The predicate is defined for the three abstractions by:*

$$\begin{aligned} s_p(\rho, \sigma, e) &\iff \sigma_p(|\rho| + 1) = e \\ s_s(\rho, \sigma, e) &\iff e \in \sigma_s \\ s_m(\rho, \sigma, e) &\iff \rho_m(e) < \sigma_m(e) \end{aligned}$$

The support predicate is used to filter the event log by removing all traces that do not support an enabled activity.

Definition 8 (Support filtering). *Let A be a set of activities and $L \in \mathcal{B}(A^*)$ an event log over A. Furthermore, let (ρ, E) be a recommendation request with $\rho \in A^*$ and $E \subseteq A$. We define the log filtered on support of enabled activity $e \in E$ and partial trace ρ as $L^s_{(\rho,e)} = [\![\sigma \in L \mid s(\rho, \sigma, e)]\!]$*[1]

Log traces from $L^s_{(\rho,e)}$ support enabled activity e and are used for the recommendation of e. Next, we define a weighing function (ω) to express the relative importance of each of these log traces for the recommendation of an enabled activity e.

3.5 Trace Weight

The support of an enabled activity determines the part of the log that serves as a basis for a recommendation. However, from the traces supporting an enabled

[1] Note that σ ranges over a multi-set traces.

activity, not every one is equally important, i.e., some log traces match the partial trace better than others. Hence, we define weighing functions that assign a weight to each log trace. The weight of a trace can be between 1 and 0, where a value of 1 indicates that two traces fully match and 0 that they do not match at all. The calculation of the degree of matching depends on the trace abstraction. For prefixes, the weight of a log trace is 1 if the partial trace is a prefix of the log trace, otherwise, the weight is 0. For the set abstraction, the weight of the log trace is defined as the fraction of distinct partial trace activities that the partial trace abstraction and log trace abstraction have in common. The weight of a trace for the multi-set abstraction is similar to the set-weight, however, the frequency of activities is also considered.

Definition 9 (Weight functions). *Let A be a set of activities and $\sigma, \rho \in A^*$. We define $\omega(\rho, \sigma)$,i.e., the relative importance of a log trace σ when considering the partial trace ρ as follows:*

$$\omega_p(\rho,\sigma) = \begin{cases} 1 \text{ , } \mathit{if} \quad \rho_p \leq \sigma_p \\ 0 \text{ , } \mathit{otherwise} \end{cases} \text{ , } \omega_s(\rho,\sigma) = \frac{|\rho_s \cap \sigma_s|}{|\rho_s|} \text{ , } \omega_m(\rho,\sigma) = \frac{|\rho_m \Cap \sigma_m|}{|\rho_m|}$$

3.6 Expected Outcome

Definition 5 states that a recommendation for enabled activity e contains predictive information about the target value. We define the expected outcome of the target value (*do* value), when e is executed in the next step, as a weighted average over target values of log traces from $L^s_{(\rho,e)}$, the log filtered on support of e. The target value of each trace from $L^s_{(\rho,e)}$ is weighted (ω) on basis of the degree of matching with the partial trace.

Definition 10 (*do* calculation). *Let A be a set of activities, τ a target function, $\rho, \sigma \in A^*$, $L \in \mathcal{B}(A^*)$ and $e \in E \subseteq A$ an enabled activity. The expected target value when ρ is completed by the user after performing activity e next is defined as:*

$$do(e,\rho,L) = \frac{\sum_{\sigma \in L^s_{(\rho,e)}} \omega(\rho,\sigma) \cdot \tau(\sigma)}{\sum_{\sigma \in L^s_{(\rho,e)}} \omega(\rho,\sigma)}$$

Similarly, we define the expected target value of not doing an enabled activity e[2]. The *dont* function determines the weighted average over *all* alternatives of e, i.e., all traces that do not support the execution of e after ρ, but do support any of the alternatives e' after ρ.

Definition 11 (*dont* calculation). *Let A be a set of activities, τ a target function, $\rho, \sigma \in A^*$, $L \in \mathcal{B}(A^*)$ and $e, e' \in E \subseteq A$ enabled activities. The expected*

[2] Note that in both *do* and *dont* Σ ranges over a multi-set of traces.

Log		Weight and support			
σ	cost	$\omega_s(\rho,\sigma)$	$s_s(\rho,\sigma,e)$		
			$e=A$	$e=B$	$e=C$
ABC	900	0	$\top$	$\top$	$\top$
DBC	500	0.5		$\top$	$\top$
FBC	500	0.5		$\top$	$\top$
DFA	1000	1	$\top$		
DFB	1500	1		$\top$	
DFC	2000	1			$\top$
DFH	1260	1			
CCA	1680	0	$\top$		$\top$

Fig. 3. Example log, with weight and support values for $\rho = \langle D, F\rangle$

target value when ρ is completed by the user after not performing activity e next is defined as:

$$dont(e,\rho,L) = \frac{\sum_{e'\in E\setminus\{e\}} \sum_{\sigma \in L^s_{(\rho,e')} \setminus L^s_{(\rho,e)}} \omega(\rho,\sigma)\cdot\tau(\sigma)}{\sum_{e'\in E\setminus\{e\}} \sum_{\sigma \in L^s_{(\rho,e')} \setminus L^s_{(\rho,e)}} \omega(\rho,\sigma)}$$

Next, we provide an example calculation for a recommendation, based on a concrete partial trace, a set of enabled events and a log.

Example 1 (Recommendation). Suppose $\rho = \langle D, F\rangle$ is a partial trace and $E = \{A, B, C\}$ is the set of enabled activities. Together, they form a recommendation request (ρ, E). The log is given by $L = [\![\langle A, B, C\rangle, \langle D, B, C\rangle, \ldots]\!]$, with $\tau(\langle A, B, C\rangle) = 900$, $\tau(\langle D, B, C\rangle) = 500$, etc. (cf. Fig. 3). For convenience, we also provide the values for support $(s_s(\rho, \sigma, e))$ and trace weight $(\omega_s(\rho, \sigma))$. For each log trace, support is denoted by $\top$. The user wants to minimize the cost and uses set abstraction. The *do* and *dont* values for the recommendation are calculated as follows.

$$do(A,\langle D,F\rangle,L) = \frac{0\cdot 900 + 1\cdot 1000 + 0\cdot 1680}{0+1+0} = 1000$$

$$do(B,\langle D,F\rangle,L) = \frac{0\cdot 900 + 0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 1500}{0+0.5+0.5+1} = 1000$$

$$do(C,\langle D,F\rangle,L) = \frac{0\cdot 900 + 0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 2000 + 0\cdot 1680}{0+0.5+0.5+1+0} = 1250$$

$$dont(A,\langle D,F\rangle,L) = \frac{(0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 1500) + (0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 2000)}{0.5+0.5+1+0.5+0.5+1} = 1125$$

$$dont(B,\langle D,F\rangle,L) = \frac{(1\cdot 1000 + 0\cdot 1680) + (1\cdot 2000 + 0\cdot 1680)}{1+0+1+0} = 1500$$

$$dont(C,\langle D,F\rangle,L) = \frac{(1\cdot 1000) + (1\cdot 1500)}{1+1} = 1250$$

The implementation of our recommendation service orders the enabled activities on the difference between *do* and *dont*, i.e., the bigger the difference, the more attractive the activity is. The recommendations for the enabled activities are (A, 1000, 1125),(B, 1000, 1500) and (C, 1250, 1250), with the differences of -125, -500 and 0 respectively. Thus, the recommendation result is

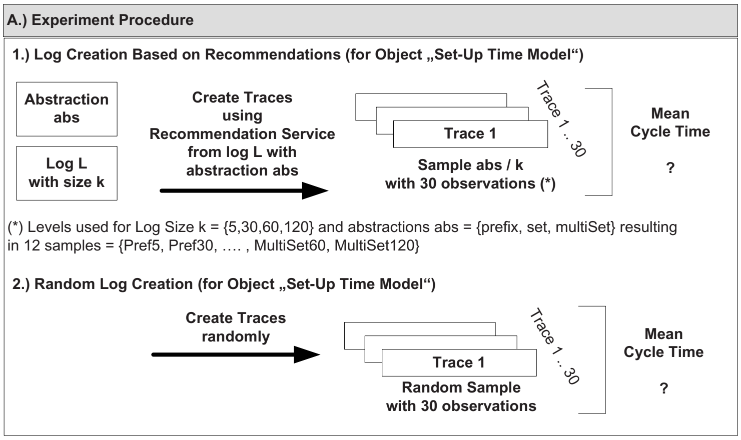

A.) Experiment Procedure
1.) Log Creation Based on Recommendations (for Object „Set-Up Time Model")
Abstraction abs
Log L with size k
Create Traces using Recommendation Service from log L with abstraction abs
Trace 1
Trace 1 .. 30
Sample abs / k with 30 observations (*)
Mean Cycle Time ?
(*) Levels used for Log Size k = {5,30,60,120} and abstractions abs = {prefix, set, multiSet} resulting in 12 samples = {Pref5, Pref30, …. , MultiSet60, MultiSet120}
2.) Random Log Creation (for Object „Set-Up Time Model")
Create Traces randomly
Trace 1
Trace 1 .. 30
Random Sample with 30 observations
Mean Cycle Time ?

Fig. 4. The experiment design

$\langle(B, 1000, 1500), (A, 1000, 1125), (C, 1250, 1250)\rangle$. If the user goal would be to maximize costs, the order will be reversed.

4 Evaluation Based on a Controlled Experiment

To evaluate the effectiveness of our recommendation service we conducted a controlled experiment. Section 4.1 describes the design underlying our experiment and Section 4.2 describes the preparatory steps we conducted. Section 4.3 explains the experiment procedure including data analysis. The results of our experiment are presented in Section 4.4. Factors threatening the validity of our experiment are discussed in 4.5.

In our experiment we use the recommendation service to support the business process, that has been explained in Section 2. The process has five activities (A, B, C, D, E) that have to be executed exactly once and can be executed in any order. Each activity has a cycle time of 10 time units, however, if C is directly executed after B, then the cycle time of the trace will be 35 time units of 50. For the experiment we assume that the user goal is to minimize the cycle time and that the recommendation service is used for support.

4.1 Experiment Design

This section describes the design underlying our experiment.

– **Object:** The object to be studied in our experiment are the traces created for the *set-up time model* with the help of our recommendation service.
– **Independent Variables:** In our experiment we consider the log abstraction and the log size as independent variables. For variable *log abstraction* we consider levels $abs \in \{prefix, set, multiset\}$ (cf Section 3.3). Variable *Log size k* represents the number of instances in the event log, i.e., the amount

of learning material based on which recommendations are made. As levels $k \in \{5, 30, 60, 120\}$ are considered.
- **Response Variable:** The response variable in our experiment is the cycle time of a trace created by the recommendation service using a log of a given size and a given abstraction.
- **Experiment Goal:** The main goal of our experiment is to investigate whether changes in the log significantly effect the cycle time[3] of the created traces given an abstraction. Another goal is to investigate whether the traces created by our recommendation service yield significantly better results than randomly created traces.

4.2 Experiment Preparation

This section describes the preparatory steps we conducted for the experiment.

- **Implementing the Recommendation Service in ProM.** As a preparation for our experiment we implemented the recommendation service described in Section 3 as a plug-in for the (Pro)cess (M)ining framework ProM[4]. ProM is a pluggable framework that provides a wide variety of plug-ins to extract information about a process from event logs [19], e.g., a process model, an organizational model or decision point information can be discovered. To implement the recommendation service we had to make several extensions to ProM as the recommendation service, in contrast to other plug-ins, is not a posteriori mining technique, but recommendations are provided in real-time during process execution. The implementation of our recommendation service is able to provide a process engine with recommendations on possible next steps knowing the enabled activities and the partial trace. In addition, the recommendation service provides means to add finished cases to the event log to make them available for recommendations in future executions.
- **Implementing a Log Creator and Log Simulator.** In addition to the recommendation service we implemented a log creator and log simulator. While the log creator allows us to randomly create logs of size k for a given process model, the log simulator can be used to create traces using the recommendation service with a log of size k and an abstraction *abs*. The log simulator takes the constraints imposed by the process model into consideration and ensures that no constraint violations can occur. Thus, the log simulator can be seen as a simulation of a process engine. Both the log creator and the log simulator have been implemented in Java using Fitnesse[5] as user interface. This allows us to configure our experiments in a fast and efficient way using a WIKI and to fully automate their execution.

[3] Note that our approach can also be used for costs, quality, utilization, etc. However, for simplicity we focus on the cycle time only.

[4] The ProM framework can be downloaded from `www.processmining.org`.

[5] Fitness Acceptance Testing Framework `fitnesse.org`

4.3 Experiment Execution and Data Analysis

The experiment procedure including the analysis of the collected data is described in this section.

- **Generation of Data.** As illustrated in Fig. 4 our experiment design comprises two independent variables (i.e., log abstraction *abs* and log size k). As a first step a log of size k is randomly created using the log creator, which is then - in a second step - taken by the log simulator as input to create traces for each combination of abstractions and log sizes. Traces are created based on the recommendations provided by the recommendation service described in Section 3. The recommendations given by the recommendation service are used throughout the entire execution of the case whereby the best recommendation (i.e., the one with the highest difference of *do* and *dont* values, see. Section 3.6) is taken. For each completed trace the log simulator records the cycle time. We repeated (n=30) the process of producing a log and creating a trace using the log simulator with this log as input. In total we obtained 12 samples covering all combinations of log size levels and abstraction levels. For example, sample `PREF5` represents the sample with $abs = prefix$ and $k = 5$.

  In addition to the 12 samples which are created using recommendations, we created one sample with 30 randomly created traces to compare this sample with the ones created using the recommendation service.
- **Effects of Changes in Log Size and Abstraction.** To analyse the effects of changes in the log size and the selection of a particular abstraction on the cycle time of the created traces we calculated 95% confidence intervals (CI 95%) on the mean cycle time for each sample. Doing so, we can say with 95% probability, for a given log size and a given abstraction, that the cycle time of a created trace will be within the calculated confidence interval.

  If we then compare the confidence intervals of two samples of a given abstraction (e.g., `PREF5` and `PREF10`) and these intervals do not overlap, we can assume that the two samples have statistically different cycle times.
- **Effectiveness of Abstractions.** To investigate whether the traces created by our recommendation service yield significantly better results than randomly created traces we compared the confidence interval of the random sample with the confidence intervals of each of the other 12 samples.

4.4 Experiment Results

The results of our experiments are summarized in Figures 5-10. Figures 5-7 depict the effect of the log size on the mean cycle time for the 12 samples created using recommendations. In Figures 8-10 we compare the different abstractions and the random strategy for a fixed log size.

- **Increasing the Log Size.** Figure 5 clearly shows the impact of increasing the log size on the cycle time for prefix abstraction. The mean cycle time

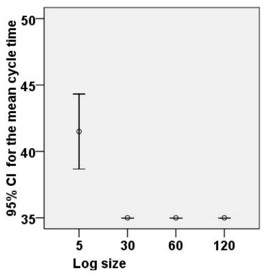


Fig. 5. Prefix

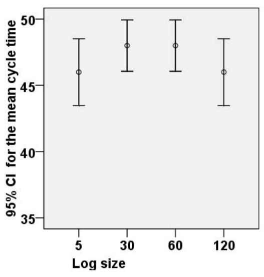


Fig. 6. Set

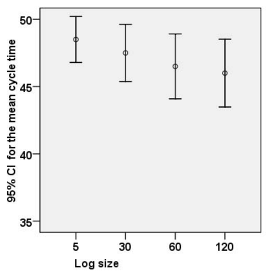


Fig. 7. Multi-set

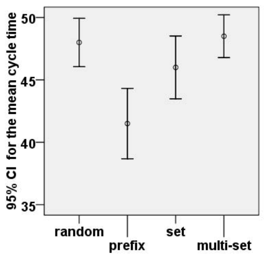


Fig. 8. Log size = 5

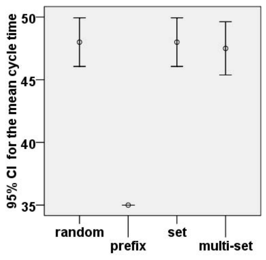


Fig. 9. Log size = 30

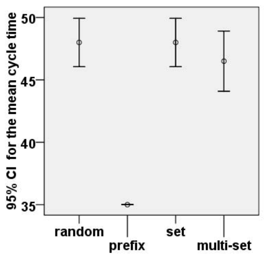


Fig. 10. Log size = 60

for sample PREF5 is given by a CI 95% [38.68,44.32] and for samples PREF30, PREF60 and PREF120 the mean cycle time is 35 (with a standard deviation of 0), which is also the optimum cycle time. When studying the results of Fig. 5 changing the log size from $k = 5$ to $k = 30$ yields a significant decrease of the cycle time. As the confidence intervals of samples PREF5 and PREF30 are not overlapping the difference in their cycle times is statistically significantly different. Further increases in the log size have no effect since the optimum cycle time has already been found for sample PREF30. Figure 6 and 7 shows the results for the set and multi-set abstraction. As all intervals are overlapping we can conclude that there is no significant improvement in the cycle time for the set and multi-set abstraction.

– **Comparing the Abstractions.** Figure 8-10 compares randomly created traces with the samples for prefix, set and multi-set. It can be observed that the prefix abstraction (i.e., samples PREF5, PREF30 and PREF60) has significantly better cycle times compared to the random sample and thus outperforms the random selection strategy. As illustrated in Figure 8-10 the difference between random selection and prefix abstraction becomes bigger with increasing log size. The prefix abstraction also outperforms the multi-set abstraction for all considered log sizes. A comparison of the set and prefix abstraction reveals that no significant differences exist between PREF5 and SET5, while PREF30 and SET30 as well as PREF60 and SET60 significantly differ. Finally, between the samples of the set and multi-set abstraction no significant differences can be observed.

In summary, our results show that an increase of the log size does effect the mean cycle time for the prefix abstraction and that this abstraction significantly outperforms the random selection strategy. For the set and multi-set abstraction changes in the log size do not significantly effect the mean cycle time. As these abstractions cannot exploit the order characteristics of the traces in the log they do not outperform the random selection strategy.

4.5 Risk Analysis

In the following we discuss factors potentially threatening the validity of our experiment. In general, it can be differentiated between threats to the internal validity (*Are the claims we made about our measurements correct?*) and threats to the external validity (*Can we generalize the claims we made?*) [10]. For our experiment most relevant threats affect the external validity:

– **Selection of Process Model.** In the selection of the business process for our experiment constitutes a threat to the external validity of our experiment. Given the properties of the chosen process model, a particular order of executing activities yields a benefit. As the prefix abstraction considers the exact ordering of activities, while the set and multi-set abstractions disregard this information, the chosen process model is favouring the prefix abstraction. For process models with different characteristics other abstractions might be more favourable. Therefore it cannot be generalized that the prefix abstraction is always better than the set and multi-set abstraction. A family of experiments using process models with different characteristics is needed for generalization. Initial investigations with a business process which is not order-oriented show that set and multi-set abstraction can perform significantly better than random selection.
– **Method of Log Creation.** For our experiment the method we used for log creation might constitute another threat to the external validity. We assume a log that only contains randomly created traces as the input for the log simulator. Using the simulator we then create, based on this log, an additional trace considering recommendations. In practice such an assumption might not always be realistic as a real-life log will most probably contain a mixture of randomly created traces and traces created using recommendations., i.e., by random/explorative and experience-based ways of working. First experiments indicate that the degree to which a log contains random traces compared to traces created based on recommendations also influences the cycle time. However, like for completely random logs an increase of the log size has led to decreases in the cycle time, but the slope of the decrease tends to be steeper for higher ratios of random behaviour in the log. An extensive investigation of logs with different ratios of random traces will be subject of further studies.

5 Related Work

The need for flexible PAISs has been recognized and several competing paradigms (e.g., adaptive process management [13, 22, 9], case-handling systems [16] and declarative processes [12]) have been proposed by both academia and industry (for an overview see [20]). In all these approaches the described trade-off between flexibility requiring user assistance can be observed.

Adaptive PAIS represent one of these paradigms by enabling users to make structural process changes to support both the handling of exceptions and the evolution of business processes. Related work in the context of adaptive PAISs addresses user support in exceptional situations. Both ProCycle [21,14] and CAKE2 [9] support users to conduct instance specific changes through change reuse. While their focus is on process changes, our recommendation service assists users in selecting among enabled activities. ProCycle and CAKE2 use case-based reasoning techniques to support change reuse. Therefore suggestions to the users are based on single experiences, (i.e., the most similar case from the past), while in our approach recommendations are based on the entire log.

In addition to adaptive process management technology, which allows for structural change of the process model, and the case-handling paradigm, which provides flexibility by focusing on the whole case, many approaches support flexibility by allowing the design of a process with regions (placeholders) whose contents is unknown at design-time and whose content is specified (Late Modeling) or selected (Late Binding) during execution of the tasks (for details see [20]). Examples of such approaches are, Worklets [2] or the Pockets of Flexibility [15] approach. Both approaches provide user assistance by providing simple support for the reuse of previously selected or defined strategies, recommendations as envisioned in our approach are not considered.

Besides the approaches described above there is a third paradigm for flexible workflows, which relies on a declarative way of modeling business processes. As opposed to imperative languages that "procedurally describe sequences of action", declarative languages "describe the dependency relationships between tasks" [6]. Generally, declarative languages propose modeling constraints that drive the model enactment [12]. When working with these systems, users have the freedom to choose between a variety of possibilities as long as they remain within the boundaries set by the constraints [11,12]. In the context of declarative workflows user assistance has not been addressed so far.

In [17] recommendations are used to select the step, which meets the performance goals of the process best. Like in our approach selection strategies (e.g., lowest cost, shortest remaining cycle time) are used. However, the recommendations are not based on a log, but on a product data model. [9,8,23,4] also address similarity measures, but unlike these approaches, our approach relies on observed behaviour rather than information derived from process models.

6 Conclusion

Existing PAISs are struggling to balance support and flexibility. Classical workflow systems provide process support by dictating the control-flow while groupware-like systems offer flexibility but provide hardly any process support. By using recommendations, we aim at offering support based on earlier experiences but not limit the user by imposing rigid control-flow structures. In this paper we presented an approach based on recommendations, i.e., based on a process model providing a lot of flexibility the set of possible activities is ranked based on *do* and *dont* values. The recommendation is based on (1) a configurable abstraction mechanism to compare the current partial case with earlier cases and (2) a target function (e.g., to minimize costs or cycle time). The whole approach has been implemented by extending ProM and can be combined with any PAIS that records events and offers work through worklists. The experimental results in this paper show the *value of information*, i.e., the more historic information is used, the better the quality of the recommendation. We experimented with different abstractions and log sizes. Clearly, the performance depends on the characteristics of the process and the abstraction. However, the experiments show that traces executed by support of recommendations often outperform traces executed without such support. This is illustrated by the difference in performance between the random selection and appropriate guided selection.

Future work will aim at characterizing the suitability of the various abstraction notions. Through a large number of real-life and simulated experiments we aim at providing insights into the expected performance of the recommendation service. Furthermore, we plan to extend our recommendation service such that in addition to control-flow information, information on data and resources is considered as well. Moreover, we plan to incorporate more sophisticated abstraction and comparison techniques, using available approaches from the field of data mining. In addition we will investigate the added value to create recommendations based on models that are extracted from the log, e.g., Markov Decision Processes.

Acknowledgements. The authors would like to thank Christian Günther, Maja Pesic, Hajo Reijers Isaac Corro Ramos and Christian Haisjackl for their help and contributions.

References

1. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
2. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Coopis 2006. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
3. Carlsen, S., Krogstie, J., Sølvberg, A., Lindland, O.I.: Evaluating Flexible Workflow Systems. In: Proc. HICSS-30 (1997)
4. Alves de Medeiros, A.K., van der Aalst, W.M.P., Weijters, A.J.M.M.: Quantifying process equivalence based on observed behavior. Data Knowl. Eng. 64(1), 55–74 (2008)
5. van Dongen, B.F., van der Aalst, W.M.P.: A meta model for process mining data. In: EMOI-INTEROP (2005)

6. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., Zbyslaw, A.: Freeflow: mediating between representation and action in workflow systems. In: Proc. CSCW 1996, pp. 190–198 (1996)
7. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., Teschke, M.: A Comprehensive Approach to Flexibility in Workflow Management Systems. In: Proc. WACC 1999, pp. 79–88. ACM Press, New York (1999)
8. Lu, R., Sadiq, S.: On the Discovery of Preferred Work Practice through Business Process Variants. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 165–180. Springer, Heidelberg (2007)
9. Minor, M., Tartakovski, A., Schmalen, D., Bergmann, R.: Agile Workflow Technology and Case-Based Change Reuse for Long-Term Processes. International Journal of Intelligent Information Technologies 1(4), 80–98 (2008)
10. Mutschler, B., Weber, B., Reichert, M.U.: Workflow management versus case handling: Results from a controlled software experiment. In: Proc. SAC 2008. ACM Press, New York (2008)
11. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes. In: Proc. DPM 2006, pp. 169–180 (2006)
12. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: CoopIS 2007 (2007)
13. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. JIIS 10(2), 93–129 (1998)
14. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating Process Learning and Process Evolution - A Semantics Based Approach. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 252–267. Springer, Heidelberg (2005)
15. Sadiq, S., Sadiq, W., Orlowska, M.: A Framework for Constraint Specification and Validation in Flexible Workflows. Information Systems 30(5), 349–378 (2005)
16. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data and Knowledge Engineering 53(2), 129–162 (2005)
17. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product based workflow support: A recommendation service for dynamic workflow execution. Technical Report BPM-08-03, BPMcenter.org (2008)
18. van der Aalst, W.M.P., Jablonski, S.: Dealing with Workflow Change: Identification of Issues an Solutions. Int'l Journal of Comp. Systems, Science and Engineering 15(5), 267–276 (2000)
19. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 27(2), 237–267 (2003)
20. Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data and Knowledge Engineering (accepted for publication)
21. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing Integrated Life Cycle Support in Process-Aware Information Systems. Int'l Journal of Cooperative Information Systems (IJCIS) (accepted for publication)
22. Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In: Proc. HICSS-34 (2001)
23. Wombacher, A., Rozie, M.: Evaluation of Workflow Similarity Measures in Service Discovery. In: Service Oriented Electronic Commerce, pp. 51–71 (2006)

Visual Support for Work Assignment in Process-Aware Information Systems

Massimiliano de Leoni[1], W.M.P. van der Aalst[2,3], and A.H.M. ter Hofstede[3]

[1] SAPIENZA - Università di Roma, Rome, Italy
deleoni@dis.uniroma1.it
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl
[3] Queensland University of Technology, Brisbane, Australia
a.terhofstede@qut.edu.au

Abstract. Process-aware information systems ranging from generic workflow systems to dedicated enterprise information systems use *work lists* to offer so-called *work items* to users. The work list handlers typically show a sorted list of work items comparable to the way that e-mails are presented in most e-mail programs. Since the work list handler is the dominant interface between the system and its users, it is worthwhile to provide a more advanced graphical interface that uses context information about work items and users. This paper uses the "map metaphor" to visualise work items and resources (e.g., users) in a sophisticated manner. Moreover, based on "distance notions" work items are visualised differently. For example, urgent work items of a type that suits the user are highlighted. The underlying map and distance notions may be of a geographical nature (e.g., a map of a city of office building), but may also be based on the process design, organisational structures, social networks, due dates, calenders, etc. The approach presented in this paper is supported by a visualisation framework implemented in the context of YAWL. The framework is set up in such a way that it can easily be combined with other workflow systems.

Keywords: Process-aware Information Systems, work list visualisation, YAWL.

1 Introduction

Originally, *Process-Aware Information Systems* (PAISs) [1] were mainly applied in the context of administrative processes. Later their application was extended to cross-organisational processes. Currently, PAISs are starting to be used for more flexible and/or pervasive processes, e.g., disaster management scenarios [2].

Independently on the application domain and underlying technology, a PAIS is driven by some process model. The model may be implicit or hidden, but the system supports the handling of cases in some (semi-)structured form. PAISs have also in common that they offer work to resources (typically people). The elementary pieces of work are called *work items*, e.g., "Approve travel request

XYZ1234". These work items are offered to the users by the so-called *work list handler.* This component takes care of work distribution and authorisation issues. Typically, PAISs use a so-called "pull mechanism", i.e., work is offered to all resources that qualify and the first resource to select it will be the only one executing it. To allow users to "pull the right work items in the right order", basic information is provided, e.g., task name, due date, etc. However, given the fact that the work list is the main interface of the PAIS with its users it seems important to provide support that goes beyond a sorted list of items. If work items are selected by less qualified users than necessary or if users select items in a non-optimal order, then the performance of the overall process is hampered. Assume the situation where multiple resources have overlapping roles and authorisations and that there are times where work is piling up (i.e., any normal business). In such a situation the questions listed below are relevant.

– "What is the most urgent work item I can perform?"
– "What work item is, geographically speaking, closest to me?"
– "Is there another resource that can perform this work item that is closer to it than me?"
– "Is it critical that I handle this work item or are there others that can also do this?"
– "How are the work items divided over the different departments?"

To our knowledge, commercial as well as open source PAISs present work lists simply as a list of work items each with a short textual description. Some products sort the work items in a work list using a certain priority scheme specified at design time and not updated at run time. To support the user in a better way and assist her in answering the above questions, we use *maps.* A map can be a geographical map (e.g., the map of a university's campus). But other maps can be used, e.g., process schema's, organisational diagrams, Gantt charts, etc. Work items can be visualised by dots on the map. By not fixing the type of map, but allowing this choice to be configurable, different types of relationships can be shown thus providing a deeper insight into the context of the work to be performed.

Work items are shown on maps. Moreover, for some maps also resources can be shown, e.g., the geographical position of a user. Besides the "map metaphor" we also use the "distance metaphor". Seen from the viewpoint of the user some work items are close while others are far away. This distance may be geographic, e.g., a field service engineer may be far away from a malfunctioning printer at the other side of the campus. However, many other distance metrics are possible. For example, one can support metrics capturing familiarity with certain types of work, levels of urgency, and organisational distance. It should be noted that *the choice of metric is orthogonal to the choice of map* thus providing a high degree of flexibility in context visualisation. Resources could for example opt to see a geographical map where work items, whose position is calculated based on a function supplied at design time, display their level of urgency.

This paper proposes different types of maps and distance metrics. Moreover, the framework has been implemented and integrated in YAWL.[1] YAWL is an open source workflow system based on the so-called workflow patterns. However, the framework and its implementation are set-up in such a way that it can easily be combined with other PAISs.

The paper is structured as follows. Section 2 discusses the state of the art in work list visualisation in PAISs, whereas Section 3 provides a detailed overview of the general framework. Section 4 focusses on the implementation of the framework, highlighting some design choices In Section 5 the framework is illustrated through a case study. Section 6 summarises the contributions of the paper and outlines avenues of future work aimed.

2 Related Work

Little work has been conducted in the field of work list visualisation. Visualisation techniques in the area of PAIS have predominantly been used to aid in the understanding of process schemas and their run time behaviour, e.g. through simulation [3] or process mining [4]. Although the value of business process visualisation is acknowledged, both in literature [5,6,7,8] and industry, little work has been done in the context of visualising work items.

The aforementioned body of work does not provide specific support for context-dependent work item selection. This is addressed though in the work by Brown and Paik [9], whose basic idea is close to the proposal of this paper. Images can be defined as maps and mappings can be specified between work items and these maps. Work items are visualised through the use of intuitive icons and the colour of work items changes according to their state. However, the approach chosen does not work so well in real-life scenarios where many work items may have the same position (especially in course-grained maps) as icons with the same position are placed alongside each other. This may lead to a situation where a map is completely obscured by its work items. In our approach, these items are coalesced in a single dot of which the size is proportionate to their number. By gradually zooming in on such a dot, the individual work items cam become visible again. In addition, in [9] there is no concept similar to our distance notion, which is an ingredient that can provide significant assistance with work item selection to resources. Finally, the work of Brown and Paik does not take the visualisation of the positions of resources into account.

Also related is the work presented in [10], where proximity of work items is considered without discussing their visualisation.

Most PAISs present work lists as a simple enumeration of their work items, their textual descriptions, and possibly information about their priority and/or their deadlines. This holds both for open source products, as e.g. jBPM[2] and

[1] www.yawlfoundation.org

[2] jBPM web site - `http://www.jboss.com/products/jbpm`

Together Workflow[3], as for commercial systems, such as SAP Netweaver[4] and Flower[5]. An exception is TIBCO's iProcess Suite[6] which provides a richer type of work list handler that partially addresses the problem of supporting resources with work item selection. For example, the work list handler can show the lengths of the work queues of other resources or position of work item on a geographic map at their location of execution. The iProcess Suite also supports a kind of look-head in the form of a list of "predicted" work items and their start times. One can also learn about projected deadline expirations and exception flows by expected durations specified at design time for the various tasks. Our visualisation framework is more accurate as it can take actual execution times of work items of a task into account through the use of log files when considering predictions for new work items of that task. Basically, the iProcess Suite provides support for some specific views (geographical position, deadline expiration) but these are isolated from each other. Our approach generalises over the type of maps (not just geographic) and unlike the iProcess Suite it is able to support multiple types of maps at the same time. The iProcess Suite is based on Google Maps while our framework does not rely on an external service for handling maps and positioning work items.

3 The General Framework

The proposed visualisation framework is based on a two-layer approach: (1) maps and (2) the visualisation of work items based on a distance notion. A work item is represented as a dot positioned along certain coordinates on a background map. A map is meant to capture a particular perspective of the context of the process. Since a work item can be associated with several perspectives, it can be visualised in several maps (at different positions). Maps can be designed as needed. When the use of a certain map is envisaged, the relationship between work items and their position on the map should be specified through a function determined at design time. Table 1 gives some examples of context views and the corresponding work item mapping.

Several active "views" can be supported whereby users can switch from one view to another. Resources can (optionally) see their own position on the map and work items are coloured according to the value of the applicable distance metric. Additionally, it may be helpful to show executing work items as well as the position of other resources. Naturally, these visualisations are governed by the authorisations that are in place.

Our framework assumes the generic lifecycle model as described in [11]. First, a work item is *created* indicating it is ready for distribution. The item is then

[3] Together Workflow web site - `http://www.together.at/together/prod/tws/`

[4] Netweaver web site - `http://www.sap.com/usa/platform/netweaver`

[5] Flower web site - `http://global.pallas-athena.com/products/bpmflower_product/`

[6] iProcess Suite web site - `http://www.tibco.com/software/business_process_management/`

Table 1. Examples of maps and mappings

Process context view	**Possible map and mapping**
The physical environment where tasks are going to be performed.	A real geographical map (e.g., Google maps). Work items are placed where they should be performed and resource are placed where they are located.
The process schema of the case that work items belong to.	The process schema is the map and work items are placed on top of tasks that they are an instance of.
Deadline expiration of work items.	The map is a time-line where the origin is the current time. Work items are placed on the time-line at the latest moment when they can start without their deadline expiring.
The organisation that is in charge of carrying out the process.	The map is an organizational chart. Work items are associated with the role required for their execution. Resources are also shown based on their organizational position.
The materials that are needed for carrying out work items.	The map is a multidimensional graph where the axes are the materials that are needed for work item execution. Let us assume that materials A and B are associated with axes x and y respectively. In this case, a work item is placed on coordinates (x, y) if it needs a quantity of x of material A and a quantity y of material B.
Costs versus benefits in executing work items.	In this case, the axes represent "Revenue" (the amount of money received for the performance of work items) and "Cost" (the expense of their execution). A work item is placed on coordinates (x, y) if the revenue of its execution is x and its cost is y. In this case one is best off executing work items close to the x axis and far from the origin.

offered to appropriate resources. A resource can commit to the execution of the item, after which it moves to the *allocated* state. The start of its execution leads it to the next state, *started*, after which it can successfully *complete*, it can be *suspended* (and subsequently *resumed*) or it can *fail* altogether. At run time a workflow engine informs the framework about the lifecyle states of work items.

3.1 Fundamentals

In this section the various notions used in our framework, e.g. work item and resource, are defined formally.

Definition 1 (Work item). *A work item w is a tuple (c, t, i, y, e, l), where:*

- *c is the identifier of the case that w belongs to.*
- *t is the identifier of the task of which w is an instance.*
- *i is a unique instance number.*
- *y is the timestamp capturing when w moved to the "offered" state.*
- *e is the (optional) deadline of w.*
- *l represents the (optional) GPS coordinates where w should be executed.*

Dimensions y and l may be *undefined* in case work item w is not yet offered or no specific execution location exists respectively. The e value concerns timers which may be defined in YAWL processes. A process region may be associated with a timer. When the timer expires, the work items part of the region are cancelled. Note that a work item can potentially be a part of more than one cancellation region. In these cases, e is assumed as the latest possible completion time with respect to every cancellation region the work item is part of.

Definition 2 (Resource). *A resource r is a pair (j, l), where:*

- *j is the identifier of the resource.*
- *l represents the (optional) GPS coordinates where the resource is currently located.*

The notation w_x is used to denote the projection on dimension x of work item w, while the notation r_y is used to denote the projection on dimension y of resource r. For example, w_t yields the task of which work item w is an instance. Work items w' and w'' are considered to be *siblings* iff $w'_t = w''_t$. The set *Coordinates* consists of all possible coordinates. Elements of this set will be used to identify various positions on a given map.

Definition 3 (Position function). *Let W and R be the set of work items and resources. Let M be the set of available maps. For each available map $m \in M$, there exists a function $position_m : W \cup R \nrightarrow Coordinates$ which returns the current coordinates for work items and available resources on map m.*

For a map $m \in M$, the function $position_m$ may be partial, since some elements of W and/or R may not have an associated position. Consider for example the case where a work item can be performed at any geographical location or where it does not really make sense to associate a resource with a position on a certain map. As the various attributes of work items and resources may vary over time it is important to see the class of functions $position_m$ as time dependent.

To formalise the notion of distance metric, a distance function is defined for each metric that yields the distance between a work item and a resource according to that metric.

Definition 4 (Distance function). *Let W and R be the set of work items and resources. Let D be the set of available distance metrics. For each distance metric $d \in D$, there exists a function $distance_d : W \times R \rightarrow [0, 1]$ that returns a number in the range [0,1] capturing the distance between work-item $w \in W$ and resource $r \in R$ with respect to metric d.*[7]

Given a certain metric d and a resource r, the next work item r should perform is a work item w for which the value $distance_d(w, r)$ is the closest to 1 among all offered work items.

3.2 Available Metrics

In Table 2 a number of general-purpose distance metrics are informally explained. These are all provided with the current implementation. Due to the limited space, we will provide more details for only one of these distance metrics. The metric chosen combines the familiarity of a resource with a certain work item and the familiarity of other resources that are able to execute that work item. In order to formalise this notion, two auxiliary functions are introduced.

[7] Please note the value 1 represents the minimum distance while 0 is the maximum.

Table 2. Distance Metrics currently provided by the implementation

Metric	**Returned Value**
$distance_{Familiarity}(w,r)$	How familiar is resource r with performing work item w. This can be measured through the number of sibling work items the resource has already performed.
$distance_{Geo\_Distance}(w,r)$	How close is resource r to work item w compared to the closest resource that was offered w. For the closest resource this distance is 1. In case w does not have a specific GPS location where it should be executed, this metric returns 1 for all resources.
$distance_{Popularity}(w,r)$	The ratio of logged-in resources having been offered w to all logged-in resources. This metric is independent from resource r making the request.
$distance_{Urgency}(w,r)$	The ratio between the current timestamp and the latest timestamp when work item w can start but is not likely to expire. The latter timestamp is obtained as the difference between w_e, the latest timestamp when w has to be finished without expiring, and w's estimated duration. This estimation is based on past execution of sibling work items of w by r.
$distance_{Past\_Execution(w,r)}$	How familiar is resource r with work item w compared to the familiarity of all other resources that w has been offered to. More information about this metric is provided in the text.

past_execution(w,r) yields the weighted mean of the past execution times of the last h-th work items performed by r among all work item siblings of w. In this context, the past execution time of work item w' is defined as the duration that elapsed between its assignment to r and its successful completion. Let $time_i(w,r)$ be the execution time of the i-th last work item among w's siblings performed by r, then:

$$past\_execution(w,r) = \frac{\sum_{i=1}^{j_{(w,r,h)}} \alpha^{i-1} \cdot time_i(w,r)}{\sum_{i=1}^{j_{(w,r,h)}} \alpha^{i-1}} \tag{1}$$

where constant $\alpha \in [0,1]$ and value $j_{(w,r,h)}$ is the minimum between a given constant h and the number of sibling work items of w performed by r. Both h and α have to be tuned through testing. If value $j_{(w,r,h)}$ is equal to zero, $past\_execution(w,r)$ is assumed to take an arbitrary large number.[8] The intuition behind this definition stems from the fact that more recent executions should be given more consideration and hence weighted more as they better reflect resources gaining experience in the execution of instances of a certain task.

Res(w) returns all currently logged-in resources that have been offered w:

$$Res(w) = \{r \in R \mid w \ is \ offered \ to \ r\}.$$

Using these auxiliary functions the following metric can be defined:

$$distance_{Relative\_Past\_Execution}(w,r) = \frac{1/past\_execution(w,r)}{\sum_{r' \in Res(w)} 1/past\_execution(w,r')}. \tag{2}$$

[8] Technically, we set it as the maximum floating value.

Again, space considerations prevent us from providing an in-depth explanation of this definition and instead, we just provide some intuition. First observe that if exactly one resource r exists capable of performing work item w, then the equation yields one. If n resources are available and they roughly have the same familiarity with performing work item w, then for each of them the distance will be about $1/n$. It is clear then that as n increases in value, the value of the distance metric approaches zero. If on the other hand many resources exist that are significantly more effective in performing w than a certain resource r, then the value of the denominator increases even more and the value of the metric for w and r will be closer to zero.

4 Implementation

The general framework described in the previous section has been operationalised through the development of a component that can be plugged into the YAWL system. The YAWL environment is an open source PAIS, based on the workflow patterns[9], using a service-oriented architecture. The YAWL engine and all other services (work list handler, web-service broker, exception handler, etc.) communicate through XML messages. The YAWL work list handler was developed as a web application. In its graphical interface different tabs are used to show the various queues (e.g. offered work items). The visualisation framework can be accessed through a newly introduced tab and is implemented as a Java Applet.

Section 4.1 illustrates some of the visualisation features provided by the implementation, whereas Section 4.2 focusses on how the component fits within the YAWL architecture.

4.1 The User Interface

The position and distance functions represent orthogonal concepts that require joint visualisation for every map. The position function for a map determines where work items and resources will be placed as dots, while the distance function will determine the colour of work items. Conceptually, work item information and resource information is split and represented in different layers. Users can choose which layers they wish to see and in case they choose both layers which of them should overlay the other.

Work-item Layer. Distances can be mapped to colours for work items through a function $\mathit{colour} : [0, 1] \rightarrow C$ which associates every metric value with a different colour in the set C. In our implementation colours range from white to red, with intermediate shades of yellow and orange. When a resource sees a red work item this could for example indicate that the item is very urgent, that it is one of those most familiar to this resource, or that it is the closest work item in terms of its geographical position. While the colour of a work item can depend on the resource viewing it, it can also depend on which state of the lifecycle it is in.

[9] www.workflowpatterns.com

Table 3. Visualisation of a work item depending on its state in the life cycle

Work item state	**Colour scheme used in the work-list handler**
Created	Work item is not shown.
Offered to single/multiple resource(s)	The colour is determined by the distance to the resource with respect to the chosen metric. The colour ranges from white through various shades of yellow and orange to red.
Allocated to a single resource	Purple.
Started	Black.
Suspended	The same as for offered.
Failed	Grey.
Completed	Work item is not shown.

Special colours are used to represent the various states of the work item lifecycle and Table 3 provides an overview. The various rows correspond to the various states and their visualisation. Resources can filter work items depending on the state of items. This is achieved through the provision of a checkbox for each of the states of Table 3. Several checkboxes can be ticked. There is an additional checkbox which allows resources to see work items that they cannot execute, but they are authorised to see.

Resources may be offered work items whose positions are the same or very close. In such cases their visualisations may overlap and they are grouped into a so-called "joint dot". The diameter of a joint dot is proportional to the number of work items involved. More precisely, the diameter D of a joint dot is determined by $D = d(1 + \lg n)$, where d is the standard diameter of a normal dot and n is the number of work items involved. Note that we use a logarithmic (lg) scaling for the relative size of a composite dot.

Combining several work items int a single dot raises the question of how the distance of this dot is determined. Four options are offered for defining the distance of a joint dot, one can take a) the maximum of all the distances of the work items involved, b) their minimum, c) their median, or d) their mean. When a resource clicks on a joint dot, all work items involved are enumerated in a list and they are coloured according to their value in terms of the distance metric chosen.

Resource Layer. When a resource clicks on a work item the positions of the other resources to whom this work item is offered are shown. Naturally this is governed by authorisation privileges and by the availability of location information for resources for the map involved.

Resource visualisation can be customised so that a resource can choose to see a) only herself, b) all resources, or c) all resources that can perform a certain work item. The latter option supports the case where a resource clicks on a work item and wishes to see the locations of the other resources that can do this work item.

4.2 Architectural Considerations

Figure 1 shows the overall architecture of the visualisation framework and the connections with other YAWL components. Specifically, the visualisation framework comprises:

The Visualisation Applet is the client-side applet that allows resources to access the visualisation framework and it resides as a separate tab in the work-list handler.

The Visualisation Designer is used by special administrators in order to define and update maps as well as to specify the position of work items on defined maps. Designers can define positions as fixed or as variable through the use of XQuery. In the latter case, an XQuery expression is defined that refers to case variables. This expression is evaluated at run time when required.

Services is the collective name for modules providing information used to depict maps and to place work items (e.g. URLs to locate map images, work item positions on various maps).

The *YAWL engine* is at the heart of the YAWL environment. It determines which work items are enabled and can thus be offered for execution and it handles the data that is involved. While the YAWL engine offers a number of external interfaces, for the visualisation component interfaces B and E are relevant. Interface B is used, for example, by the work list handler to learn about work items that need to be offered for execution. This interface can also be used for starting new cases. Interface E provides an abstraction mechanism to access log information, and can thus e.g. be used to learn about past executions of siblings of a work item. In particular one can learn how long a certain work item remained in a certain state.

The *work list handler* is used by resources to access their "to-do" list. The standard version of the work list handler provides queues containing work items in a specific state. This component provides interface G which allows other components to access information about the relationships between work items and resources. For example, which resources have been offered a certain work item

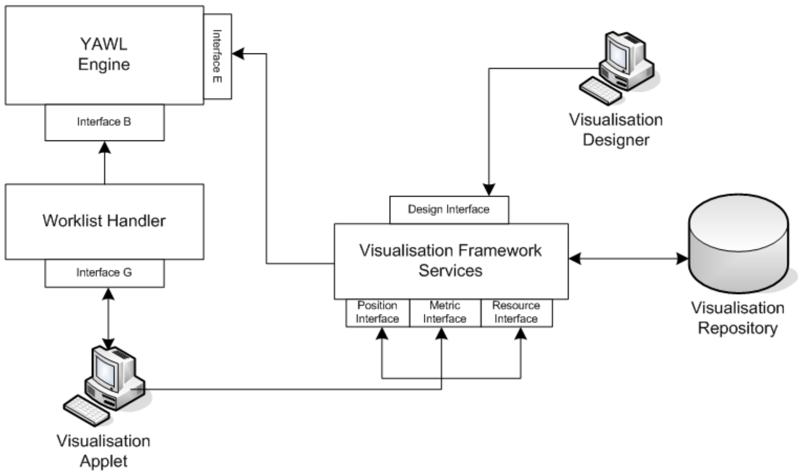


Fig. 1. Position of the visualisation components in the YAWL architecture

or which work items are in a certain state. Naturally this component is vital to the Visualisation Applet.

In addition to interface G, the Visualisation Applet also connects to the *Services* modules through the following interfaces:

The Position Interface provides information about maps and the positioning of work items on these maps. Specifically, it returns an XQuery over the YAWL net variables that the Visualisation Applet has to compute. The work list handler needs to be consulted to retrieve the current values of these variables.

The Metric Interface provides information about available metrics and their values for specific work item - resource combinations.

The Resource Interface is used to update and retrieve information concerning positions of active resources on maps.

The visualisation framework was integrated into the standard work list handler of YAWL through the addition of a JSP (Java Server Page).

All of the services of the visualisation framework share a repository, referred to as *Visualisation Repository* in Figure 1, which stores, among others, XQueries to compute positioning information, resource locations in various maps, and names and URLs of maps. Services periodically retrieve log data through Interface E in order to compute distance metric values for offered work items. For instance, to compute the metric *Relative Past Execution* (Equation 2) for a certain resource, one can see from Equation 1 that information is required about the h past executions of sibling work items performed by that resource.

To conclude this section, we would like to stress that the approach and implementation are highly generic, i.e., it is relatively easy to embed the visualisation framework in another PAIS.

5 Example: Emergency Management

In this section we are going to illustrate a number of features of the visualisation framework by considering a potential scenario from emergency management. This scenario stems from a user requirement analysis conducted in the context of a European-funded project [2]. Teams are sent to an area to make an assessment of the aftermath of an earthquake. Team members are equipped with a laptop and their work is coordinated through the use of a PAIS.

The main process of workflow for assessing buildings is named *Disaster Management*. The first task *Assess the affected area* represents a quick on-the-spot inspection to determine damage to buildings, monuments and objects. For each object identified as worthy of further examination an instance of the sub-process *Assess every sensible object* (of which we do not show the actual decomposition for space reasons) is started as part of which a questionnaire is filled in and photos are taken. This can be an iterative process as an evaluation is conducted to determine whether the questionnaire requires further refinement or more photos need to be taken. After these assessments have finished, the task *Send data to* *the headquarters* can start which involves the collection of all questionnaires and photos and their subsequent dispatch to headquarters. This information is used to determine whether these objects are in imminent danger of collapsing and if so, whether this can be prevented and how that can be achieved. Depending on this outcome a decision is made to destroy the object or to try and restore it.

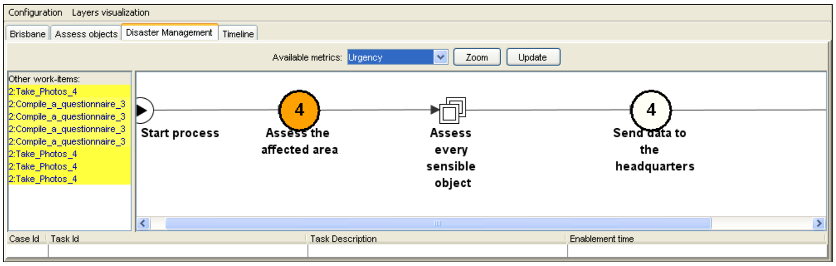
For the purposes of illustrating our framework we assume that an earthquake has occurred in the city of Brisbane. Hence a number of cases are started by instantiating the *Disaster Management* workflow described above.

Each case deals with the activities of an inspection teams in a specific zone. Figure 2 shows three maps. In each map, the dots refer to work items. Figure 2(a) shows the main process of the *Disaster Management* workflow, including eight work items. Dots for work items which are instances of tasks *Assess the affected area* and *Send data to the headquarter* are placed on top of these tasks in this figure. Figure 2(b) shows the decomposition of the *Assess every sensible object* sub-net. Here also eight work items are shown. No resources are shown in these diagrams. Note that on the left-hand side is shown a list of work items that are not on the map. For example, the eight work items shown in the map in Figure 2(a) appear in the list of *"other work items"* in Figure 2(b).

Figure 2(a) uses the urgency distance metric to highlight urgent cases while Figure 2(b) uses the familiarity metric to highlight cases closer to the user in terms of earlier experiences.

As another illustration consider Figure 2(c) where work items are positioned according to their deadlines. This can be an important view in the context of disaster management where saving minutes may save lives. In the map shown, the x-axis represents the time remaining before a work item expires, while the y-axis represents the case number of the case the work item belongs to. A work item is placed at location $(100 + 2 * \widetilde{x}, 10 + 4 * \widetilde{y})$ on that map, if $\widetilde{x}$ minutes are remaining to the deadline of the work item and its case number is $\widetilde{y}$. In this example, work items are coloured in accordance with the *popularity* distance metric.

Figures 3 and 4 show some screenshots of a geographical map of the city of Brisbane. Note that geographic maps are plain JPG images and have been obtained by capturing some screen shots from Google Maps. On these maps, work items are placed at the location where they should be executed. If their locations are so close that their corresponding dots overlap, a larger dot (i.e., a joint-dot) is used to represent the work items involved and the number inside corresponds to the number of these items. The green triangle is a representation of the resource whose work list is visualised here. Work items for tasks *Assess the affected area* and *Send data to the headquarters* are not shown on the map as they can be performed anywhere. In this example, dots are coloured according to the *familiarity* distance metric. A dot that is selected as focus obtains a blue colour and further information about the corresponding work item is shown at the bottom of the screen (as is the case for work item *Take_Photos_4* in Figure 3(b)).

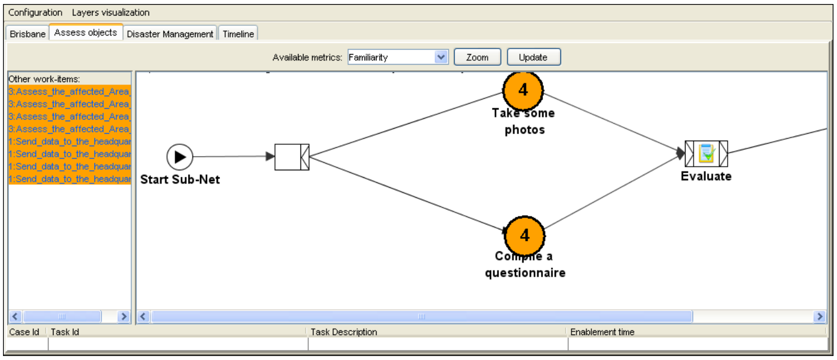


Available metrics: Urgency

Other work-items:
2:Take_Photos_4
2:Compile_a_questionnaire_3
2:Compile_a_questionnaire_3
2:Compile_a_questionnaire_3
2:Compile_a_questionnaire_3
2:Take_Photos_4
2:Take_Photos_4
2:Take_Photos_4

Case Id	Task Id	Task Description	Enablement time

(a) *Disaster Management* process map showing 4+4=8 work items

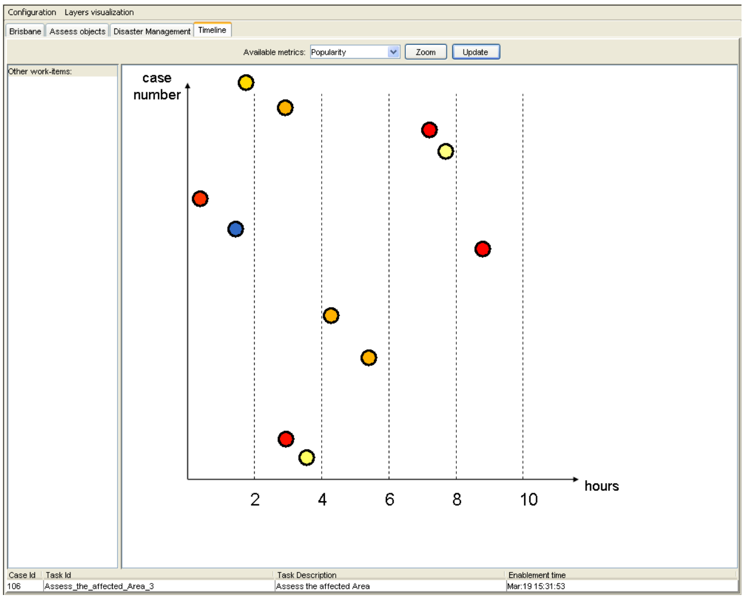


Available metrics: Familiarity

Other work-items:
3:Assess_the_affected_Area
3:Assess_the_affected_Area
3:Assess_the_affected_Area
3:Assess_the_affected_Area
1:Send_data_to_the_headquar
1:Send_data_to_the_headquar
1:Send_data_to_the_headquar
1:Send_data_to_the_headquar

Case Id	Task Id	Task Description	Enablement time

(b) *Assess the affected area* sub-net also showing 8 work items



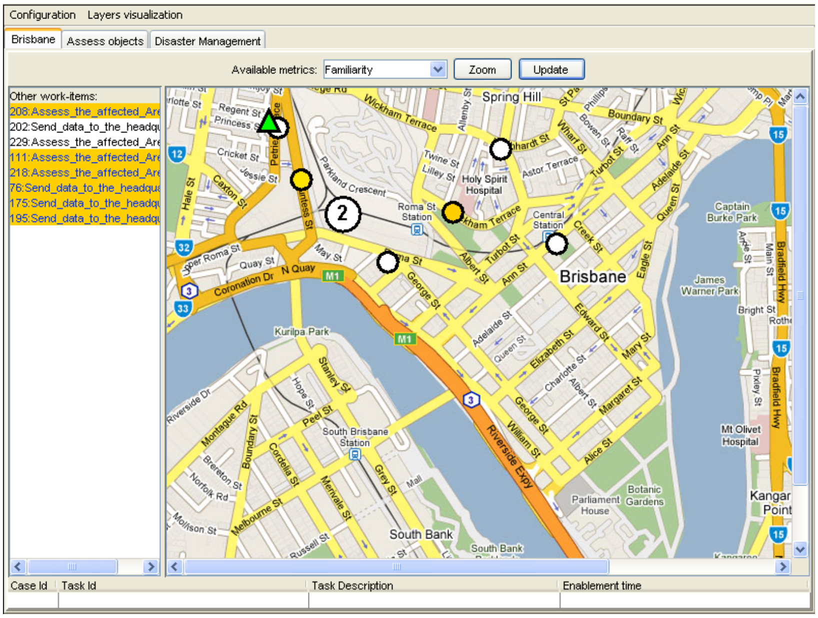
Available metrics: Popularity

Other work-items:

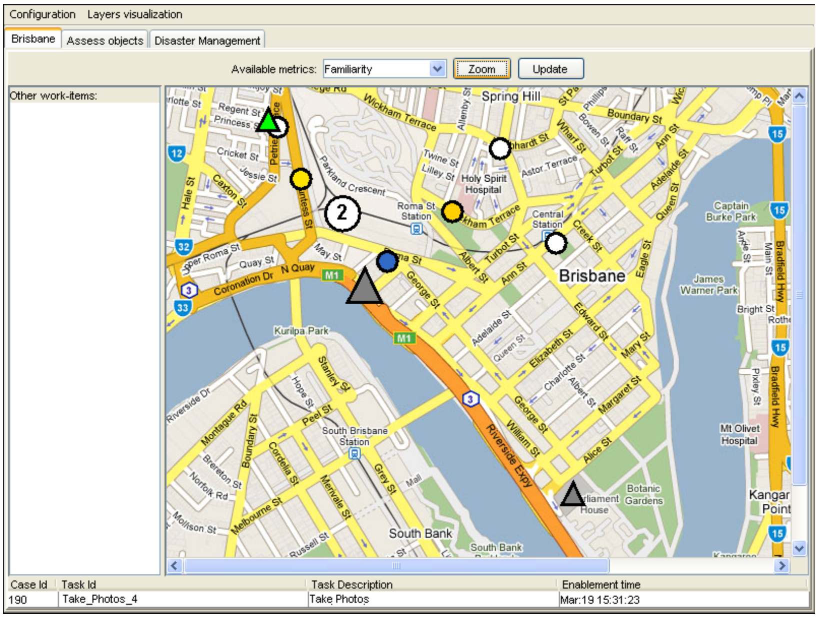
Case Id	Task Id	Task Description	Enablement time
106	Assess_the_affected_Area_3	Assess the affected Area	Mar:19 15:31:53

(c) Example of a timeline map for showing 11 work items

Fig. 2. Examples of Process and Timeline Maps for Disaster Management


Configuration
Layers visualization
Brisbane
Assess objects
Disaster Management
Available metrics: Familiarity
Zoom
Update
Other work-items:
208:Assess_the_affected_Are
202:Send_data_to_the_headqu
229:Assess_the_affected_Are
111:Assess_the_affected_Are
218:Assess_the_affected_Are
76:Send_data_to_the_headqu
175:Send_data_to_the_headqu
195:Send_data_to_the_headqu
2
Case Id
Task Id
Task Description
Enablement time

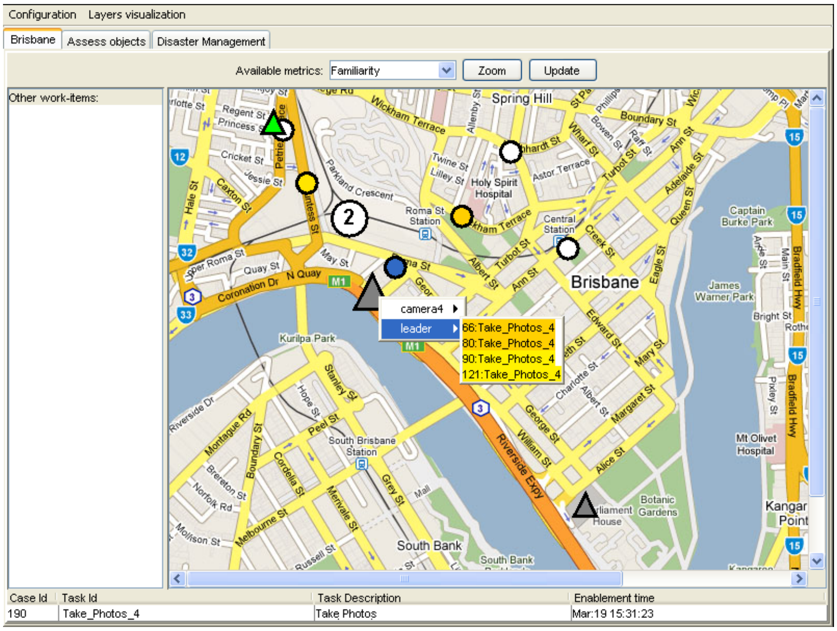
(a) Map showing the geographic locations of work items and resources: the triangle represents the resource and most work items are shown as single dots except for the two work items that are clustered into a single dot labeled "2"



Case Id	Task Id	Task Description	Enablement time
190	Take_Photos_4	Take Photos	Mar:19 15:31:23

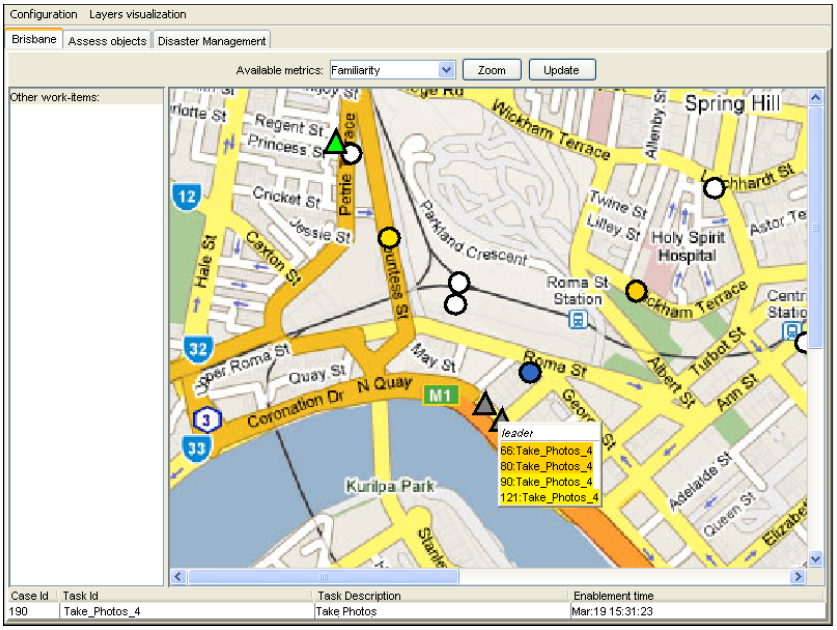
(b) Information about the selected dot (blue dot) is shown and also other resources are shown

Fig. 3. Examples of Geographic Maps for Disaster Management



Case Id	Task Id	Task Description	Enablement time
190	Take_Photos_4	Take Photos	Mar:19 15:31:23

(a) When a triangle is selected, the corresponding resources and offered work items are shown



Case Id	Task Id	Task Description	Enablement time
190	Take_Photos_4	Take Photos	Mar:19 15:31:23

(b) When zooming in, clustered work items and resources are separated

Fig. 4. Further examples of Geographic Maps for Disaster Management

One can click on a dot and see the positions of other resources that have been offered the corresponding work item. For example, by clicking on the dot representing work item *Take_Photos_4*, other resources, represented by triangles, are shown (see Figure 3(b)). As for work items, overlapping triangles representing resources are combined. For examples, the larger triangle shown in Figure 3(b) represents two resources.

Figure 4(a) shows the screen shot after clicking on the joint triangle. A resource can thus see the list of resources associated with this triangle. By selecting one of the resources shown in the list, the work items offered to that resource can be seen. The colour of these work items is determined by their value for the chosen distance metric. A zooming feature is also provided. Figure 4(b) shows the result of zooming in a bit further on the map of Figure 4(a). As can be seen, no dots nor any triangles are overlapping anymore.

6 Conclusions

In this paper a general visualisation framework is proposed that can aid users in selecting the "right" work item among a potentially large number of work items offered to them. The framework uses the "map metaphor" to show the locations of work items and resources. The "distance metaphor" is used to show which work items are "close" (e.g., urgent, similar to earlier work items, or geographically close). Both concepts are orthogonal and this provides a great deal of flexibility when it comes to presenting work to people. For example, one can choose a geographical map to display work items and resources and use a distance metric capturing urgency. The proposed framework was operationalised as a component of the YAWL environment. By using well-defined interfaces the component is generic so that in principle it could be exploited by other PAISs as well under the provision that they are sufficiently "open" and provide the required interface methods. The component is also highly configurable, e.g., it allows resources to choose how distances should be computed for dots representing a number of work items and provides customizable support for determining which resources should be visible. Our operationalisation does not rely on external services such as Google Maps for map visualisation support. Maps are just images on which dots representing work items are to be positioned. Hence our approach is not restricted to certain types of maps.

Finally, it should be pointed out that the implementation for the Visualisation Designer is still lacking. In the current evaluation, we manually updated the information stored in the Visualisation Repository by accessing tables in the DBMS. All other parts are fully operational.

Further research aims at connecting the current framework to geographical information systems and process mining tools like ProM [4]. Geographical information systems store data based on locations and process mining can be used to extract data from event logs and visualise this on maps, e.g., it is possible to make a "movie" showing the evolution of work items based on historic data.

Acknowledgements. The work was primarily conducted during a visit of Massimiliano de Leoni to the Business Process Management Group at Queensland University of Technology. His internship has been partly supported by the European Commission through the project FP6-2005-IST-5-034749 WORKPAD. The authors would like to thank Michael Adams for implementing certain required YAWL interfaces and for answering numerous questions. We also gratefully acknowledge Guy Redding for providing some code to connect to the YAWL interfaces.

References

1. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley, Chichester (2005)
2. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Vetere, G., Salvatore, B., Dustdar, S., Juszczyk, L., Manzoor, A., Truong, H.L.: Pervasive Software Environments for Supporting Disaster Responses. IEEE Internet Computing 12, 26–37 (2008)
3. Hansen, G.: Automated Business Process Reengineering: Using the Power of Visual Simulation Strategies to Improve Performance and Profit. Prentice-Hall, Englewood Cliffs (1997)
4. van der Aalst, W.M.P., van Dongen, B., Christian, G., Mans, R.S., Alva de Medeiros, A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: Prom 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
5. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
6. Luttighuis, P., Lankhorst, M., Wetering, R., Bal, R., Berg, H.: Visualising business processes. Computer Languages 27, 39–59 (2001)
7. Streit, A., Pham, B., Brown, R.: Visualization support for managing large business process specifications. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 205–219. Springer, Heidelberg (2005)
8. Wright, W.: Business Visualization Adds Value. IEEE Computer Graphics and Applications 18, 39 (1998)
9. Brown, R., Paik, H.Y.: Resource-centric worklist visualisation. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 94–111. Springer, Heidelberg (2005)
10. Kumar, A., van der Aalst, W.M.P., Verbeek, H.: Dynamic Work Distribution in Workflow Management Systems: How to Balance Quality and Performance? Journal of Management Information Systems 18, 157–193 (2002)
11. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005)

From Personal Task Management to End-User Driven Business Process Modeling

Todor Stoitsev[1], Stefan Scheidl[1], Felix Flentge[2], and Max Mühlhäuser[2]

[1] SAP Research, SAP AG, Bleichstr. 8, 64283 Darmstadt, Germany
[2] Telecooperation Group, Darmstadt University of Technology, 64283 Darmstadt, Germany
{todor.stoitsev, stefan.scheidl}@sap.com,
felix@tk.informatik.tu-darmstadt.de,
max@informatik.tu-darmstadt.de

Abstract. The need to involve business users in process modeling is largely perceived in the context of Business Process Management systems. This can facilitate the elaboration of consistent process models which are better turned to users' needs and organizational changes. Despite the variety of tools and notations, process modeling remains hardly accessible for business users, who lack advanced technical skills. This paper presents an integrated approach for end-user driven business process modeling which uses web service based activity tracking to generate weakly-structured process models by capturing data on personal task management. These models can be adapted and reused for ad-hoc process support or exported to formal workflows by delivering the business knowledge to process designers and software developers. Interconnection of ad-hoc and formal workflows results in enhanced process flexibility and allows complementation of formal workflows through deviations at runtime. The approach is validated through the Collaborative Task Management (CTM) prototype.

Keywords: business process modeling, process-enhanced groupware, end-user development, agile workflow, computer supported cooperative work.

1 Introduction

Effective Business Process Management (BPM) can bring competitive advantages to enterprises in the fast evolving global market. Often, the only ones to understand the matter and complexity of business processes are the end users of enterprise software, who execute them on a daily basis. The need to use the detailed process knowledge of end users during the implementation of BPM solutions in enterprises is clearly perceived and emerges in analyst reports e.g. as the need for "increased business collaboration in process modeling" [9]. It calls for bridging the process understanding of all stakeholders involved in a Workflow (Wf) project - the business users and the business technology staff, i.e. process designers and developers. As a result, standardized graphical notations such as the Business Process Modeling Notation (BPMN) [18] have emerged. Visual process modeling is enabled in a variety of enhanced BPM solutions. However, achieving process support that is better turned to users' needs and organizational changes by "letting end users do the tailoring" demands "both domain expertise and advanced skills in computer use" [17]. Therefore upfront process modeling remains inaccessible for business users, who have detailed domain knowledge but limited technical expertise.

This paper presents an integrated approach which overcomes the above limitation and enables end users to become informed participants in business process modeling. The approach is based on collaborative task management. It is implemented and validated through a process-enhanced groupware system - the Collaborative Task Manager (CTM) which provides enhanced End-User Development capabilities. End-User Development is defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artefact" [15]. In the presented paper a process model is considered as a software artifact, which can be adapted and enacted to support human-centric business processes. The major motivation behind the tool is to render appropriation of process models to the end users.

Section 2 provides an overview of related work on agile process support. In section 3 we present the approach for end-user driven process modeling. The basic components of the CTM prototype, implementing the approach, are presented in section 4. A validation of the approach based on a CTM case study is described in section 5. Section 6 provides conclusions and gives future research directions.

2 Related Work

The need to support knowledge-intensive business processes raises advanced flexibility expectations on Wf management systems [20]. Tailoring of task and process representations according to the individual point of view and interconnecting them towards the achievement of global enterprise goals emerges as a common strategy for realizing process agility. Riss et al. suggest the recognition and reuse of emerging "task patterns" and "process patterns" as alternative to static Wfs [19]. Holz et al. [11] present a further task-centric approach which enables proactive information delivery on tasks and instance-based task reuse. Ad-hoc task hierarchies are further used to bridge routine and ad-hoc work [5, 13]. The above approaches discuss agile process support but do not consider involving end users in formal process modeling and enabling process "tailoring as collaboration" [17] between business users, process designers and developers. Wf projects often suffer from inconsistencies, resulting e.g. from "projecting the sequence of an interview onto real work situations or by assuming logical dependencies which do not correspond with reality" [10]. We therefore suggest that enabling a seamless transition from underspecified to formal process definitions is important as it could enable the derivation of consistent, real-life compliant Wfs for rigidly recurring activities and shorten the Wf implementation lifecycle. This study presents an approach for involving business users in process modeling towards enhanced adaptability of BPM to users' needs and process changes.

We suggest that similarly to tailoring of software systems, process tailoring should be ensured through a "gentle slope of complexity" [16], where users with different business and Information Technology (IT) background are able to efficiently tailor reusable process definitions. Process mining approaches are capable of generating workflows from logged data on ad-hoc collaboration or events in formal systems [1]. However they do not allow users to tailor the emergent workflows at use time. The need for user-centric approaches arises, which ensure unobtrusiveness and in the same time enable “informed participation” of end users in business process composition by fostering “social creativity” [8] and allowing domain experts to proactively drive process optimization in enterprises.

Related literature reveals that user strategies for organizing daily activities are far from any process or case-definition context and mostly rely on common office tools such as email [4] or personal to-do lists [3]. Agostini et al. [2] cross the boundaries of the personal workspace and integrate to-do lists and email within email-based workflows. However, the authors do not discuss decoupling of Wfs from the system as explicit process models, and how such models can be exchanged, adapted and reused. As end users have different levels of technical expertise and attitudes towards maintaining process data, we suggest that it is important to consider possibilities for “seeding, evolutionary growth, and reseeding (SER)” [8] of user-defined process models for their iterative refinement and complementation.

Similarly to email-based Wfs, we suggest involving end users in process composition by leveraging their experience with standard tools for task management (to-do lists) and collaboration (email). In this respect, a “gentle slope of complexity” [16] for process tailoring can be provided by closely integrating the process definition in the actual user working environment and unfolding emergent processes behind the scenes in an unobtrusive, implicit manner. For achieving this we propose enabling enterprise-wide, collaborative “programming by example” [14] by implicitly reconciling data on personal task management of multiple process participants to end-to-end process execution examples. In our previous work [21] we have described a framework for light-weight composition of ad-hoc business processes. It generally enables end users to create hierarchical to-do lists by breaking down tasks into sub tasks. Tasks can be delegated over email, whereby the recipients can further break down the received tasks and delegate resulting (sub)tasks to other end users. Changes of individual tasks in the personal end users’ to-do lists are tracked over web services on a central server instance where task data is replicated in a tracking repository. Tracking of email exchange for task delegation integrates the personal to-do lists of different process participants to overall Task Delegation Graphs (TDG) on the server. TDGs represent weakly-structured process models that are captured as actual process execution examples and contain all task data including artifacts (attachments) and stakeholders’ information. TDGs enable informed participation of end users in process composition by providing a workflow-like overview of evolving collaborative tasks beyond the capabilities of common email and to-do lists.

The introduced framework enables SER of weakly-structured process models through extraction, adaptation and reuse of Task Patterns (TP) [19, 21]. In the following a TP is considered as a reusable task structure, comprising one task with its sub task hierarchy and the complete context information of the contained tasks like e.g. description, used resources, involved persons etc. TPs can be enacted to create a new process instance and execute it along the provided example flow. This flow can be altered by changing suggested task delegations or reusing referenced TP hierarchies. TP adaptation and reuse can result in evolution and complementation of captured process examples. This evolution is traced through task instance-based ancestor/descendant relationships [21]. TPs generally enable end users to establish best-practices and to trace best-practice deviations in different application cases.

In the presented paper we discuss an approach that involves end users in formal process modeling based on implicitly generated TDGs by bridging ad-hoc and formal Wf models towards increased "business collaboration in process modeling" [9].

3 Approach

The presented approach supports process formalization through transformation of user-defined TDGs to formal workflows based on the task change and evolution history. The resulting workflows are hence implicitly modeled by all process participants and can be extended by process designers or developers in a shared context, containing ad-hoc and formal process representations. This enables process "tailoring as collaboration" [17] between business users, process designers and developers. An overview of the process definition cycle is given in Figure 1.

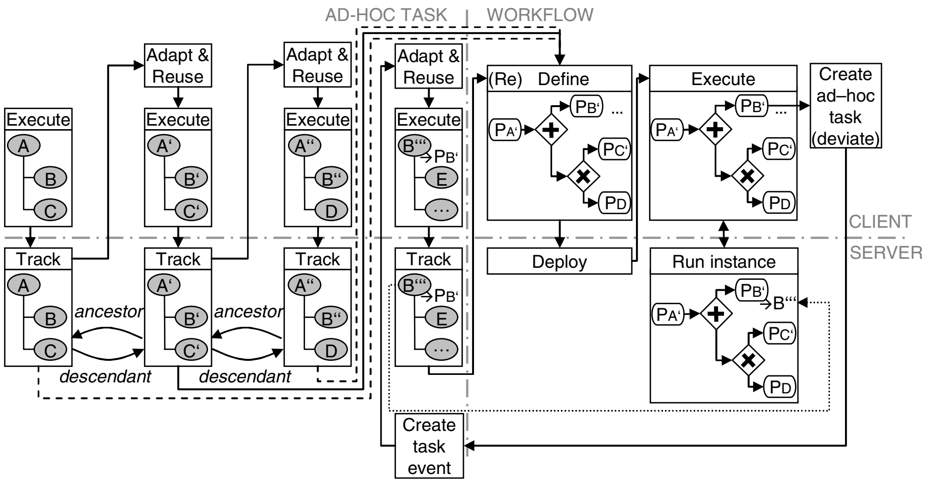


Fig. 1. Process definition cycle

A user is managing and executing tasks in a hierarchical to-do list in a task management client (root task *A* with sub tasks *B* and *C* in upper *CLIENT* layer). Task changes are tracked over web services to replicate task data in a tracking repository on a central server (lower hierarchy of task *A* in *SERVER* layer). Tracked tasks can be extracted, *adapted* and *reused* (root task *A'* with sub tasks *B'* and *C'*). Task instance-based *ancestor/descendant* relationships to the corresponding originating task are set iteratively for each task in the resulting hierarchy. Task reuse can result in different task variances, e.g. in task *A''* the expected task *C''* is replaced with task *D*.

When a process definition is triggered for given task (*A'*), the formal Wf is defined based on the complete evolution history, e.g. for task *A'* these are the *ancestor* and *descendant* task hierarchies respectively of *A* and *A''*, and task change history of related task instances. Task changes which alter task status, percent complete or task artifacts, are considered as task processing changes, denoting that the user is acting on a given task. Parallel flows in the resulting formal Wf are created for tasks, which have received task processing changes in parallel. For example if task *B'* has received a first task processing change in given time t_1 and a further task processing change at given time t_n, each task and each delegated task on the same tree level under the parent task of *B'* (such are *C'* and *D*) is considered parallel to *B'* if it has received a task processing change at a given time t_i such that $t_1 \leq t_i \leq t_n$. The period t_1 to t_n is referred to as the range of task *B'*.

Task ranges are a simplified way to suggest task sequencing. This is due to the fact that ad-hoc tasks can be executed without meeting any pre- or post-conditions. The resulting sequencing is hence based on suggestions and during model conversion, the user should be able to view the task change and evolution history and estimate whether the suggested flow is correct. SER can improve the accuracy of the generated workflows, i.e. if a given TP is reused multiple times and task ranges overlap in multiple executions, the tasks can be considered parallel with greater certainty. SER can enable also the modeling of alternative flows, i.e. based on substitution and cancellation of subsequent tasks in different TP application cases (in Figure 1 '+' denotes parallel and 'X' exclusive split).

The hierarchical order of tasks in TDGs is considered during model transformation by enabling different export modes for a task with subtasks: (i) as sub process, containing the sub tasks – this mode is pre-selected if a parent task contains data like e.g. attachments, detailed description etc., which is transferred to one or more of the sub tasks; (ii) as atomic task before the sub tasks' sequence – this mode is pre-selected if the parent task data is not transferred to any of the child tasks; (iii) as group element (e.g. BPMN group artifact – cf. [18]), embracing the sub tasks as logical association – this mode is pre-selected if the parent task contains only a subject.

Delegations in a TDG are considered during the model transformation as follows: (i) if a delegated task has no sub tasks on requester side it can be omitted, or preserved along with the recipient tasks in the resulting model. Omission is pre-selected as it results in model simplification when the task was fully processed by the recipients. In case of delegation to multiple recipients sub tasks of recipient tasks are handled as children of the same parent and checked for overlapping ranges (parallel execution). (ii) if a task was delegated, but the requester has added subtasks to it in their to-do list, requester and recipient tasks can be preserved as independent process nodes, or they can be merged by selecting one of them as the preferred, resulting Wf task. In the latter case requester and recipient sub tasks are handled as children of the same parent and checked for overlapping ranges.

Generated Wf tasks receive a reference to the originating ad-hoc task ($P_{A'}$, $P_{B'}$ etc.). A Wf is deployed on the server and executed through a Wf engine. During execution, users are enabled to deviate from a formal Wf by creating an ad-hoc task for a given Wf task. This issues an event over the server, creating an ad-hoc task in the to-do list of the respective delegate by additionally transferring the Wf task information to the resulting ad-hoc task, including a reference to the ad-hoc task, used for Wf task definition (for $P_{B'}$ this is *B'*). The recipient of the deviating ad-hoc task can adapt and reuse the original task (*B'*) (*ancestor*/*descendant* references for task *B'''* are not shown for simplicity). The resulting task (*B'''*) receives a reference to the deviated Wf task ($P_{B'}$), and the latter receives a reference to the deviating ad-hoc task (*B'''*) when it is tracked. This allows interrelation of the Wf task to ad-hoc task and vice-versa and navigation from the to-do list and TDG to the suspended Wf and from the Wf to the TDG of the ad-hoc task. The execution of the deviated Wf task can continue, i.e. if the deviation is an extension to the suspended Wf rather than an exception that requires Wf termination. While the ad-hoc task management server tracks the changes of the deviating ad-hoc task hierarchy (of *B'''*), the Wf server tracks the state of the deviated Wf task ($P_{B'}$ – started, suspended, ended). This allows evaluation whether a deviating ad-hoc task and the respective Wf task continue in parallel or the ad-hoc task is completed before the Wf task is processed further. After the Wf has ended, the Wf model can be *redefined* by considering the deviating ad-hoc flow in addition to the original ad-hoc task hierarchies, used for Wf definition.

4 Collaborative Task Manager (CTM)

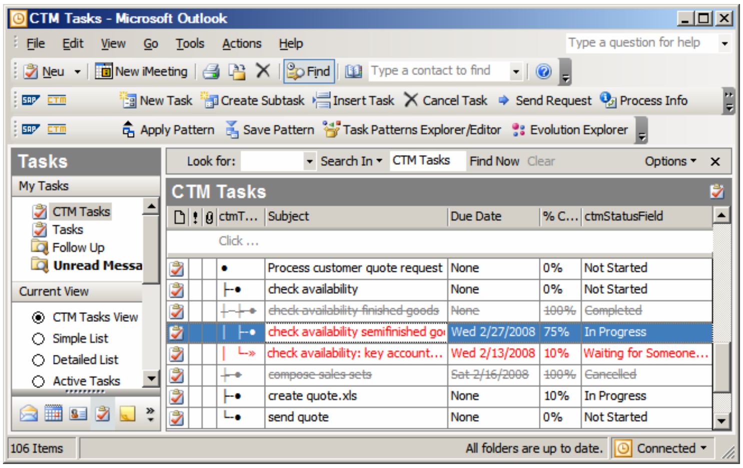
The presented approach is implemented and validated through the CTM prototype. CTM is a task management tool with enhanced End-User Development capabilities and addresses two main issues: (i) light-weight composition of weakly-structured process models for ad-hoc process support; (ii) formalization of weakly-structured process models for automation of rigidly recurring processes.

4.1 Programming by Example of Weakly-Structured Process Models

In order to ensure integrated support in a common user working environment, the CTM font-end is delivered as a Microsoft Outlook (OL) add-in. CTM extends OL mail and task items and enables "programming by example" by capturing OL events and using web services to replicate task data in a tracking repository, residing in a database on the CTM server. The CTM to-do list is shown in Figure 2. Extensions to the standard OL tasks enable end users to create hierarchical to-do lists. When the end user is creating or editing a CTM task they work with the familiar OL task fields. Files can be added to CTM tasks as common OL attachments.

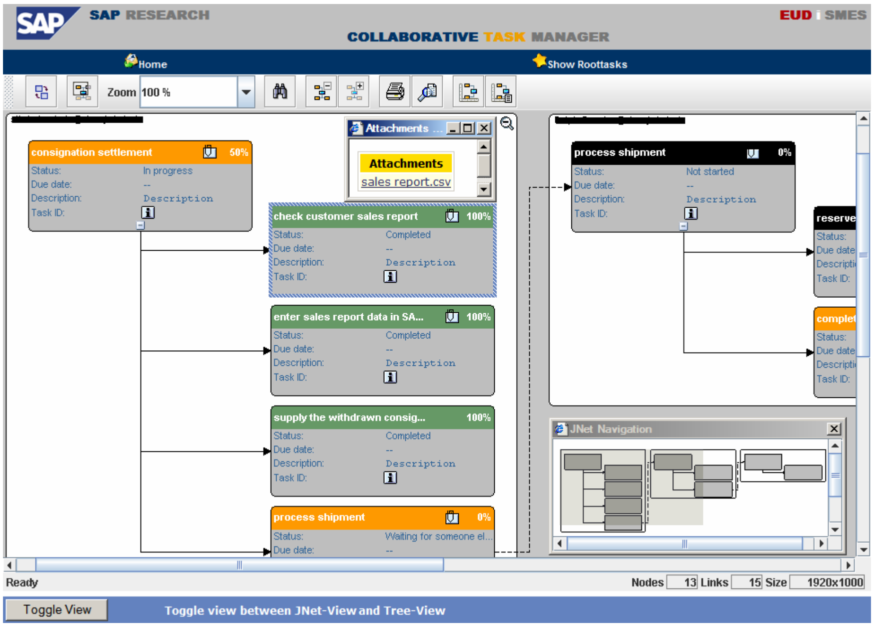
A CTM task is delegated through a "Request" email message, which recipients can "Accept", "Decline" (similarly to meeting requests in OL) or "Negotiate". The latter action allows iterative clarifications on tasks. When a request is accepted, and later on completed by a recipient, they issue a "Declare Complete" message, to which the requester can respond with "Approve Completion" or "Decline Completion". The actual discourse takes place in the email text, independently from the given message type. This allows open-ended collaboration and prevents from submitting user behavior to strict speech-act rules, which is a known limitation in speech-acts adoption [7]. All task-related email exchange is associated to a task dialog and stored on the server. Dialogs can be inspected through a process tree web overview, where the nodes provide links to task and email information including text and attachments.

CTM tracks the task-related email exchange and integrates the to-do lists of different process participants to a TDG [21] as shown in Figure 3, where individual tasks



ctmT…	Subject	Due Date	% C…	ctmStatusField
Click …				
•	Process customer quote request	None	0%	Not Started
├•	check availability	None	0%	Not Started
├┼•	~~check availability finished goods~~	~~None~~	~~100%~~	~~Completed~~
├•	check availability semifinished goo	Wed 2/27/2008	75%	In Progress
└»	check availability: key account…	Wed 2/13/2008	10%	Waiting for Someone…
┼•	~~compose sales sets~~	~~Sat 2/16/2008~~	~~100%~~	~~Cancelled~~
├•	create quote.xls	None	10%	In Progress
└•	send quote	None	0%	Not Started

Fig. 2. CTM to-do list



[redacted]

consignation settlement 50%
Status: In progress
Due date: --
Description: Description
Task ID:

check customer sales report 100%
Status: Completed
Due date: --
Description: Description
Task ID:

enter sales report data in SA... 100%
Status: Completed
Due date: --
Description: Description
Task ID:

supply the withdrawn consig... 100%
Status: Completed
Due date: --
Description: Description
Task ID:

process shipment 0%
Status: Waiting for someone el...
Due date: --

[redacted]

process shipment 0%
Status: Not started
Due date: --
Description: Description
Task ID:

Fig. 3. Task Delegation Graph (TDG)

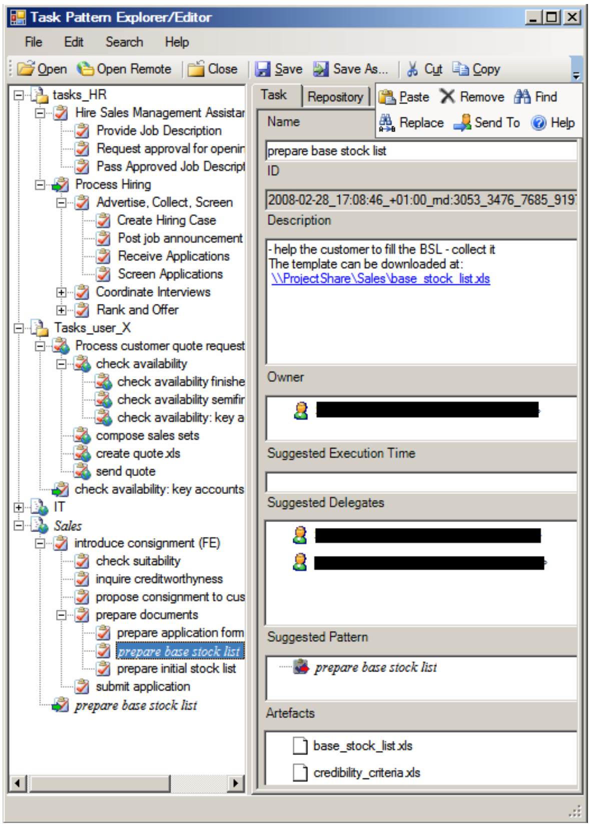
reside in different user containers (user data is blacked-out for privacy reasons in all figures in the paper). TDGs provide a workflow-like overview of collaborative activities where users can view status of related tasks, identify potential bottlenecks and evaluate work distribution. Currently, due date, task processing status and percent complete indications are provided. Attachments, added in OL tasks, are replicated in a central artifacts repository in a database on the CTM server, and are accessible in the task nodes. We focus on the composition and adaptation of process models by considering business users who can share information without extensive privacy requirements. Therefore no fine-grained authorization framework is currently provided. Such needs to be considered for CTM usage in a larger enterprise context.

4.2 SER of Weakly-Structured Process Models

CTM enables export of a local task from the personal to-do list to a single TP, and export of a complete TDG from the server to multiple TPs which represent the personal task hierarchies of different users and are interlinked through suggestions according to the delegation flow. TPs can be saved in local or remote TP repositories. A local TP repository is a XML document [21]. Remote TP repositories reside in a database on the CTM server. TPs are managed in the TP Explorer (see Figure 4), which provides rich editing and search functionality on task trees and on data in context fields on the right hand side, and allows also task search and extraction of TPs from the tracking repository. When editing the process execution examples (interlinked TPs) in this component "the user is not required to interact in the interface domain of computational abstraction, but works directly with the data that interests him or her" [15]. In that sense CTM enables editing through direct manipulation of the task fields. The "Name", "Description" and "Suggested Execution Time" fields hold simple task information in text format and are self-explanatory. The "Owner" field recommends expertise, i.e. when a task is extracted from an executed process the owner is the person, in whose to-do list the task was residing. The field "Suggested Delegates" contains information about the persons, who have the expertise to execute a given task, i.e. upon task extraction from collaborative process the task recipients are set in this field. The "Suggested Pattern" field holds a reference to a TP which should be used for the further processing of a task. In case of TDG extraction, such references in requester tasks point at recipient tasks, used for the further task processing. The recipient tasks are themselves extracted as separate TPs. Task attachments are represented as "Artifacts". Adding of custom artifacts in the TP Explorer replicates these to the artifacts repository.

TPs can be reused through an "Apply Pattern" operation in the to-do list. It opens the TP Explorer, where the user can search for TPs in TP repositories and in the tracking repository. Applying a TP reactivates the process example by generating the task hierarchy and filling the pre-modeled content information in the to-do list. Available delegates are suggested when delegation is initiated. Suggested TP references are also included in the resulting tasks and can be used by the person, activating the TP, to accomplish the task themselves without delegations. If a delegation is issued, the recipient task receives a reference to the suggested TP so that the recipient(s) can adapt and reuse it.

SER of TP through their iterative adaptation and reuse can result in refinement of captured process examples. CTM enables tracing of evolving TPs through task instance-based ancestor/descendant relationships [21]. Such are set iteratively between the tasks in the originating hierarchy and the corresponding tasks in the resulting hierarchy always, when a task hierarchy is reused, e.g. through copy/paste in the TP Explorer or save/apply pattern. Through navigating in evolution hierarchies, the user


Task Pattern Explorer/Editor
File Edit Search Help
Open Open Remote Close Save Save As... Cut Copy
Task Repository Paste Remove Find Replace Send To Help
tasks_HR
Hire Sales Management Assistar
Provide Job Description
Request approval for openir
Pass Approved Job Descript
Process Hiring
Advertise, Collect, Screen
Create Hiring Case
Post job announcement
Receive Applications
Screen Applications
Coordinate Interviews
Rank and Offer
Tasks_user_X
Process customer quote request
check availability
check availability finishe
check availability semifir
check availability: key a
compose sales sets
create quote.xls
send quote
check availability: key accounts
IT
Sales
introduce consignment (FE)
check suitability
inquire creditworthyness
propose consignment to cus
prepare documents
prepare application form
prepare base stock list
prepare initial stock list
submit application
prepare base stock list

Name
prepare base stock list

ID
2008-02-28_17:08:46_+01:00_md:3053_3476_7685_919

Description
- help the customer to fill the BSL - collect it
The template can be downloaded at:
\\ProjectShare\Sales\base_stock_list.xls

Owner
[redacted]

Suggested Execution Time

Suggested Delegates
[redacted]
[redacted]

Suggested Pattern
prepare base stock list

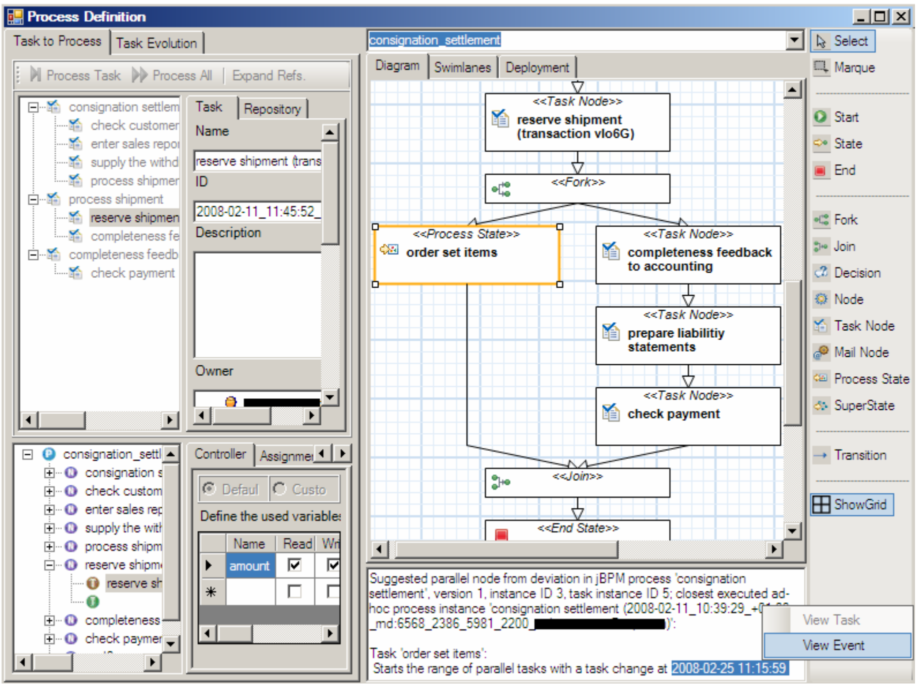
Artefacts
base_stock_list.xls
credibility_criteria.xls

Fig. 4. Task Pattern Explorer/Editor

can view the TDGs and dialog flows of tracked ancestors/descendants. Task evolution can be viewed in an Evolution Explorer in the CTM OL add-in.

4.3 From Email and To-Do to Formal Workflows

In CTM, rigidly recurring process fragments can be detected based on the captured TP evolution resulting from SER. For process formalization CTM uses the JBoss Business Process Management (jBPM) solution [12]. jBPM Wfs are modeled in a graph-oriented, visual language – the jBPM Process Definition Language (JPDL). The Wfs can be deployed and executed on a JBoss server, where they are accessed over a web front-end. jBPM process modeling is originally performed in a JPDL designer, provided as an Eclipse plug-in. However, CTM enables transformation of user-defined TDG to formal JPDL Wfs in the CTM OL add-in, by bridging ad-hoc and formal process representations. We should stress here, that TDGs result from ad-hoc user behavior which is not constrained through formal business rules. Therefore, the process expert performing the transformation has to ensure that inconsistencies in the TDGs will not impact on the quality of the formal models. The added value from



Name: reserve shipment (trans
ID: 2008-02-11_11:45:52_
Description:
Owner: [redacted]

Define the used variables

Name	Read	Wr
amount	✓	✓
*	☐	☐

Suggested parallel node from deviation in jBPM process 'consignation settlement', version 1, instance ID 3, task instance ID 5; closest executed ad-hoc process instance 'consignation settlement (2008-02-11_10:39:29_+0
_md:6568_2386_5981_2200_[redacted])':

Task 'order set items':
Starts the range of parallel tasks with a task change at 2008-02-25 11:15:59

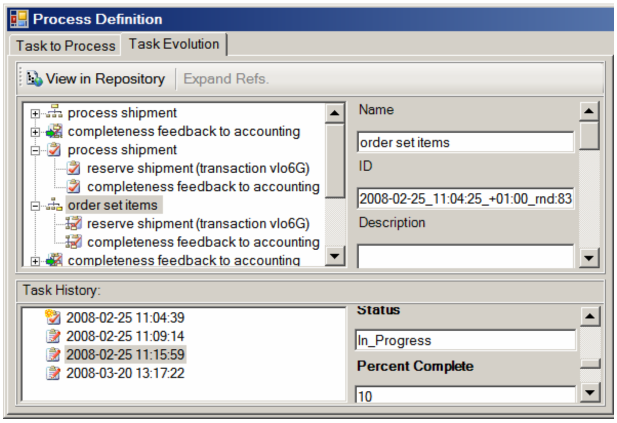
Fig. 5. CTM process definition environment

the introduced approach is that the expert is able to work with data, which was implicitly defined by the business users during their daily activities. The degree, to which the generated formal Wf models will need to be corrected or complemented, depends on how the end users are dealing with ad-hoc CTM tasks.

CTM Process Definition Environment. The CTM process definition environment is shown in Figure 5. The upper left corner contains a view, displaying the task hierarchy in the same manner as the TP Explorer. Processed tasks receive the jBPM task icon and a gray foreground. Tasks can be processed along the hierarchy through the 'Process Task' (stepwise) and 'Process All' (iteration) buttons. During task processing the appropriate export modes (cf. 2) for tasks with sub tasks and delegated tasks are provided in additional dialogs. A jBPM super state is used as a group element. The generated JPDL graph is displayed in the upper, central view in Figure 5. A toolbox on the right hand side allows advanced users to select appropriate tools and edit the model. If multiple (sub)processes are exported, the user can switch between them in the drop-down list in the upper central part of Figure 5. The tree in the lower left part contains the generated jBPM process entities (nodes and transitions). A tab control for setting their properties is provided on the right. The 'Controller' tab enables users to set parameters for task nodes. An 'Assignment' tab allows setting of jBPM task assignments such as e.g. swimlanes. The latter are automatically generated based on task owner information where each swimlane is defined through an expression 'user(email_address)' (swimlanes can be edited in a dedicated 'Swimlanes' tab - see upper central part of Figure 5). The task properties tab control further contains a ‘Form’ tab, where advanced users can edit the xhtml code of a jBPM task’s web form. CTM automatically generates this code by additionally embedding links to the original TDG and used artifacts (available in the artifacts repository) of ad-hoc tasks, and controls for creating an ad-hoc task for deviation from a jBPM workflow and for accessing the to-do list of such a task.

A textual explanation of the relevant transformations for each task is given in the lower central part of Figure 5. It describes the overlapping ranges and refers to the appropriate change events. Task change and evolution history is provided in the ‘Task Evolution’ tab, shown in Figure 6. The task evolution tree in the upper left part contains on root level the task ancestors and their references resulting from delegations, followed by the currently processed task and task descendants if available. The TDG of tracked ancestors/descendants can be viewed through the “View in Repository” button. Task change history is displayed in the lower tree. Changes are given with their time of occurrence and changed properties on the right.

Generated jBPM Wfs can be saved as process files or deployed as fully-functional Wfs on the jBPM server. Both functionalities are provided in the ‘Deployment’ tab in the upper central part of Figure 5. Process files can be copied in the JPDL designer, where the Wfs can be extended by developers.



process shipment
completeness feedback to accounting
process shipment
reserve shipment (transaction vlo6G)
completeness feedback to accounting
order set items
reserve shipment (transaction vlo6G)
completeness feedback to accounting
completeness feedback to accounting

Name: order set items
ID: 2008-02-25_11:04:25_+01:00_rnd:83
Description:

2008-02-25 11:04:39
2008-02-25 11:09:14
2008-02-25 11:15:59
2008-03-20 13:17:22

Status: In_Progress
Percent Complete: 10

Fig. 6. Task evolution and change history

Workflow Execution and Deviations. Deployed jBPM Wfs can be executed through the jBPM engine. Process instances are started and monitored through the jBPM web front-end. Wf tasks, generated in CTM, contain in their web forms additional buttons for creating and accessing ad-hoc tasks. Creating an ad-hoc task opens a web form, where the user can provide recipient information (email address), task subject and description. When the form is submitted, this information is sent to the CTM server along with the process instance and task IDs of the deviated jBPM task. The server issues a “create task” event to the CTM client of the specified recipient, which creates a new CTM task in their to-do list. CTM uses web service events, for which each OL client subscribes on the server on OL startup. Identification for sending events to a particular client is based on the user email address provided upon subscription. When the created ad-hoc task is tracked, the jBPM process instance and task ID are used to map the resulting ad-hoc task to the deviated Wf task on the server. The TDG of the created ad-hoc task can be opened from the Wf task's form in the jBPM front-end and vice-versa. After a process is completed, the Wf can be redefined by considering the deviating ad-hoc task hierarchy along with the original hierarchies for Wf definition.

5 Case Study

Setting and Extent of Use. The CTM case study was conducted at a manufacturing company (150 employees) and involved 6 users: COA - Chief Officer Assistant; CSO - Chief Sales Officer; SL1 & SL2 - Sales Employees; ITL - IT Department Lead; ITE - IT Employee. ITL and ITE were dealing with computers at an advanced level but did not have any programming skills and hence matched the type of end-user tailors. The other participants were typical business users. All users used OL as email client. CSO, SL1 and ITL also used OL tasks before the CTM installation. The trial was initiated with a workshop in which we gave a 1 hour presentation on the tool, followed by 30 minutes individual training of each user on the basic functionalities. Detailed user guides were provided to all participants. The jBPM export functionality was not included in the installations and manuals to preserve the focus on informal process support, addressing equally IT and business users. The trial lasted 8 weeks. Daily backups of the CTM database were scheduled and collected for evaluation each week. The evaluation concluded with a short video recording and transcription of the tool use, followed by a structured debriefing interview, in which we asked each participant to assess the basic features and rate to what extent CTM improved their ability to manage work using Likert scales and freeform explanations.

In a second iteration with SL1, SL2 and CSO we additionally performed an exercise for execution and refinement of a recurring process. The process was for settlement of consignment sales and occurred twice in the database backups. As consignment sales reports were sent in the end of each week and consignations were settled each Monday, the process seemed very appropriate for automation. We generated a jBPM Wf from a captured TDG and organized a workshop with the involved users. The workshop started with a 40 minutes tutorial on the jBPM web front-end where we explained to the users how deviations can be handled through creation of ad-hoc CTM tasks. Then we asked the users to process a weekly consignment settlement for a customer by maintaining the tasks in the jBPM Wf and deviating where needed. We used think-aloud and contextual inquiry [6] methods to track their strategies and intents. The exercises were videotaped for analysis.

Findings - Ad-Hoc Process Support. An excerpt from the case study metrics is given in Table 1. All participants reported that creating CTM tasks did not impede their work. We observed that users generally manage percent complete and status information, however not as precise estimation of work completion, but moreover *"to indicate that I'm working on it [a task] and avoid getting calls and emails from the others [sales], asking about status"* (ITE). We further encountered that users maintained attachments in CTM tasks, which was considered *"faster than email, as I only* *needed to attach the updated document and the others can pull the latest version [from the TDG]"* (SL1). As CTM was used only by a small group of people, privacy issues were not raised during the trial. However ITL stated that authorization has to be considered for extended CTM use in the enterprise by providing the possibility to hide certain process fragments in black-box containers in the TDG overview. The users further considered that having *"a kind of checklist [TP] with all things I need to do and the documents I need is very useful … especially if she [CSO] is not in the office [vacation]"* (SL2). The overall attitude was that global TP should be delivered by a (senior) domain expert, who can handle also the responsibility for providing them. Due to the restricted CTM usage, it was not possible to distribute TPs throughout the company, which prevented from developing a global strategy for TP management e.g. as alternative to text-based documents. Eventually, 2 remote TP were finally available (from ITL & CSO) whereas SL2 and ITE had developed local TPs.

Table 1. Excerpt of case study metrics

Metric	N
Created root tasks (ad-hoc processes)	8
Created tasks (overall)	46
Delegations	14
Unique attachments added	25
Attachment changes (diff. checksum, same name)	12
Percent complete changes	45
Task changes overall (only edit, no create/delete)	68
Created remote TP	2
Created local TP (files on user PCs)	4
Reused remote TP	1
Reused local TP	2

Findings - Formal Wf Definition and Refinement. A captured TDG of a process for settlement of consignment sales is shown in Figure 3 (task names are freely translated by the authors from German, customer name is removed for privacy reasons). SL1 receives a consignment sales report from a customer per email. The report is a CSV (Comma Separated Values) file, describing customer data, such as e.g. International Location Number (ILN), address etc., and consignment sales balance. SL1 "checks the report for consistency" as wrong input data like ILN can cause errors in the further processing. After that she "enters the sales report data in SAP R/3" system by copying the report in a special folder, from where the file is automatically read into the system. SL1 then describes the "supply for the withdrawn consignment items" in R/3 by specifying e.g. type and number of items. Then she asks SL2 to "process the shipment". SL2 "reserves the amount for shipment" in another transaction in R/3 and sends a "feedback about the completeness" of the settlement to the CSO for accounting purposes. CSO receives the feedback and later on "checks the payment" for the re-supplied goods. We generated a jBPM Wf from the captured TDG, which contained the above tasks in a strictly sequential order. We then asked SL1, SL2 and CSO to process a weekly consignation settlement for a customer by maintaining the corresponding tasks in the jBPM Wf and deviating where necessary.

After SL1 transferred the data from the customer sales report to the R/3 system, she cross-checked the resulting invoiced amount in the system with the amount in the sales report. There was a slight difference in both sums: *"Yes, sometimes the reported customer prices differ from our company prices … this is mostly due to the different calculation of taxes as customer calculates per delivery and we per item"* (SL1). The differences were minimal and were considered insignificant: *"Well, as in this case it is usually a matter of cents … we continue the settlement with the customer prices and ask Mrs. … [COA] to contact the customer and request them to correct the prices for the next settlement."* (SL1). As a result SL1 deviated from the currently started jBPM task "enter sales report data in SAP R/3", and created a CTM task in her to-do list with the same name. She then created a sub task "cross-check invoiced amount" and to this subtask she added another subtask "ask customer for correction", which she delegated to COA. As the process could in this case continue (with customer prices), SL1 returned back to the deviated jBPM task and completed it. She then completed the "supply for the withdrawn consignment items" task without deviations.

When SL2 started the "reserve amount for shipment" task he inspected the data about previous deliveries in R/3 and the reported amount of sold items in the customer sales report. For one of the consignment items he noticed that the reported sales exceeded the previously delivered amount: *"We ship this item per store and I assume that the customer has transferred items between their stores, without notifying us. … I'll need to inform Mrs. … [CSO] so that she can issue liability statements for the excess"* (SL2). SL2 considered that such inconsistencies will be propagated with the "completeness feedback" to CSO, so he entered a comment in the jBPM Wf, explaining the inconsistency. A further consignment item needed to be shipped as a set of multiple, smaller items. In the concrete case, items from the set were not delivered to the customer in the required amount and had to be re-supplied additionally: *"Sets are often requested with different content from different customers … we have to adapt and deliver the set items on demand."* (SL2). SL2 hence deviated from the started "reserve amount for shipment" task in the Wf and created an ad-hoc task "order set items" in his to-do list: *"This is actually the same shipment procedure as for the other items … We just process such set item deliveries independently as a special case."* (SL2). He then reserved the shipment of the currently handled consignment items, leaving the set items for later, and returned to the deviated jBPM task to complete it, so that CSO can handle further the consignation settlement. SL2 then started processing the order of the set items.

When handling the "completeness feedback" task in the jBPM Wf, CSO read the comment of SL2 about the inconsistency in delivered and sold consignment items: *"I need to create liability statements for that [inconsistency] so that the customer can correct the problem on their side"* (CSO). For that CSO created an ad-hoc task "prepare liability statement" in the to-do list and started preparing the document. When she was ready later on, she returned to the jBPM Wf and completed the active "completeness feedback" task. For the missing set items, she later on received a delegated CTM task "completeness feedback" from SL2, who had reserved the shipment for these items. We were not able to follow the processing of the "check of payments" task of CSO as this required customer actions. But CSO agreed that this would end the consignation settlement process and completed the task in the Wf.

Finally, we re-generated the jBPM process model with all available data from the initial TDG and from the execution of the jBPM Wf with deviating tasks, i.e. under the supervision of SL1, SL2 and CSO, with who we discussed the export modes of ad-hoc tasks (cf. 2). The "order set items" was exported as parallel sub process whereas the other deviations were exported as sequential Wf tasks. A screenshot, showing a part of the final model is given in Figure 5. Users appreciated having the complete Wf with all possible deviations in it: *"If the reported balance is ok, I'll just complete this task [liability statement] straight away … but I certainly want to have it there to make sure I won't forget it"* (CSO). Users highly appreciated the provided jBPM Wf functionality as the automated task assignment would save them the effort to distribute tasks per email as usual. They further reported that they consider the final Wf real-life compliant and will try to use it on regular basis and possibly to develop several variations for different customers.

6 Conclusions

The paper presents an integrated approach enabling informed participation of end users in business process composition by using collaborative "programming by example" based on personal task management. The approach is implemented and validated through the CTM prototype. Through a CTM case study we have shown that the presented approach is adequate and efficiently reduces the cognitive distance between work tasks and Wf modeling (End-User Development) tasks. The approach introduces several gentle slopes of complexity and provides added value on personal task management as motivation to overcome each one of them. Usage of CTM ad-hoc tasks is motivated through transparency in collaborative processes, exceeding common email and to-do list capabilities. The proactive extraction and adaptation of TPs is motivated through the ability to exchange and reuse previous experience.

The presented approach further enables transformation of implicitly generated TDGs to formal process models. The formalization benefits from multiple representations and fosters tailoring as collaboration between business users, process designers and developers by allowing the latter to work in a shared context between user-defined and formal process representations. Deviations from formal Wfs during execution are enabled with on-demand, ad-hoc task hierarchies. In the case study we have shown how such deviations enable end-user driven process model refinement.

We will continue to investigate further scenarios of CTM usage in order to enhance the ad-hoc to formal conversion capabilities, considering also possibilities for Wf extensions with automated, computational tasks.

Acknowledgments. The reported work was supported financially by the German "Federal Ministry of Education and Research" (BMBF, project EUDISMES, number 01 IS E03 C). We thank to all participants in our user studies for their cooperation.

References

1. van der Aalst, W., Weijters, A.: Process mining: a research agenda. Computers in Industry, vol. 53. Elsevier B.V, Amsterdam (2003)
2. Agostini, A., De Michelis, G.: Rethinking CSCW systems: the architecture of Milano. In: ECSCW 1997, pp. 33–48. Springer, Heidelberg (1997)
3. Bellotti, V., Dalal, B., Good, N., Flynn, P., Bobrow, D.G., Ducheneaut, N.: What a To-Do: Studies of Task Management towards the Design of a Personal Task List Manager. In: CHI 2004, pp. 735–742. ACM Press, New York (2004)
4. Bellotti, V., Ducheneaut, N., Howard, M., Smith, I., Grinter, R.: Quality Versus Quantity: E-Mail-Centric Task Management and Its Relation With Overload. Human-Computer Interaction, vol. 20, pp. 89–138. Lawrence Erlbaum Associates, Mahwah (2005)
5. Bernstein, A.: How Can Cooperative Work Tools Support Dynamic Group Processes? Bridging the Specificity Frontier. In: CSCW 2000, pp. 279–288. ACM Press, New York (2000)
6. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann, San Francisco (1998)
7. Button, G.: What's Wrong With Speech-Act Theory. Computer Supported Cooperative Work 3(1), 39–42 (1994)
8. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A., Mehanjiev, N.: Meta-Design: A Manifesto for End-User Development. Communication of the ACM 47(9) (September 2004)
9. Forrester Research. Increase Business Agility with BPM Suites. Forrester Research Inc. (2006)
10. Herrmann, T.: Evolving Workflows by User-driven Coordination. In: Reichwald, R., Schlichter, J. (eds.) Tagungsband D-CSCW 2000, pp. 103–114. Teubner (2000)
11. Holz, H., Maus, H., Bernardi, A., Rostanin, O.: From Lightweight, Proactive Information Delivery to Business Process-Oriented Knowledge Management. Journal of Universal Knowledge Management (2), 101–127 (2005)
12. JBoss Business Process Management, jBPM, `http://docs.jboss.org/jbpm/v3/userguide/index.html`
13. Jorgensen, H.D.: Interactive Process Models. Ph.D. Thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
14. Lieberman, H.: Your Wish is My Command: Programming by Example. Morgan Kaufmann, San Francisco (2001)
15. Lieberman, H., Paterno, F., Wulf, V.: End-User Development. Springer, Heidelberg (2006)
16. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable systems: pressing the issues with buttons. In: Proc. CHI 1990, pp. 175–182. ACM Press, New York (1990)
17. Mørch, A., Mehandjiev, N.: Tailoring as Collaboration: The Mediating Role of Multiple Representations and Application Units. Computer Supported Cooperative Work 9(1), 75–100 (2000)
18. Object Management Group, BPMN, `http://www.bpmn.org/`
19. Riss, U., Rickayzen, A., Maus, H., van der Aalst, W.: Challenges for Business Process and Task Managemen. Journal of Universal Knowledge Management (2), 77–100 (2005)
20. Schwarz, S., Abecker, A., Maus, H., Sintek, M.: Anforderungen an die Workflow-Unterstützung für wissensintensive Geschäftsprozesse. In: WM 2001, 1st Conference for Professional Knowledge Management, Baden-Baden, Germany (2001)
21. Stoitsev, T., Scheidl, S., Spahn, M.: A Framework for Light-Weight Composition and Management of Ad-Hoc Business Processes. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 213–226. Springer, Heidelberg (2007)

The Refined Process Structure Tree

Jussi Vanhatalo[1,2], Hagen Völzer[1], and Jana Koehler[1]

[1] IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
{juv,hvo,koe}@zurich.ibm.com
[2] Institute of Architecture of Application Systems, University of Stuttgart, Germany

Abstract. We consider workflow graphs as a model for the control flow of a business process model and study the problem of *workflow graph parsing*, i.e., finding the structure of a workflow graph. More precisely, we want to find a decomposition of a workflow graph into a hierarchy of sub-workflows that are subgraphs with a single entry and a single exit of control. Such a decomposition is the crucial step, for example, to translate a process modeled in a graph-based language such as BPMN into a process modeled in a block-based language such as BPEL. For this and other applications, it is desirable that the decomposition be unique, *modular* and as fine as possible, where *modular* means that a local change of the workflow graph can only cause a local change of the decomposition. In this paper, we provide a decomposition that is unique, modular and finer than in previous work. It is based on and extends similar work for sequential programs by Tarjan and Valdes [11]. We show that our decomposition can be computed in linear time based on an algorithm by Hopcroft and Tarjan [3] that finds the triconnected components of a biconnected graph.

Keywords: Workflow graph parsing, Control flow, Model decomposition, BPMN to BPEL translation/ roundtripping, Subprocess detection, Graph theory.

1 Introduction

The control flow of a business process can often be modeled as a *workflow graph* [10]. Workflow graphs capture the core of many business process languages such as UML activity diagrams, BPMN and EPCs. We study the problem of *parsing* a workflow graph, that is, decomposing the workflow graph into a hierarchy of sub-workflows that have a single entry and a single exit of control, often also called *blocks*, and labeling these blocks with a syntactical category they belong to. Such categories are *sequence*, *if*, *repeat-until*, etc., see Fig. 1(a). Such a decomposition is also called a *parse* of the workflow graph. It can also be shown as a *parse tree*, see Fig. 1(c).

The parsing problem occurs when we want to translate a graph-based process description (e.g. a BPMN diagram) into a block-based process description (e.g. BPEL process), but there are also other use cases for workflow graph parsing. For example, Vanhatalo, Völzer and Leymann [14] show how parsing speeds up control-flow analysis. Küster et al. [6] show how differences between two process models can be detected and resolved based on decompositions of these process models. We believe that parsing also helps in understanding large processes and in finding reusable subprocesses.

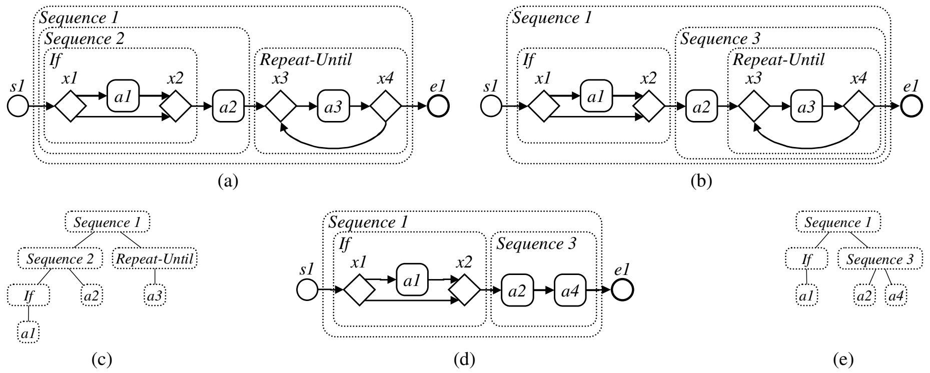


Fig. 1. (a), (b) Two parses of the same workflow graph. (c) Parse tree corresponding to (a). (d) Workflow graph obtained by a local change and its parse. (e) Parse tree corresponding to (d).

For a roundtripping between a BPMN diagram and a BPEL process, it is desirable that the decomposition be unique, i.e., the same BPMN diagram always translates to the same BPEL process. Consider, for example, the workflow graph in Fig. 1(a). The translation algorithm proposed by Ouyang et al. [9] is nondeterministic. It may produce one of the two parses shown in Fig. 1(a) and (b), depending on whether the if-block or the repeat-until-block is found first by the parsing algorithm.

One idea to resolve some of this nondeterminism is to define priorities on the syntactic categories to be found [9,7,8]. For example, if in each step the parsing algorithm tries to find sequences first, then if-blocks and then repeat-until-blocks, we can only obtain the parse in Fig. 1(a) in our example. However, this can introduce another problem. If we change a single block, say, the repeat-until block by replacing it, e.g. by a single task, we obtain the workflow graph shown in Fig. 1(d). Fig. 1(d) also shows the parse we obtain with the particular priorities mentioned above. The corresponding parse tree is shown in Fig. 1(e). It cannot be derived from the tree in Fig. 1(c) by just a local change, viz., by replacing the Repeat-Until subtree. For a roundtripping between a BPMN diagram and a BPEL process, it would be much more desirable that a local change in the BPMN diagram also result in only a local change in the BPEL process. Replacing a block in the BPMN diagram would therefore only require replacing the corresponding block in the BPEL process. We then call a such decomposition *modular*. The existing approach to the BPMN to BPEL translation problem [9] is not modular. Furthermore, it does not provide, because of the above problems, a specification of the translation that is independent of the actual translation algorithm.

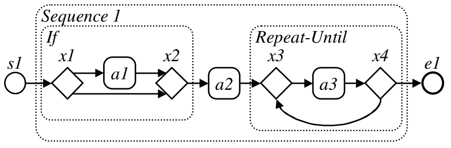


Fig. 2. Modular decomposition of the process from Fig. 1

A unique and modular decomposition is provided by the *program structure tree* defined by Johnson et al. [4,5] for sequential programs. It was applied to workflow graphs by Vanhatalo et al. [14] to find control-flow errors. The corresponding decomposition for our first example is shown in Fig. 2. It uses the same notion of a block as Ouyang et al. [9] do, that is, a block is a connected subgraph with a single entry and a single exit edge. But in contrast to the approach of Ouyang et al. [9], non-maximal sequences are disregarded in the program structure tree. For example, Sequence 2 in Fig. 1(a) [likewise Sequence 3 in subfigure (b)] is non-maximal: it is in sequence with another block.

Another general requirement for parsing is to find as much structure as possible, i.e., to decompose into blocks that are as fine as possible. As we will see (cf. Sect. 4), this allows us to map more BPMN diagrams to BPEL in a structured way. It has also been argued [9] that the BPEL process is more readable if it contains more blocks. Furthermore, debugging is easier when an error is local to a small block rather than to a large one.

In this paper, we provide a new decomposition that is finer than the program structure tree as defined by Johnson et al. [4,5]. It is based on and extends similar work for sequential programs by Tarjan and Valdes [11]. The underlying notion of a block is a connected subgraph with unique entry and exit *nodes* (as opposed to *edges* in the previous approach). Accordingly, all blocks of the previous approach are found, but more may be found, resulting in a more refined parse tree. We prove that our decomposition is unique and modular. Moreover, we show that it can be computed in linear time, based on an algorithm by Hopcroft and Tarjan [3] that finds the triconnected components of a biconnected graph.

The paper is structured as follows. In Sect. 2, we define the refined process structure tree and discuss its main properties. In Sect. 3, we describe how to compute the process structure tree in linear time. Proofs of the main theorems can be found in a technical report [13].

2 The Refined Process Structure Tree

In this section, we define the refined process structure tree (PST, for short). First, we explain our notion of fragments in Subsection 2.1. Fragments have a strong relationship with the *triconnected components* of the workflow graph, which we explain in Subsection 2.2. Subsection 2.3 defines the process structure tree. Finally, we show that our decomposition is modular.

2.1 Fragments

We start by recalling some basic notions of graph theory. A *multi-graph* is a graph in which two nodes may be connected by more than one edge. This can be formalized as a triple $G = (V, E, M)$, where V is the set of nodes, E the set of edges and M a mapping that assigns each edge an ordered or unordered pair of nodes—for a directed or undirected multi-graph, respectively. We will use multi-graphs throughout the paper, directed and undirected, but will call them graphs for simplicity.

Let G be a graph. If e is an edge of G that connects two nodes u and v, we also say that u and v are *incident* to e, e is *incident* to u and v, and nodes u and v are *adjacent*.

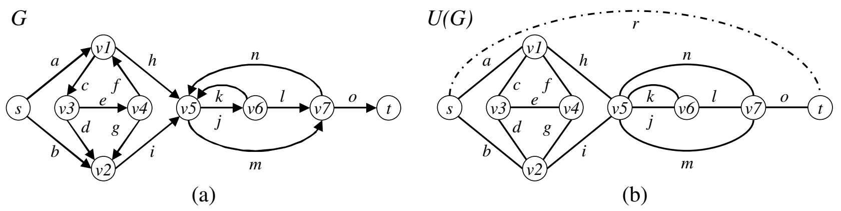


Fig. 3. (a) Two-terminal graph G, and (b) its undirected version $U(G)$, where r is the return edge

Workflow graphs are based on *two-terminal graphs*[1]. A *two-terminal graph* (*TTG* for short) is a directed graph G without self-loops such that there is a unique source node s and a unique sink node $t \neq s$ and each node v is on a directed path from s to t. The *undirected version* of G, denoted $U(G)$, is the undirected graph that results from ignoring the direction of all the edges of G and adding an additional edge between the source and the sink. The additional edge is called the *return edge* of $U(G)$. Figure 3 shows examples of (a) a two-terminal graph G, and (b) its undirected version $U(G)$, where r is the return edge.

For a subset F of edges, let V_F denote the set of nodes that are incident to some edge in F and let G_F denote the subgraph with nodes V_F and edges F. We say that G_F is *formed by* F.

Let G be a TTG and F a subset of its edges such that G_F is a connected subgraph of G. A node $v \in V_F$ is a *boundary node* of F if it is the source or sink node of G, or if G has edges $e \in F$ and $e' \notin F$ such that v is incident to e and e'. A boundary node v is an *entry* of F if no incoming edge of v is in F or if all outgoing edges of v are in F. A boundary node v is an *exit* of F if all incoming edges of v are in F or if no outgoing edge of v is in F. F is called a *fragment* of G if it has exactly two boundary nodes, an entry and an exit. Let $\mathscr{F}(u, v)$ denote the set of all fragments with entry u and exit v.

Figure 4 shows examples of fragments. A fragment is indicated as a dotted box. It contains all those edges that either are inside the box or cross the boundary of the box. Thus, the box in subfigure (a) denotes the fragment $F1 = \{a, b, c\}$. Node u is the entry and v is the exit of $F1$. In subfigure (b), $F2 = \{a, b, c, d\}$ is a fragment with entry u and exit v. In subfigure (c), $F3 = \{a, b, c, d\}$ has two boundary nodes, u and v, neither of them is an entry or an exit of $F3$. Therefore, $F3$ is not a fragment.

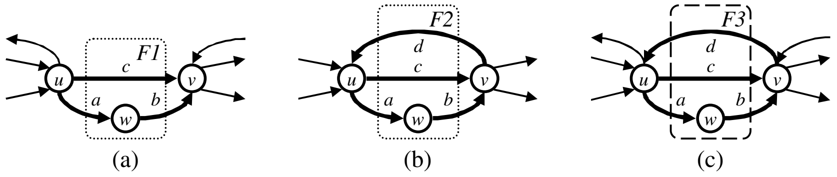


Fig. 4. (a), (b) Examples and (c) counterexamples of entry, exit and fragment

[1] A workflow graph is a two-terminal graph in which each node is labeled with some control flow logic such as AND, OR, etc.

Note that it can be checked locally whether a boundary node is an entry or an exit. This notion of fragment was proposed by Tarjan and Valdes [11], where a TTG modeled the control flow of a sequential program. When control flows through any of the edges of a fragment, then it must have flown through the entry before and must flow through the exit after. Their notion of fragment is, in a sense, the most general notion of fragment having this property that can still be verified locally [11].

2.2 Triconnected Components

Tarjan and Valdes [11] have shown that the fragments of a TTG are closely related to the *triconnected components* of its undirected version. We have to introduce a few more notions from graph theory.

Let G be an undirected graph. The following notions are also used for directed graphs by ignoring the direction of their edges. Let W be a subset of nodes of G. Two nodes $u, v \notin W$ are *connected without* W if there is a path that contains u and v but no node $w \in W$. For instance, in Fig. 3(a) nodes s and t are connected without $v6$, but not connected without $v5$. Two edges e, f are *connected without* W if there exists a path containing e and f in which a node $w \in W$ may only occur as the first or last element. A graph without self-loops is *k-connected*, $k > 0$, if it has at least $k + 1$ nodes and for every set W of $k - 1$ nodes, any two nodes $u, v \notin W$ are connected without W. We say *connected* for 1-connected, *biconnected* for 2-connected and *triconnected* for 3-connected. A *separation point* (*separation pair*) of G is a node u (pair $\{u, v\}$ of nodes) such that there exists two nodes that are not connected without $\{u\}$ (without $\{u, v\}$). Therefore a graph is biconnected (triconnected) if and only if it has no separation point (separation pair). For instance in Fig. 3, G is not biconnected, because $v5$ is a separation point, whereas $U(G)$ is biconnected, because it has no separation points. $U(G)$ is not triconnected, because $\{v5, v7\}$ is a separation pair. In Fig. 5(a), $T2$ is an example of a triconnected graph if the dashed edge x is considered as an ordinary edge.

We say that a graph is *weakly biconnected* if it is biconnected or if it contains exactly two nodes and at least two edges between these two nodes. For instance, in Fig. 5(a), $B1$ is weakly biconnected, but not biconnected.

Throughout the paper, we assume that $U(G)$ is weakly biconnected. This can easily be achieved by splitting each separation point into two nodes, where the only outgoing edge of the first node is the only incoming edge of the second node. [2]

Let $\{u, v\}$ be a pair of nodes. A *separation class* with respect to (w.r.t.) $\{u, v\}$ is a maximal set S of edges such that any pair of edges in S is connected without $\{u, v\}$; S is a *proper separation class* or *branch* if it does not contain the return edge; $\{u, v\}$ is called a *boundary pair* if there are at least two separation classes w.r.t. $\{u, v\}$. Note that a pair $\{u, v\}$ of nodes is a boundary pair if and only if it is a separation pair or u and v are adjacent in G. For instance in Fig. 3(b), $\{v5, v7\}$ and $\{v6, v7\}$ are boundary pairs. The pair $\{v5, v7\}$ is also a separation pair, but $\{v6, v7\}$ is not.

Each weakly biconnected graph can be uniquely decomposed into a set of graphs, called its *triconnected components* [3], where each triconnected component is either a

[2] It is often assumed for workflow graphs that no node has both multiple incoming and multiple outgoing edges. In that case it follows that $U(G)$ is biconnected. See also Sect. 4.

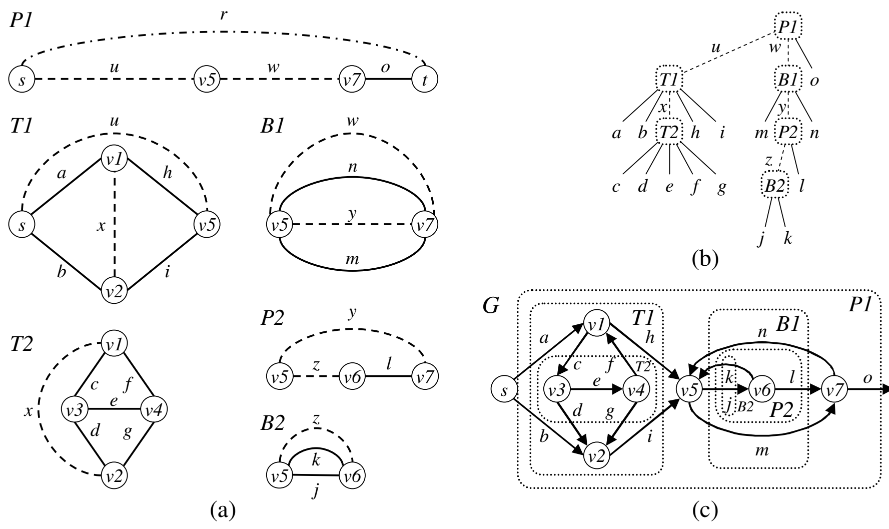


Fig. 5. (a) The triconnected components of $U(G)$ in Fig. 3, (b) the tree of the triconnected components of $U(G)$, and (c) the corresponding component subgraphs of G

bond, a *polygon* or a triconnected graph. A *bond* is a graph that contains exactly two nodes and at least two edges between them. A *polygon* is a graph that contains at least three nodes, exactly as many edges as nodes such that there is a cycle that contains all its nodes and all its edges.

Part (a) in Fig. 5 shows the six triconnected components of graph $U(G)$ from Fig. 3. $P1$ and $P2$ are polygons, $B1$ and $B2$ are bonds, and $T1$ and $T2$ are triconnected graphs. Each component contains *virtual edges* (shown as dashed lines), which are not contained in the original graph. They contain the information on how the components are related to each other: Each virtual edge occurs in exactly two components, whereas each original edge occurs in exactly one component. For example, the virtual edge x occurs in components $T1$ and $T2$. In component $T1$, x represents the component $T2$, whereas x represents $T1$ in $T2$. Therefore, we obtain the original graph by merging the triconnected components at the virtual edges (which removes them).

The triconnected components can be arranged in a tree, cf. Fig. 5(b), where two components are connected if they share a virtual edge. The root of the tree is the unique component that contains the return edge. Each original edge is also shown in the tree under the unique component that contains that edge. Therefore, each component C determines a set F of edges of the original graph, namely all the leafs of the subtree that C corresponds to. For example, component $T1$ determines the set $F = \{a, b, c, d, e, f, g, h, i\}$ of edges. We call the subgraph formed by such a set F of edges the *component subgraph* of C. Figure 5(c) shows the component subgraphs of G. Note that the component subgraphs $B1$, $P1$ and $T1$ are fragments, whereas $B2$, $P2$ and $T2$ are not. There are also fragments that are not component subgraphs, for instance, $\{j, k, l, m\}$.

The precise definition of the triconnected components is rather lengthy and has therefore been omitted (see [12,2,3]). Instead we present here the exact relationship between the triconnected components and fragments we are going to exploit. The proofs of the following two theorems can be found in [12]. First, we observe that triconnected components are closely related to boundary pairs.

Theorem 1. *A set* $\{u, v\}$ *of two nodes is a boundary pair of* $U(G)$ *if and only if*

1. *nodes* u *and* v *are adjacent in* $U(G)$*,*
2. *a triconnected component of* $U(G)$ *contains a virtual edge between* u *and* v*, or*
3. *a triconnected component of* $U(G)$ *is a polygon and contains* u *and* v*.*

We show examples based on $U(G)$ in Fig. 3(b) and its triconnected components in Fig. 5(a). For instance, the boundary pair $\{v6, v7\}$ contains two adjacent nodes of $U(G)$, the boundary pair $\{v1, v2\}$ corresponds to a virtual edge x of $T2$, and the boundary pair $\{s, v7\}$ contains two nodes of the polygon $P1$. Boundary pairs are closely related to fragments as follows.

Theorem 2. 1. *If* $F \in \mathscr{F}(u, v)$*, then* $\{u, v\}$ *is a boundary pair of* $U(G)$ *and* F *is the union of one or more proper separation classes w.r.t.* $\{u, v\}$*.*
2. *Let* $\{u, v\}$ *be a boundary pair of* $U(G)$ *and* F *the union of one or more proper separation classes w.r.t.* $\{u, v\}$*. If* u *is an entry of* F *and* v *is an exit of* F*, then* $F \in \mathscr{F}(u, v)$*.*

For instance, the boundary pair $\{v5, v7\}$ has three proper separation classes $\{m\}$, $P2 = \{j, k, l\}$, and $\{n\}$. $P2$ is not a fragment, because $v5$ is neither its entry nor its exit, whereas $\{m\} \in \mathscr{F}(v5, v7)$ and $\{n\} \in \mathscr{F}(v7, v5)$ are fragments. The union of $P2$ and $\{m\}$ is a fragment, whereas $P2 \cup \{n\}$ and $\{m\} \cup \{n\}$ are not. $P2 \cup \{m\} \cup \{n\}$ is a fragment.

Theorem 1 says that the boundary pairs can be obtained from the triconnected components while Thm. 2 says that the fragments can be obtained from the boundary pairs.

2.3 Canonical Fragments and the Process Structure Tree

Two fragments F_1 and F_2 may *overlap*, that is, we have $F_1 \cap F_2 \neq \emptyset$, $F_1 \setminus F_2 \neq \emptyset$ and $F_2 \setminus F_1 \neq \emptyset$. Examples of overlapping fragments are shown in Fig. 6. Overlapping fragments give rise to nondeterministic parsing as explained in Sect. 1. We are therefore interested in a subset of fragments that do not overlap with each other. These will be called *canonical*. We comment on our particular definition of canonical fragments in Sect. 4. We start by defining various types of *bond fragments*.

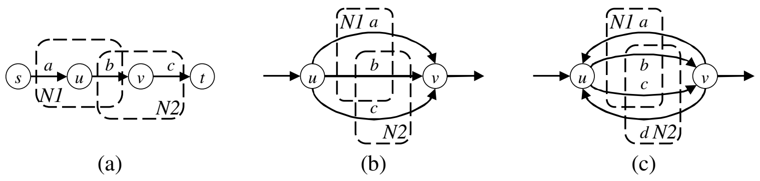


Fig. 6. Examples of overlapping fragments

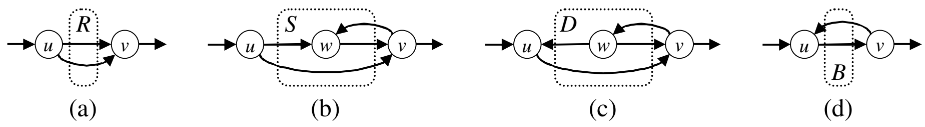


Fig. 7. Examples of (a) a pure bond fragment, (b) a semi-pure bond fragment, (c) a directed bond fragment, and (d) a bond fragment

Definition 1 (Bond fragments). *Let S be a proper separation class (i.e., a branch) w.r.t. $\{u, v\}$. S is* directed from u to v *if it contains neither an incoming edge of u nor an outgoing edge of v. $\mathscr{D}(u, v)$ denotes the set of directed branches from u to v. S is* undirected *if it is neither in $\mathscr{D}(u, v)$ nor in $\mathscr{D}(v, u)$. The set of undirected branches between u and v is denoted by $\mathscr{U}(u, v)$. A fragment $X \in \mathscr{F}(u, v)$ is*

1. *a* bond fragment *if it is the union of at least two branches from $\mathscr{D}(u, v) \cup \mathscr{U}(u, v) \cup \mathscr{D}(v, u)$.*
2. *a* directed bond fragment *if it is the union of at least two branches from $\mathscr{D}(u, v) \cup \mathscr{U}(u, v)$.*
3. *a* semi-pure bond fragment *if it is the union of at least two branches from $\mathscr{D}(u, v) \cup \mathscr{U}(u, v)$, and*
   (a) there exists no $Y \in \mathscr{U}(u, v)$ such that $Y \subseteq X$, Y has an edge incoming to u, or
   (b) there exists no $Y \in \mathscr{U}(u, v)$ such that $Y \subseteq X$, Y has an edge outgoing from v.
4. *a* pure bond fragment *if it is the union of at least two branches from $\mathscr{D}(u, v)$.*

Note that the various bond-fragment types form a hierarchy, i.e., each pure bond fragment is a semi-pure bond fragment, each semi-pure bond fragment is a directed bond fragment etc. Fig. 7 shows examples of various classes of bond fragments that do not belong to a lower class. Bond fragments are closed under composition, i.e., we have:

Proposition 1. *If $X, Y \in \mathscr{F}(u, v)$ and $F = X \cup Y$, then $F \in \mathscr{F}(u, v)$. If X and Y are bond fragments, so is F. If X and Y are directed (semi-pure) [pure] bond fragments, so is F.*

Proposition 1 assures that a maximal bond fragment, maximal directed, maximal semi-pure, or maximal pure bond fragment is unique if it exists. We are now ready to define canonical fragments.

Definition 2 (Canonical fragment)

1. *If $F_0 \in \mathscr{F}(v_0, v_1)$ and $F_1 \in \mathscr{F}(v_1, v_2)$ such that $F_0 \cup F_1 = F \in \mathscr{F}(v_0, v_2)$, we say that F_0 and F_1 are* in sequence *(likewise: F_1 and F_0 are in sequence) and that F is a* sequence. *F is a* maximal sequence *if there is no fragment F_2 such that F and F_2 are in sequence.*
2. *A bond fragment (directed bond fragment etc.) $F \in \mathscr{F}(u, v)$ is* maximal *if there is no bond fragment (directed bond fragment etc.) $F' \in \mathscr{F}(u, v)$ that properly contains*

F. A bond fragment $F \in \mathscr{F}(u, v)$ is canonical *if it is a maximal bond fragment, a maximal directed, maximal semi-pure, or maximal pure bond fragment such that F is not properly contained in any bond fragment $F' \in \mathscr{F}(v, u)$.*

3. *A fragment is* canonical *if it is a maximal sequence, a canonical bond fragment, or neither a sequence nor a bond fragment.*

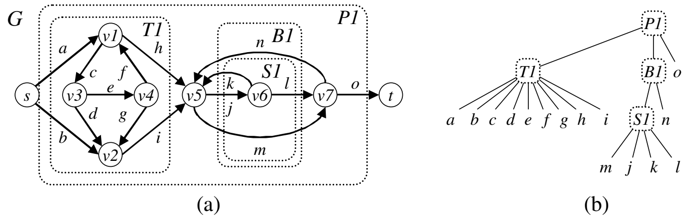


Fig. 8. (a) The non-trivial canonical fragments of G, and (b) the process structure tree of G

Note that each edge is a canonical fragment, which we call a *trivial fragment*. Part (a) of Fig. 8 shows the non-trivial canonical fragments of graph G. $S1 \in \mathscr{F}(v5, v7)$ is a maximal semi-pure bond fragment, and $B1 \in \mathscr{F}(v5, v7)$ is a maximal bond fragment. $P1$ is a maximal sequence. $T1$ is neither a sequence nor a bond fragment.

To prove that canonical fragments do not overlap, i.e., two canonical fragments are either nested or disjoint, this claim is proven first for bond fragments that have the same entry-exit pair.

Lemma 1. *Let $X, Y \in \mathscr{F}(u, v)$ be canonical bond fragments. Then $X \subseteq Y$, $Y \subseteq X$ or $X \cap Y = \emptyset$.*

We continue by showing that two canonical bond fragments that share the same boundary pair do not overlap. In general, we can encounter two situations depending on whether the union of all branches with respect to a boundary pair is a fragment. These two cases are shown in Fig. 9.

In Fig. 9(a), the union of all branches with respect to the boundary pair {u, v} is the maximal bond fragment from u to v called B. Fragments D, S and R are the maximal directed bond, semi-pure bond, and pure bond fragment from u to v respectively. Compared with part (a) of Fig. 9, part (b) has an additional edge outgoing from u that is outside of the union of all branches with respect to the boundary pair $\{u, v\}$. Because of this added edge, neither u or v is an entry of this subgraph. Thus, this set of edges is not a fragment. Fragment $R1$ is the maximal pure bond fragment from u to v. Fragment S is the maximal semi-pure bond fragment from u to v. As there is no larger bond fragment

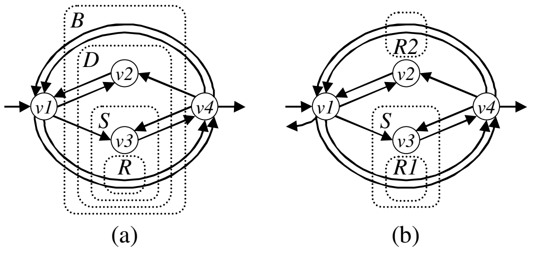


Fig. 9. Examples of canonical bond fragments

from u to v, S is also the maximal directed bond fragment and the maximal bond fragment from u to v. $R2$ is the maximal pure bond fragment from v to u. As there is no larger bond fragment from v to u, $R2$ is also the maximal semi-pure bond, the maximal directed bond and the maximal bond fragment from v to u. Through an analysis of these two cases, we can prove the following:

Lemma 2. *Let $X \in \mathscr{F}(u, v)$ and $Y \in \mathscr{F}(v, u)$ be canonical bond fragments. Then $X \subseteq Y$, $Y \subseteq X$ or $X \cap Y = \emptyset$.*

We are now ready to present the main theorem.

Theorem 3. *Let X, Y be two canonical fragments. Then $X \subseteq Y$, $Y \subseteq X$ or $X \cap Y = \emptyset$.*

Theorem 3 allows us to define the unique process structure tree of a workflow graph.

Definition 3 (Process structure tree). *Let G be a TTG. The* process structure tree *(*PST*, for short) is the tree of canonical fragments of G such that the parent of a canonical fragment F is the smallest canonical fragment of G that properly contains F.*

Thus, the largest fragment that contains a whole workflow graph G is the root fragment of the PST. Part (b) of Fig. 8 shows the PST of graph G in part (a). The child fragments of a sequence $P1$ are ordered left to right from the entry to the exit of $P1$. For example, the order of child fragments of maximal sequence $P1$ is $T1$, $B1$ and o. Moreover, as $T1$ has the same entry as $P1$, the exit of $T1$ ($B1$) is the entry of $B1$ (o), and o has the same exit as $P1$. We use this ordering in Sect. 2.4 to derive all fragments from the canonical fragments. For this, it is not necessary to order the child fragments of a bond or a triconnected graph.

2.4 Computing All Fragments from the Canonical Fragments

The following proposition indicates how to derive all fragments from the canonical fragments. This is useful for example if one wants to find the smallest fragment that contains some given set of graph elements.

Proposition 2. *Let F be a set of edges in a TTG. F is fragment if and only if F is a canonical fragment or F is*

1. *a union of consecutive child fragments of a maximal sequence,*
2. *a union of child fragments of a maximal pure bond fragment, or*
3. *a union of child fragments of a maximal bond fragment B such that B is not a maximal directed bond fragment.*

For example, the maximal sequence $P1$ in Fig. 8 has $T1$, $B1$ and o as ordered child fragments. Besides these canonical fragments and the maximal sequence, also the union of $T1$ and $B1$ ($B1$ and o) is a fragment. However, the union of $T1$ and o is not a fragment, as these are not consecutive child fragments, i.e., they do not share a boundary node.

Part (a) of Fig. 10 shows a maximal pure bond fragment $R = \{a, b, c\}$. Its child fragments are $\{a\}$, $\{b\}$, and $\{c\}$. It follows from Prop. 2 that $\{a, b\}$, $\{b, c\}$, and $\{a, c\}$ are the non-canonical fragments in R. Part (b) of Fig. 10 shows a maximal bond fragment $B = \{a, b, c, d\} \in \mathscr{F}(u, v)$ and a maximal directed bond fragment $R = \{a, b\} \in \mathscr{F}(u, v)$ such that $B \neq R$. It follows from Prop. 2 that $\{c, d\}$, $\{a, b, c\}$ and $\{a, b, d\}$ are the non-canonical fragments in B. Note that $\{a, c, d\}$ and $\{b, c, d\}$ are not fragments, because their boundary nodes are neither entries nor exits.

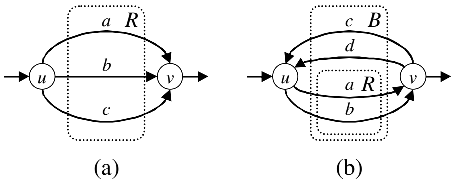


Fig. 10. Deriving non-canonical fragments from child fragments of (a) a maximal pure bond fragment and (b) a maximal bond fragment

2.5 Modularity

Finally, we state what we mean by saying that our decomposition is modular.

Theorem 4. *Let G be a TTG and $X \in \mathscr{F}(u, v)$ be a canonical fragment of G. Let G' be the TTG obtained by replacing the subgraph that is formed by X by some other subgraph formed by a set of (fresh) edges X' such that $X' \in \mathscr{F}(u, v)$ is again a fragment of G' (but not necessarily canonical) with the same entry-exit pair as X. Assume that A is the parent fragment of X in G and $F \neq X$ is a child fragment of A in G. Let $A' = (A \setminus X) \cup X'$ and $F' = F$. Then A' and F' are canonical fragments of G' where F' is a child fragment of A' in G'.*

Theorem 4 means that a local change to the TTG, i.e., changing a canonical fragment X, only affects the PST locally. The parent and siblings of a changed canonical fragment remain in the PST in the same place and it follows inductively that this is also true for all canonical fragments above the parent and all canonical fragments below the siblings of X.

3 Computing the PST

In this section, we describe an algorithm that computes the PST in linear time. We have extended the algorithm by Valdes [1] to find all the canonical fragments (his algorithm produces a coarser decomposition, cf. Sect. 4). The algorithm has three high-level steps that are illustrated in Fig. 11 and described in Alg. 1. In Step 1, the tree of the triconnected components is computed, using e.g. the linear-time algorithm by Hopcroft and Tarjan [3]. Gutwenger and Mutzel [2] present some corrections to this algorithm. We illustrate the computed triconnected components through the respective component subgraphs in Fig. 11.

In Step 2, we analyze each triconnected component to determine whether the respective component subgraph is a fragment. This can be done in linear time with the following approach that takes advantage of the hierarchy of fragments. We analyze the tree of the triconnected components bottom-up—all children before a parent. For each

Algorithm 1. Computes the PST for a two-terminal graph

```
buildPST(Graph $G$)
Require: $G$ is a weakly biconnected TTG.
  // Step 1. Compute the tree of the triconnected components of the TTG.
  Tree := Compute the tree of the triconnected components of $G$.
  // Step 2. Analyze each component to determine whether it is a fragment.
  for each Component $c$ in Tree in a post-order of a depth-first traversal do
    Count the number of edges (that are children of $c$) incoming to/outgoing from each boundary
    node. (4 counts)
    Add these edge counts to the respective edge counts of the parent component for each shared
    boundary node.
    Compare these edge counts to the total number of incoming/outgoing edges to determine
    whether each boundary node is entry, exit, or neither.
    Based on these boundary node types, determine whether $c$ is a fragment.
    if $c$ is a polygon then
      Count the number of entry and exit nodes of the child components.
      If a child component is a fragment, order the child components from entry to exit.
  // Step 3. Restructure the tree of the triconnected components into the tree of the canonical
  fragments (the PST).
  for each Component $c$ in Tree in a post-order of a depth-first traversal do
    if $c$ is a polygon then
      Merge consecutive child components (that are not fragments if any exist) if those form a
      minimal child fragment.
      if $c$ is not a fragment and $c$ has at least two child fragments then
        Create a maximal sequence (that contains a proper subset of children of $c$).
    if $c$ is a bond then
      Classify each branch of $c$ based on the edge counts of the boundary nodes of the respective
      child components of $c$.
      if $c$ is a fragment then
        Based on the classifications of the branches, create the maximal pure, the maximal
        semi-pure, and the maximal directed bond fragment, if any exists.
      else
        Based on the classifications of the branches, create the maximal pure bond fragments,
        the maximal semi-pure bond fragment, if any exists.
    for each Component $d$ that has been created in this iteration do
      Merge the child fragments of each component in the to-be-merged-list of $d$ to $d$.
    if $c$ is not a fragment then
      Add $c$ to the to-be-merged list (a linked list) of its parent component.
      Concatenate the to-be-merged list of $c$ to the to-be-merged list of its parent.
    else
      Merge the child fragments of each component in the to-be-merged list of $c$ to $c$.
  return Tree
```

child edge of a triconnected component c, we check whether it is incoming to or outgoing from one of the two boundary nodes of c. We count these edges to determine whether a boundary node is an entry, an exit, or neither. Based on this information, we can determine whether the respective component subgraph is a fragment. Note that

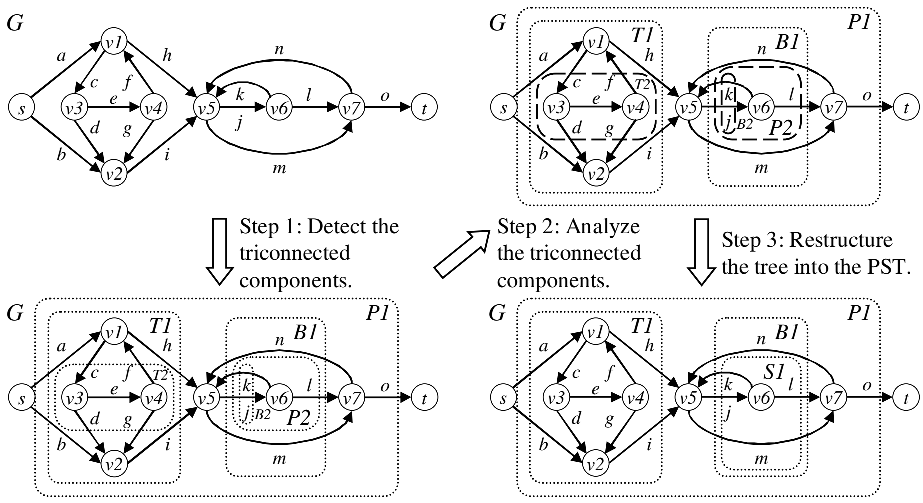

G
Step 1: Detect the triconnected components.
Step 2: Analyze the triconnected components.
Step 3: Restructure the tree into the PST.
T1
T2
P1
B1
B2
P2
S1
s
v1
v2
v3
v4
v5
v6
v7
t

Fig. 11. The high-level steps of Alg. 1. Step 1: Detect the triconnected components. Step 2: Analyze each triconnected component to determine whether the respective component subgraph is a fragment. Step 3: Create the missing canonical fragments and merge the triconnected components that are not fragments.

when a triconnected component shares a boundary node with its parent, the same edges do not have to be counted twice, because an edge inside a child is also inside its parent.

In Step 3, we create the missing canonical fragments, and merge each component subgraph that is not a fragment to the smallest canonical fragment that contains it. This restructuring is based on the information computed in Step 2. New fragments are created only in those cases where a bond or a polygon contains canonical fragments that are not component subgraphs. Such a fragment is created as a union of at least two (but not all) children of this bond or polygon. We show examples in the following.

We process the tree of the triconnected components bottom-up as in Step 2. Thus, in Fig. 11, we can begin with $T2$. It contains no new canonical fragments, because it is neither a sequence nor a bond. $T2$ is not a fragment, because $v1$ is neither its entry nor its exit. Thus, it will be merged into its parent fragment $T1$, that is, the children of $T2$ become children of $T1$.

The bond $B2$ is not a fragment, so it will be merged. $B2$ contains no new canonical fragments, because it has only two children. The same applies to $P2$. More interestingly, $B1$ is a fragment and has three children. Each child of a bond is a branch, and we classify them to find out whether they form new canonical bond fragments. $\{m\}$ is a directed branch from $v5$ to $v7$, $P2$ is an undirected branch that has no outgoing edges from $v7$, and $\{n\}$ is a directed branch from $v7$ to $v5$. Note that the branches can be classified based on the counts of edges incident to each boundary node of a branch computed in Step 2. There is a new semi-pure bond fragment $S1 = \{m\} \cup P2$. $B2$ and $P2$ are merged to $S1$. $S1$ and $\{n\}$ become the children of the restructured $B1$. Finally, $P1$ and all its children are fragments, thus there is no need to restructure $P1$.

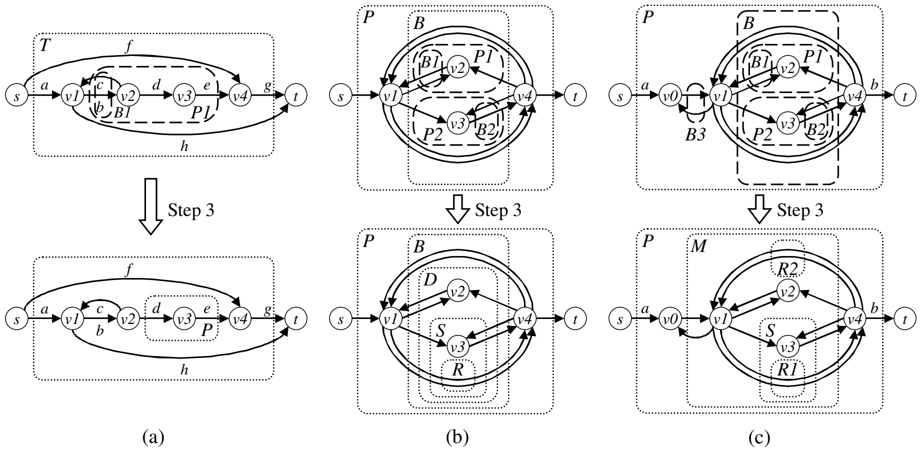


Fig. 12. Step 3: From the tree of the component subgraphs to the tree of the canonical fragments

In the following examples we show polygons and bonds in which more restructuring is required. In Fig. 12(a), $B1$ and $P1$ are not fragments. However, polygon $P1$ has two consecutive child fragments $\{d\}$ and $\{e\}$ that form a maximal sequence $P = \{d\} \cup \{e\}$. To determine whether a polygon contains such a new maximal sequence, we compute the number of entries and exits of its children already at the end of Step 2. A polygon that is not a fragment contains a maximal sequence as a union of its children if and only if its children have at least three entries or at least three exits in total.

In Fig. 12(b), $B1$, $P1$, $B2$, and $P2$ are not fragments and will be merged. Bond B is a fragment from $v1$ to $v4$ and has six branches: two edges as directed branches from $v1$ to $v4$, and one undirected branch, $P2$, that has no edge incoming to the entry of B, one undirected branch, $P1$, that has both an edge incoming to the entry of B and an edge outgoing from the exit of B, and another two edges as directed branches from $v4$ to $v1$. The directed branches from the entry to the exit of B form a new maximal pure bond fragment R. The union of $P2$ and R is a new maximal semi-pure bond fragment S. The union of $P1$ and S is a new maximal directed bond fragment. D and the remaining two directed branches are the children of B. $B1$ and $P1$ are merged to D, and $B2$ and $P2$ to S. P is a maximal sequence.

Figure 12(c) shows an example of a bond B that is not a fragment, but its children form new canonical fragments. As there are at least two directed branches to each direction, these branches form two new pure bond fragments, $R1$ and $R2$. The union of $R1$ and branch $P2$ is a semi-pure bond fragment S. Thus, $B2$ and $P2$ are merged to S. The polygon P has four children $\{a\}$, $B3$, B, and $\{b\}$. $B3$ and B not fragments, but the union of these consecutive siblings is a fragment. Thus, B is merged to $B3$ to form a new fragment M. $B1$ and $P1$ are also merged to M. The fragment P has only three children.

Each step of the algorithm can be performed in linear time. Thus, also the entire algorithm has linear time complexity.

Theorem 5. *The PST of a TTG G can be computed in time linear in the number of edges of G.*

4 Conclusion

We have presented a modular technique of workflow graph parsing to obtain fine-grained fragments with a single entry and single exit node. The result of the parsing process, the process structure tree, is obtained in linear time. We have mentioned a couple of use cases in Sect. 1. Coarser-grained decompositions may be desirable for some use cases. Those can easily be derived from the refined process structure tree by flattening. One such coarser decomposition, which can be derived and which is also modular, is the decomposition into fragments with a single entry edge and a single exit edge presented by Vanhatalo, Völzer and Leymann [14]. The new, refined decomposition presented here allows us to translate more BPMN diagrams to BPEL in a structured way. As an example, consider the workflow graph in Fig. 13 and (a) its decomposition with the existing techniques [9,14] and (b) with our new technique. In Fig. 13(a), X cannot be represented as a single BPEL block, whereas in Fig. 13(b) each fragment can be represented as a single BPEL block.

The main idea of the technique presented is taken from Tarjan and Valdes [11,1]. They describe an algorithm that produces a unique parse tree. However, they do not provide a specification of the parse tree, i.e., a definition of canonical fragments or claim or prove modularity. Moreover, our PST is more refined than their parse tree. Figure 12 shows examples of workflow graphs where this is the case. The fragments that are not identified by them are P in (a), D, S and R in (b), and S, $R1$ and $R2$ in (c).

We have made some simplifying assumptions about workflow graphs. The assumption that we have unique source and sink nodes can be lifted. Also the assumptions that the undirected version of the workflow graph is weakly biconnected and does not contain self-loops can be lifted. The necessary constructions to deal with these cases will be presented in an extended version of this paper. Thus the remaining assumption on workflow graphs will be that each node is on a path from some source to some sink.

The reader might wonder what justifies our particular definition of canonical fragments. It can be shown that the canonical fragments are exactly those fragments that do not overlap with any (canonical or non-canonical) fragment. This means, they are exactly the 'objective' fragments in the sense that they are compatible with any parse and hence appear in every maximal parse. Any finer decomposition into fragments can

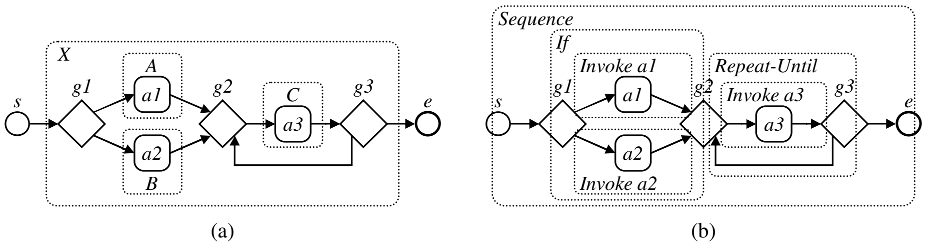


Fig. 13. A workflow graph and (a) decomposition presented in [9,14] and (b) our decomposition

only be obtained by arbitrating between overlapping fragments. Our definition is further justified by Prop. 2, i.e., by the fact that all fragments and hence all parses can be derived from the PST in a simple way.

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

References

1. Ayesta, J.V.: Parsing flowcharts and series-parallel graphs. PhD thesis, Stanford University, CA, USA (1978)
2. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
3. Hopcroft, J., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. 2, 135–158 (1973)
4. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI), pp. 171–185 (1994)
5. Johnson, R.C.: Efficient program analysis using dependence flow graphs. PhD thesis, Cornell University, Ithaca, NY, USA (1995)
6. Küster, J., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M., and Shan, M.-C., (eds.) BPM 2008. LNCS, vol. 5240, pp. 244–260. Springer, Heidelberg (2008)
7. Lassen, K.B., van der Aalst, W.M.P.: WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 127–144. Springer, Heidelberg (2006)
8. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into human-readable abstract BPEL processes. In: Modellierung 2008, GI, March 2008. Lecture Notes in Informatics (LNI), vol. P-127, pp. 57–72 (2008)
9. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, W.M.P.: From BPMN process models to BPEL web services. In: ICWS, pp. 285–292. IEEE Computer Society, Los Alamitos (2006)
10. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. 25(2), 117–134 (2000)
11. Tarjan, R.E., Valdes, J.: Prime subprogram parsing of a program. In: POPL 1980: Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 95–105. ACM, New York (1980)
12. Vanhatalo, J.: Structural analysis of business process models using the process structure trees (to appear)
13. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. IBM Research Report RZ 3712 (2008)
14. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models though SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)

Covering Places and Transitions in Open Nets

Christian Stahl[1] and Karsten Wolf[2]

[1] Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
stahl@informatik.hu-berlin.de
[2] Universität Rostock, Institut für Informatik
18051 Rostock, Germany
karsten.wolf@uni-rostock.de

Abstract. We present a finite representation of all services M where the composition with a given service N is deadlock-free, and a given set of activities of N can be *covered* (i.e. is not dead). Our representation is an extension of the existing notion of an operating guideline which only cared about deadlock freedom. We further present an algorithm to decide whether a service M matches with the extended operating guideline of N.

Keywords: process modeling and analysis, SOA, Petri nets, operating guidelines.

1 Introduction

One of the objectives of service-oriented computing (SOC) [1] is the modular structuring and loose coupling of interorganisational business processes. In this aspect, SOC meets the area of modeling and analysing workflows [2]. While SOC aims at composing complex business activities from more elementary ones (services), workflow modeling is (among others) concerned with the study of well-designed workflows and business processes. Central to the wellformedness of workflows is the concept of *soundness*. This property basically states that every process instance will terminate in a well-defined final state while there are no useless (dead) activities. In the intersection of SOC and workflow modeling, we are thus interested in mechanisms for service composition (and related tasks such as discovery) which assure soundness in the overall system (e.g. a service orchestration).

Current approaches for matching and discovering services are incapable of asserting soundness in service discovery scenarios. Some approaches propose to compute and publish a public view P' of a provided service P [3,4]. Then, a service requester R can check its composition $R \oplus P'$ to decide proper interaction. However, public view approaches do not explicitly state whether soundness of $P' \oplus R$ implies soundness of $P \oplus R$. Thus, existing public view approaches cannot be applied to obtain a globally sound system.

Other approaches suggest to compute an *operating guideline* OG_P for a given service P which represents all correctly interacting partners of P [5]. Then, a matching procedure between R and OG_P can be used for deciding whether $P \oplus R$ would interact correctly. Here, correctness refers to deadlock freedom so far.

Deadlock freedom is a necessary but insufficient condition for soundness. In this paper, we extend the operating guideline approach by asserting—in addition to deadlock freedom—the absence of dead activities in the composed system. This is another necessary condition for soundness. The only remaining gap between the new approach and soundness is the possible existence of livelocks. For acyclic services, our approach already establishes soundness in the composed system since acyclic services cannot contain livelocks.

Another motivating scenario for our approach is inspired by [6]. In this article, all partners of a given service, which *enforce* or *exclude* certain behavioral patterns such as occurrences of activities, are characterized. This approach can be used, among others, for

– filtering of service registries for services that fit specific specifications ("enforce book": I want to get a book selling service; "exclude credit card": I do not want to pay by credit card),
– validating services by checking whether there exist partners that access certain features

Sometimes, enforcing some behavior is too strict. Consider an application for a credit with an online bank service. Of course, the user (service requester) wishes to have the activity "credit approved" executed in the service. However, there is hardly an online bank service where "credit approved" can be enforced by the user (which would mean that the user can always obtain a credit by just following a suitable communication pattern). There will rather be an internal decision based on which a credit is either approved or denied. In typical service models, the decision appears to the user as a nondeterministic choice. Thus, we need a weaker criterion that rules out at least all those services where "credit approved" is completely impossible. That is, R should match with P if and only if it is at least *possible* to execute activity "credit approved" in the composition of the online bank service and the requester.

Formally, we want to compute a finite representation of the (generally infinite) set of all those partners R of a given service P where the composition $P \oplus R$ of both services is deadlock-free, and a certain set X of activities is not dead. For establishing soundness, this set X would be the set of all activities of P. In the online banking example, X would consist only of activity "credit approved". We achieve this goal by extending the existing operating guideline approach with deadlock-free interaction.

The paper is structured as follows. In Sect. 2 we recall open nets and operating guidelines. Next, in Sect. 3, we extend our notion of partners R for P to those partners R' where a certain set of places and transitions in $P \oplus R'$ is covered (i.e. each place can be marked and each transition is not dead). We show how to calculate a finite representation of all these partners by extending our notion of an operating guideline with a global constraint. Section 4 presents related work and finally conclusions are drawn in Sect. 5.

2 Preliminaries

2.1 Open Nets

We assume the usual definition of a (place/transition) Petri net $N = (P, T, F)$ (see [7], for instance) and use standard notation to denote the preset and postset of a place or a transition: ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.

Definition 1 (Open net). *An* open net $N = (P, T, F, I, O, m_0, \Omega)$ *consists of a Petri net* (P, T, F) *together with*

- *an* interface *defined as a set* $I \subseteq P$ *of* input places *such that* ${}^\bullet p = \emptyset$ *for any* $p \in I$ *and a set* $O \subseteq P$ *of* output places *such that* $p^\bullet = \emptyset$ *for any* $p \in O$ *and* $I \cap O = \emptyset$*,*
- *a distinguished* initial marking m_0*, and*
- *a set* Ω *of* final markings *such that no transition of* N *is enabled at any* $m \in \Omega$*.*

We further require that $m \in \Omega \cup \{m_0\}$ *implies* $m(p) = 0$ *for all* $p \in I \cup O$*; that is, in the initial and the final markings the interface places are not marked.*

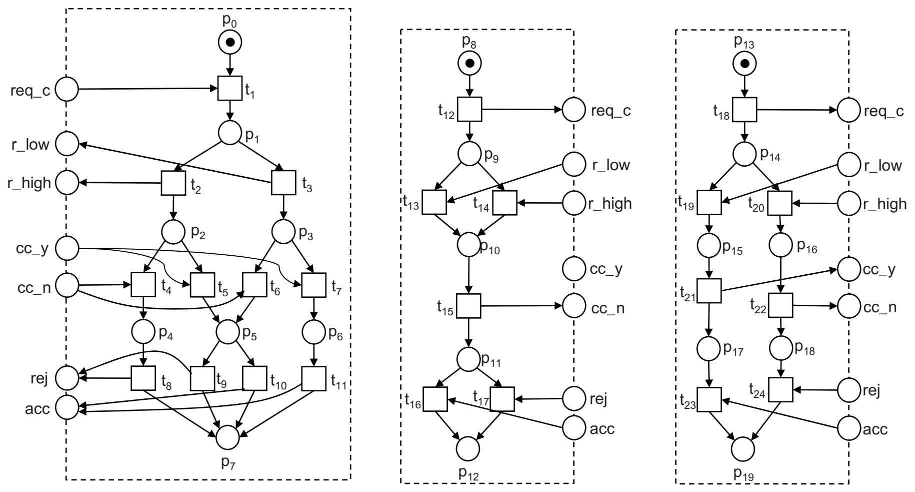
We use indices to distinguish the constituents of different open nets (e. g. I_j refers to the set of input places of open net N_j).

The behavior of an open net is defined using the standard Petri net semantics [7]; that is, a transition is enabled if each place of its preset holds a token. An enabled transition t can fire in a marking m by consuming tokens from the preset places and producing tokens on the postset places, yielding a marking m'. The firing of t is denoted by $m \xrightarrow{t} m'$ (a t-step), the successively firing of a sequence of transitions is denoted by $m \xrightarrow{*} m'$.

In order to assign a reasonable meaning to *final* markings, we restrict our approach to such open nets where a marking in Ω does not enable any transition.

As an example, consider the open net N_c depicted in Fig. 1(a). The initial marking is $m_{0_{N_c}} = [\mathsf{p}_0]$ and the set of final markings is defined by $\Omega_{N_c} = \{[\mathsf{p}_7]\}$. N_c has three input and four output places that are depicted on the dashed frame: $I_{N_c} = \{\mathsf{req\_c}, \mathsf{cc\_y}, \mathsf{cc\_n}\}$ and $O_{N_c} = \{\mathsf{r\_low}, \mathsf{r\_high}, \mathsf{rej}, \mathsf{acc}\}$. The open net models a credit approval process of an online banking service. After the customer has requested a credit (transition t_1), the bank decides whether the risk is high or low (transitions t_2 and t_3). Then, the customer has to decide whether he accepts a credit control or not (transitions $\mathsf{t}_4 - \mathsf{t}_7$). Based on this information the bank distinguishes three cases: If the risk is high and the customer does not accept a credit control, then the credit request is rejected (transition t_8). If there is only low risk and the customer accepts a credit control, then the request is accepted (transition t_{11}). In the third case, that is, if the risk is high and the customer accepts a credit control or the risk is low but the customer does not accept a credit control, the request is examined by an employee of the bank which is modeled by a nondeterministic choice (transitions t_9 and t_{10}).

The $inner_N$ of an open net N defines the Petri net that results from removing the interface places and the adjacent arcs from N. Obviously, $inner_N$ and N



(a) Open net N_c modeling the credit approval process of an online banking service.

(b) Open net M_1.

(c) Open net M_2.

Fig. 1. The running example process N_c and two strategies M_1 and M_2

coincide if N has an empty interface. The inner of N_c, $inner_{N_c}$, is the net inside the dashed frame in Fig. 1(a).

As a correctness criterion for an open net N we require the absence of deadlocks in N.

Definition 2 (Deadlock). *Let N be an open net. A* deadlock *is a nonfinal marking in N that does not enable a transition. If N does not have deadlocks, it is called* deadlock-free*.*

Two open nets M and N are *composable* if all constituents (except for the interfaces) are pairwise disjoint. This can be achieved easily by renaming. For the interfaces, we require that the input places of M are the output places of N and vice versa (i.e. $I_M = O_N$ and $O_M = I_N$). For markings $m_M \in M, m_N \in N$, their composition $m = m_M \oplus m_N$ is defined by $(m_M \oplus m_N)(p) = m_M(p) + m_N(p)$ (assuming $m_M(p) = 0$ for $p \notin P_M$ and $m_N(p) = 0$ for $p \notin P_N$). These considerations lead to the following definition of composition.

Definition 3 (Composition of open nets). *Let M, N be composable open nets. Then, the* composition *of M and N is the open net $M \oplus N$ defined as follows:*

- $P = P_M \cup P_N$,
- $T = T_M \cup T_N$,
- $F = F_M \cup F_N$,
- $I = O = \emptyset$,

– $m_0 = m_{0_M} \oplus m_{0_N}$, *and*
– $\Omega = \{m_M \oplus m_N \mid m_M \in \Omega_M, m_N \in \Omega_N\}$.

Consider the two open nets M_1 and M_2 depicted in Fig. 1(b) and Fig. 1(c), respectively and assume $m_{0_{M_1}} = [\mathsf{p}_8]$, $\Omega_{M_1} = \{[\mathsf{p}_{12}]\}$, $m_{0_{M_2}} = [\mathsf{p}_{13}]$, and $\Omega_{M_2} = \{[\mathsf{p}_{19}]\}$. Then, N_c and M_1 as well as N_c and M_2 are composable. Notice that place cc_y becomes internal in the composition $N_c \oplus M_1$, but it is never marked.

Clearly, we are mostly interested in composing open nets such that the composition is deadlock-free. To this end, we define the notion of a strategy.

Definition 4 (Strategy). *An open net M is a* strategy *for an open net N if $M \oplus N$ is deadlock-free. $Strat(N)$ denotes the set of all strategies for N.*

Both, $M_1 \oplus N_c$ and $M_2 \oplus N_c$, are deadlock-free and thus, M_1 and M_2 are strategies for N_c.

2.2 Operating Guidelines

In the following we recapitulate our concept of an *operating guideline* [8,5]. With the help of operating guidelines we are able to represent the set of all strategies M for an open net N in a compact way. Technically, an operating guideline is a special annotated automaton. An annotated automaton A^Φ consists of a finite deterministic automaton A and a function Φ that assigns to each state q of A a Boolean formula $\Phi(q)$. A^Φ represents a set $Strat(A^\Phi)$ of open nets. For each element of $N \in Strat(A^\Phi)$, we say that N *matches* with A^Φ. We continue by first defining the notions of annotated automata and matching in general and then introducing operating guidelines.

Definition 5 (Annotated automaton). $A^\Phi = [Q, C, \delta, q_0, \Phi]$ *is an* annotated automaton *iff Q is a nonempty finite set of* states, *C is a set of* labels, *$\delta \subseteq Q \times C \times Q$ is a* transition relation *such that every state $q \in Q$ is reachable from q_0 via transitive applications of δ, $q_0 \in Q$ is the* initial state, *and Φ is an* annotation *function, where, for all $q \in Q$, $\Phi(q)$ is a Boolean formula over literals in C.*

We use annotated automata to represent a set of *open nets*. Therefore, we take an annotated automaton A^Φ with Boolean formulae over literals in $C = I \cup O$ and a special literal *final* and define when a service described in terms of an open net M with the interface $I \cup O$ matches with A^Φ. Intuitively, M matches with A^Φ if (1) its *behavior* is simulated by A^Φ and (2) if a marking m of M is simulated by a state q of A^Φ, then the arcs leaving m — interpreted as an assignment assigning *true* to the corresponding literals of the formula $\Phi(q)$ — satisfy $\Phi(q)$. For more details, we refer to [9,5].

In order to simplify presentation, we assume that each transition of an open net is connected to at most one interface place. This assumption does, however, not restrict generality as every open net can be transformed into an equivalent one that obeys this restriction [5].

Definition 6 (Matching with A^{Φ}). *Let M be an open net that obeys the assumption stated above and let Y be the set of all reachable markings of the Petri net $M^* = inner_M$. Let $A^{\Phi} = (Q, C, \delta, q_0, \Phi)$ be an annotated automaton with $C = I_M \cup O_M \cup \{final\}$. Then M* matches *with A^{Φ} iff there is a relation $\rho \subseteq Y \times Q$ inductively defined as follows:*

1. $(m_{0_M}, q_0) \in \rho$;
2. *If t is an internal transition of M (i. e., t is not connected to any interface place), $m, m' \in Y$, and $m \xrightarrow{t} m'$, then $(m, q) \in \rho$ implies $(m', q) \in \rho$;*
3. *If t is a receiving transition of M with $c \in I_M$, $c \in {}^{\bullet}t$, $m, m' \in Y$, and $(m + [c]) \xrightarrow{t} m'$, then $(m, q) \in \rho$ implies $(m', q') \in \rho$ for some q' with $(q, c, q') \in \delta$;*
4. *If t is a sending transition of M with $c \in O_M$, $c \in t^{\bullet}$, $m, m' \in Y$, and $m \xrightarrow{t} (m' + [c])$, then $(m, q) \in \rho$ implies $(m', q') \in \rho$ for some q' with $(q, c, q') \in \delta$;*
5. *For all $m \in Y$, at least one of the following properties holds:*
   - *An internal transition t is enabled at m; or,*
   - *for all q such that $(m, q) \in \rho$, $\Phi(q)$ evaluates to* true *for the following assignment β:*
     - $\beta(c) =$ true *if $c \in O_M$ and there is a transition t with $c \in t^{\bullet}$ that is enabled at m;*
     - $\beta(c) =$ true *if $c \in I_M$ and there is a transition t with $c \in {}^{\bullet}t$ that is enabled at $m + [c]$;*
     - $\beta(c) =$ true *if $c = final$ and $m \in \Omega_M$;*
     - $\beta(c) =$ false*, otherwise.*

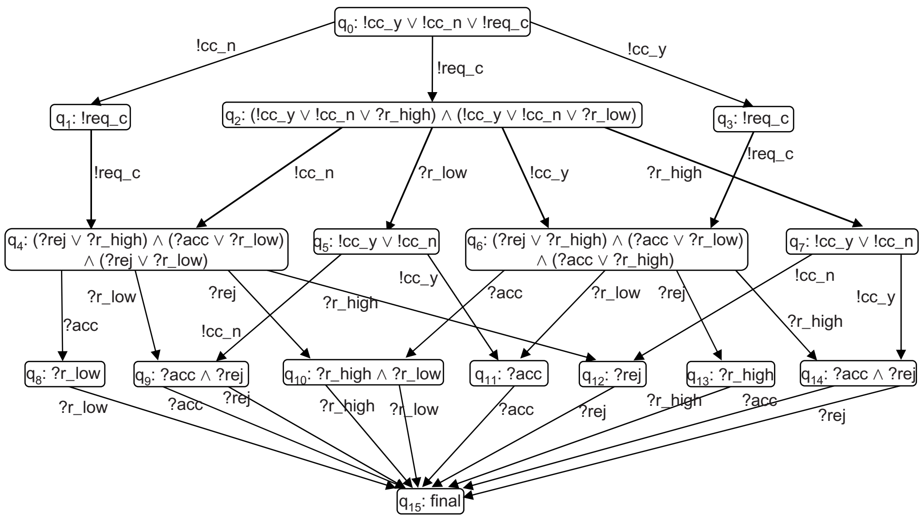
Let $Match(A^{\Phi})$ denote the set of all M such that M matches A^{Φ}.

In the formal definition, ρ represents the informally described (weak) simulation relation. The assignment used for evaluating an annotation represents transitions t of M that leave the considered marking m of M^*.

An operating guideline OG_N of an open net N is a special annotated automaton, such that an open net M matches with OG_N if and only if M is a strategy for N.

Definition 7 (Operating guideline). *An annotated automaton is an* operating guideline OG_N *of an open net N iff $Strat(N) = Match(OG_N)$.*

Figure 2 depicts the operating guideline OG_{N_c} for the credit approval process N_c (see Fig. 1(a)). It consists of 16 nodes and 31 edges and was calculated by our tool Fiona [10]. In the initial state $\mathsf{q_0}$, the annotation is $\mathsf{!cc\_y} \vee \mathsf{!cc\_n} \vee \mathsf{!req\_c}$ reflecting the possible choices of a strategy M for N_c. More precisely, M must be able to send at least one (expressed by the disjunction) of the three messages cc_y, cc_n, and req_c in its initial state. In contrast, annotation $\mathsf{?acc} \wedge \mathsf{?rej}$ in state $\mathsf{q_{14}}$ reflects the fact that M being in marking m with $(m, q_{14}) \in \rho$ must be able to receive message acc *and* message rej. The two open nets M_1 and M_2 fulfil the requirements of Def. 6 and thus match with OG_{N_c}.



q0: !cc_y ∨ !cc_n ∨ !req_c
q1: !req_c
q2: (!cc_y ∨ !cc_n ∨ ?r_high) ∧ (!cc_y ∨ !cc_n ∨ ?r_low)
q3: !req_c
q4: (?rej ∨ ?r_high) ∧ (?acc ∨ ?r_low) ∧ (?rej ∨ ?r_low)
q5: !cc_y ∨ !cc_n
q6: (?rej ∨ ?r_high) ∧ (?acc ∨ ?r_low) ∧ (?acc ∨ ?r_high)
q7: !cc_y ∨ !cc_n
q8: ?r_low
q9: ?acc ∧ ?rej
q10: ?r_high ∧ ?r_low
q11: ?acc
q12: ?rej
q13: ?r_high
q14: ?acc ∧ ?rej
q15: final
!cc_n
!req_c
!cc_y
?r_low
?r_high
?acc
?rej

Fig. 2. The operating guideline OG_{N_c} for the credit approval process N_c depicted in Fig. 1(a). For better readability, we add a leading "!" ("?") to a literal x in the graphics of an OG_N if x is an output (input) place of a strategy M for N.

3 Covering Open Net Nodes

The notion of soundness guarantees (among others) the absence of dead transitions in a workflow net. In this section, this idea is adapted to open nets. For an open net N and a set $X \subseteq P_N \cup T_N$ of open net nodes, we will characterize those strategies M for N such that X is *covered* in the composition $M \oplus N$. Here, to cover a place p means that p can be marked in some reachable marking while to cover a transition t means that t is not dead. Such a strategy M is then called a Cover_X*-strategy* for N. Clearly, if X contains all transitions of N, our coverage notion for open nets coincides with soundness, except for the fact that the composition may contain livelocks.

The motivation for dealing with Cover_X-strategies is to figure out if some functionality of a service (i.e. some communication patterns), for example a credit approval, can in principle be used by other services. We further show how to calculate a finite representation of all Cover_X-strategies for N by extending operating guidelines with a global constraint.

3.1 Deciding the Coverage of Open Net Nodes

In this section, we show how a strategy M for N can be discovered as a Cover_X-strategy by just considering the operating guideline of N. In order to define our notion of Cover_X-strategies, we need to define what it means to cover an open net node.

Definition 8 (Cover a place/transition). *Let $N = (P, T, F, I, O, m_0, \Omega)$ be a deadlock-free open net with empty interface ($I = O = \emptyset$), and let $X \subseteq P \cup T$, $p \in P$, and $t \in T$. N* covers *X iff for all $p \in X \cap P$ (for all $t \in X \cap T$) there exists a run of N that includes a marking m with $m(p) \geq 1$ (a t-step).*

Notice that if N covers two nodes, there is not necessarily a run in which both nodes are covered. In the example, transitions $\mathsf{t}_1 - \mathsf{t}_4$, t_6, and $\mathsf{t}_8 - \mathsf{t}_{10}$ are covered in $M_1 \oplus N_c$ and transitions $\mathsf{t}_1 - \mathsf{t}_3$, t_5, t_7, and $\mathsf{t}_9 - \mathsf{t}_{11}$ are covered in $M_2 \oplus N_c$.

The following definition canonically extends strategies to strategies that cover a set X of open net nodes.

Definition 9 ($Cover_X$-strategy). *Let M be a strategy for an open net N, and let $X \subseteq P_N \cup T_N$. M is a $Cover_X$-*strategy *for N iff X is covered in $M \oplus N$. With $Strat_{Cover_X}(N)$ we denote the set of all $Cover_X$-strategies for N.*

For N_c let $X = \{\mathsf{acc}\}$ be given. That means, we are interested whether a credit approval is possible. Then, M_1 and M_2 are $Cover_X$-strategies for N_c. Let $X = \{\mathsf{t}_5, \mathsf{t}_6\}$, that is, we are interested whether it is possible that a credit request has to be examined by an employee if the customer is not fixed in his credit control decision. Then M_1 is a $Cover_X$-strategy for N_c, but M_2 is not (because transitions $\mathsf{t}_5, \mathsf{t}_6$ cannot be enabled in $M_2 \oplus N$).

By definition, every $Cover_X$-strategy for N is also a strategy for N. Obviously, covering open net nodes restricts the set of strategies for N. Thus, we conclude $Strat_{Cover_X}(N) \subseteq Strat(N)$.

In the remainder of this section, we will define some notions and prove some properties of operating guidelines. Based on these properties, we can prove a criterion to decide whether an open net M is a $Cover_X$-strategy for N. We start with the definition of the most permissive strategy for N. This strategy has the least restrictions of all strategies. Thus, the state space of its inner corresponds exactly to the transition system of the underlying automaton of OG_N.

Definition 10 (Most permissive strategy). *Let $OG_N = (Q, C, \delta, q_0, \Phi)$. The* most permissive strategy *for N is the open net $MP_N = (P, T, F, I, O, m_0, \Omega)$ whose behavior corresponds exactly to the transition system (Q, C, δ, q_0) with*
- $P = Q \cup C$,
- $T = \{t_{q_1,c,q_2} \mid (q_1, c, q_2) \in \delta, \text{ with } q_1, q_2 \in Q, c \in C\}$,
- $F = \{(q_1, t_{q_1,c,q_2}), (t_{q_1,c,q_2}, q_2) \mid (q_1, c, q_2) \in \delta\} \cup \begin{cases} (c, t_{q_1,c,q_2}), \text{ if } c \in I; \\ (t_{q_1,c,q_2}, c), \text{ if } c \in O. \end{cases}$,
- $I = O_N$,
- $O = I_N$,
- $m_0 = q_0$, *and*
- $\Omega = \{q \mid c \text{ is in } \Phi(q) \text{ with } c = \mathit{final}\}$.

The resulting open net MP is a state machine. Figure 3 illustrates the construction of the most permissive strategy MP_{N_c} of the operating guideline OG_{N_c} depicted in Fig. 2. As the whole open net would be too big, we depict only the first few nodes.

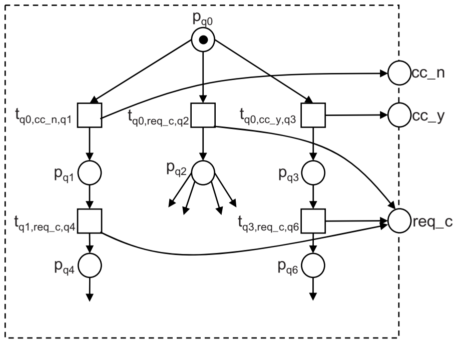


Fig. 3. The initial part of the most permissive strategy MP_{N_c} for N_c which has been constructed according to Def. 10

By the help of the following corollary, we prove that the most permissive strategy MP for N is indeed a strategy for N.

Corollary 1. *The most permissive strategy MP for N is a strategy for N.*

For the proof of this corollary, we rely on a fact about operating guidelines as constructed in [8]. As we cannot repeat the whole approach of [8], we just state this fact without proof.

Proposition 1 ([8]). *For every operating guideline $OG_N = (Q, C, \delta, q_0, \Phi)$ (of some service N) and all $q \in Q$, the formula $\Phi(q)$*

1. *uses only literals c where there is some $q' \in Q$ with $(q, c, q') \in \delta$, and*
2. *is satisfied for the assignment assigning* true *to all literals in $\Phi(q)$.*

Proof (of Corollary 1). Let $OG_N = (Q, C, \delta, q_0, \Phi)$. We construct open net MP as described in Def. 10. Let m_{q_0} be the initial marking of MP. By induction, it can be shown that, for all $q \in Q$, m_q is reached by Def. 6, with $(m_q, q) \in \rho$.

As there is a transition for each $(q, c, q') \in \delta$, we can derive from Prop. 1 that all annotations evaluate to *true* when MP is evaluated according to Def. 6. Consequently, MP matches with OG_N and hence MP is a strategy for N. □

The next definition establishes a connection between markings of an open net N and the inner of a strategy $M \in Strat(N)$. If $inner_M$ is in a marking m, then $K(m)$ (the knowledge that $inner_M$ has about N) is the set of markings of N that N might be in while $inner_M$ is in marking m.

Definition 11 (Knowledge). *Let M be a strategy for an open net N. Let $Mark_{M^*}$ and $Mark_N$ denote the set of all reachable markings of $inner_M$ and N, respectively. Let further m_M denote a marking of M and m_{M^*} denote its restriction to places in $inner_M$. The* knowledge *$K : Mark_{M^*} \to \mathcal{P}(Mark_N)$ that $inner_{MP}$ has about the possible markings of N in marking m_{M^*} is defined by $K(m_{M^*}) = \{m_N \mid (m_M \oplus m_N)$ is reachable from $(m_{0_M} \oplus m_{0_N})\}$.*

For the most permissive strategy MP_{N_c} for N_c (see Fig. 3), we have the following knowledge values:

$$\begin{aligned}
&K([\mathsf{p_{q0}}]) = \{[\mathsf{p_0}]\},\\
&K([\mathsf{p_{q1}}]) = \{[\mathsf{p_0}, \mathsf{cc\_n}]\},\\
&K([\mathsf{p_{q2}}]) = \{[\mathsf{p_0}, \mathsf{req\_c}], [\mathsf{p_1}], [\mathsf{p_2}, \mathsf{r\_high}], [\mathsf{p_3}, \mathsf{r\_low}]\},\\
&K([\mathsf{p_{q3}}]) = \{[\mathsf{p_0}, \mathsf{cc\_y}]\},\\
&K([\mathsf{p_{q4}}]) = \{[\mathsf{p_0}, \mathsf{cc\_n}, \mathsf{req\_c}], [\mathsf{p_1}, \mathsf{cc\_n}], [\mathsf{p_2}, \mathsf{cc\_n}, \mathsf{r\_high}], [\mathsf{p_3}, \mathsf{cc\_n}, \mathsf{r\_low}],\\
&\qquad [\mathsf{p_4}, \mathsf{r\_high}], [\mathsf{p_5}, \mathsf{r\_low}], [\mathsf{p_7}, \mathsf{r\_high}, \mathsf{rej}], [\mathsf{p_7}, \mathsf{r\_low}, \mathsf{acc}], [\mathsf{p_7}, \mathsf{r\_low}, \mathsf{rej}]\}
\end{aligned}$$

The simulation relation ρ used in Def. 6 actually establishes a relation between the knowledge values of the involved states. As the following proposition states, $(m, q) \in \rho$ implies that $K(m) \supseteq K(m_q)$ where m_q is the marking in the most permissive partner that correpsonds to state q of an operating guideline.

Proposition 2 ([5]). *Let M be a strategy for N and MP be the most permissive strategy for N. Let m_q denote the marking in $inner_{MP}$ that corresponds to state $q \in Q$ in OG_N (i.e. $(m_q, q) \in \rho_{MP}$). Let m be reachable in $inner_M$. Then $K(m) = \bigcup_{q:(m,q)\in\rho} K(m_q)$.*

The matching relation ρ relates a marking m of $inner_M$ to a (possible) set of states q of OG_N. Therefore, the knowledge that $inner_M$ has about the possible markings of N in m is equivalent to the union of the knowledge values of all markings m_q of $inner_{MP}$ with $(m_q, q) \in \rho_{MP}$.

The notion of knowledge can be applied to the operating guideline OG_N of N. As every marking m_q in $inner_{MP}$ corresponds to a state q of OG_N, the knowledge OG_N has about N in q is equivalent to the knowledge $inner_{MP}$ has about N in m_q.

Definition 12 (Knowledge in OG). *For an open net N let MP be the most permissive strategy for N and $OG_N = (Q, C, \delta, q_0, \Phi)$. Let $Mark_N$ denote the set of markings of N and m_q be a marking of $inner_{MP}$. The* knowledge *$K : Q \to \mathcal{P}(Mark_N)$ that OG_N has about the possible markings of N in state $q \in Q$ is defined by $K(q) = K(m_q)$.*

The following theorem presents a way to decide, on the basis of an operating guideline, whether a strategy M for N is also a $Cover_X$-strategy for N.

Theorem 1 (Place/Transition coverability). *Let M be a strategy for open net N. A place $p \in P_N$ (a transition $t \in T_N$) is covered in $M \oplus N$ iff there is a state $q \in Q$ of OG_N, a marking m_M in $inner_M$, and a marking $m_N \in K(q)$ with $(m_M, q) \in \rho$, and $m_N(p) \geq 1$ (t is enabled in m_N).*

Proof. We present the proof for the case of a covered transition only. The case of a covered place is analogous.

($\Rightarrow$) Let N, OG_N, and $M \in Strat(N)$ be given and let transition t be covered in $M \oplus N$. Then, according to Def. 8, there is a run $m_{0_{M\oplus N}} \xrightarrow{t_1} \ldots \xrightarrow{t_n} m_{M\oplus N} \xrightarrow{t} m'_{M\oplus N}$ in $M \oplus N$, $m'_{M\oplus N}(p) \geq 1$. Let m_M and m_N be the restrictions of marking

$m_{M \oplus N}$ to places in $inner_M$ and N, respectively. As t is a transition of N, t is enabled in m_N as well. By Def. 11, we have $m_N \in K(m_M)$. By Proposition 2, there must be a state q in OG_N where $m_N \in K(q)$ and hence the implication of this theorem holds.

($\Leftarrow$) Let N be an open net and $OG_N = (Q, C, \delta, q_0, \Phi)$. Let M be a strategy for N. Since M is a strategy for N, there is a matching relation ρ of states in Q and markings in $inner_M$. Let m_M, q, and m_N be as assumed. Thus, $(m_M, q) \in \rho$, $m_N \in K(q)$, and t is enabled in m_N. From Proposition 2 follows $m_N \in K(m_M)$. Consequently, there is a run in $M \oplus N$ that reaches $m_M \oplus m_N$ which can be extended by an occurrence of t since activation of t in m_N implies activation of t in $m_M \oplus m_N$. Since every run in $M \oplus N$ is deadlock-free (follows from M being a strategy for N), we can conclude that the considered run is deadlock-free, too. So there exists a deadlock-free run in $M \oplus N$ where t is covered and hence the replication of this theorem holds. $\square$

The value of Theorem 1 is that it gives us a criterion to check whether an open net node is covered or not. A place p of N is covered by a strategy for N if there is a state q in OG_N and the knowledge in q contains a marking of N where p is marked. A transition t of N is covered by a strategy for N if there is a state q and the knowledge in q contains a marking m of N where t is enabled. That way, it is easily possible to annotate each state q of OG_N with all places and transitions which are covered in q. This can be done during the calculation of the operating guideline.

As an example, based on the knowledge values $K([\mathsf{p}_{\mathsf{q}0}]) - K([\mathsf{p}_{\mathsf{q}4}])$ we presented above we can derive the following sets of nodes of N_c that are covered in states $\mathsf{q}_0 - \mathsf{q}_4$ of OG_{N_c}:

$\mathsf{q}_0 : \{\mathsf{p}_0\}$
$\mathsf{q}_1 : \{\mathsf{p}_0, \mathsf{cc\_n}\}$
$\mathsf{q}_2 : \{\mathsf{p}_0 - \mathsf{p}_3, \mathsf{req\_c}, \mathsf{r\_high}, \mathsf{r\_low}, \mathsf{t}_1 - \mathsf{t}_3\}$
$\mathsf{q}_3 : \{\mathsf{p}_0, \mathsf{cc\_y}\}$
$\mathsf{q}_4 : \{\mathsf{p}_0 - \mathsf{p}_5, \mathsf{p}_7, \mathsf{cc\_n}, \mathsf{cc\_y}, \mathsf{req\_c}, \mathsf{r\_high}, \mathsf{r\_low}, \mathsf{acc}, \mathsf{rej}, \mathsf{t}_1 - \mathsf{t}_4, \mathsf{t}_6, \mathsf{t}_8 - \mathsf{t}_{10}\}$

3.2 A Finite Representation of All *Cover*$_X$-Strategies

In this section, we introduce a notion of an operating guideline with a global constraint as a representation of all *Cover*$_X$-strategies for N. We further present an algorithm for deciding when an open net M matches with such an operating guideline.

Consider again our running example N_c in Fig. 1(a). Assume we want to cover $X = \{\mathsf{acc}\}$ in N_c, that is, we are interested in strategies in which a customer may receive an approval for his credit request. We have $[\mathsf{acc}] \in K(\mathsf{q}_4), K(\mathsf{q}_6), K(\mathsf{q}_9), K(\mathsf{q}_{11}), K(\mathsf{q}_{14})$. So according to Theorem 1, a strategy M for N_c is a *Cover*$_X$-strategy for N_c if it has at least a marking m_{acc} of $inner_M$ that matches with q_4, q_6, q_9, q_{11}, or q_{14}. As a second example assume $X = \{\mathsf{t}_5, \mathsf{t}_6\}$, that is, we are interested in strategies in which a customer is not fixed in his credit control decision and the credit request can be examined by an employee. In that case we have $[\mathsf{t}_5] \in K(\mathsf{q}_6), K(\mathsf{q}_{14})$ and $[\mathsf{t}_6] \in K(\mathsf{q}_4), K(\mathsf{q}_9)$. So M is a Cover_X-strategy for N_c if it has at least a marking m_{t5} of inner_M that matches with q_6 or q_{14} and it has a marking m_{t6} of inner_M that matches with q_4 or q_9.

The examples illustrate that is in general not possible to express the constraints for covering open net nodes in the shape of local annotations in each state of the operating guideline. Consequently, the present concept of an annotated automata fails at representing all Cover_X-strategies of N. To overcome this problem, we propose another representation of all Cover_X-strategies of N that takes the non-locality of covering open net nodes into account. To this end, we will slightly enhance the concept of an operating guideline.

Consider again the example above. Since OG_{N_c} (see Fig. 2) represents all strategies and every Cover_X-strategy for N_c is a strategy for N_c, we have to restrict OG_{N_c} to Cover_X-strategies. This can be achieved by a *global constraint* specifying that, for every open net node $x \in X$ to be covered, at least one state q in OG_{N_c} with $x \in K(q)$ must be present in the matching relation between OG_{N_c} and a Cover_X-strategy. This constraint can be expressed as a Boolean formula ψ_X.

In the following, we formalize annotated automata enhanced with a global constraint and define the matching relation between an open net and such an annotated automaton.

Definition 13 (Annotated automaton with global constraint). *Let $A^{\Phi} = (Q, C, \delta, q_0, \Phi)$ be an annotated automaton and ψ be a Boolean formula with propositions taken from the set Q. Then, $A^{\Phi,\psi} = (A^{\Phi}, \psi)$ is an* annotated automaton with global constraint *ψ.*

As an example for a global constraint to OG_{N_c}, consider $\psi = (\mathsf{q}_6 \vee \mathsf{q}_{14}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_9)$. This formula is satisfied if and only if true is assigned to sufficiently many states to cover set $X = \{\mathsf{t}_5, \mathsf{t}_6\}$.

Enhancing an annotated automaton with a global constraint makes it necessary to redefine the matching relation of an open net M with an annotated automaton. M matches with an annotated automaton with global constraint $A^{\Phi,\psi}$ if it matches with the annotated automaton A^{Φ}, and in addition satisfies ψ.

Definition 14 (Matching with $A^{\Phi,\psi}$). *Let M be an open net, and let $A^{\Phi,\psi}$ be an annotated automaton A^{Φ} with global constraint ψ. M matches with $A^{\Phi,\psi}$ iff M matches with A^{Φ} using relation ρ and ψ evaluates to true in the assignment $\gamma_M : Q_A \to \{\mathit{true}, \mathit{false}\}$ where $\gamma_M(q) = \mathit{true}$ iff there is a marking m of M such that $(m, q) \in \rho$.*

Finally, we are ready to construct the operating guideline with global constraint $OG_{\psi_X}(N)$ of an open net N as a representation of the set $\mathit{Strat}_{\mathit{Cover}_X}(N)$ of all Cover_X-strategies for N.

Definition 15 (Global constraint for covering X). *Let N be an open net and OG_N an operating guideline of N. Let $X \subseteq P_N \cup T_N$. For a place $p \in P$, let $p \sim q$ iff there is an $m \in K(q)$ where $m(p) > 0$. For a transition $t \in T$, let $t \sim q$ iff there is an $m \in K(q)$ where t is enabled. Then ψ_X is the formula*

$$\bigwedge_{x : x \in X} \bigvee_{q : x \sim q} q$$

$OG_{\psi_X}(N) = (OG_N, \psi_X)$ defines an operating guideline with global constraint *of N.*

As a direct consequence of Theorem 1, we obtain the main result of this section, that is, $OG_{\psi_X}(N)$ represents all $Cover_X$-strategies for N.

Theorem 2. *M is a $Cover_X$-strategy for N iff M matches with $OG_{\psi_X}(N)$.*

The operating guideline representing all $Cover_X$-strategies for N_c with $X = \{\mathsf{t}_5, \mathsf{t}_6\}$ is the operating guideline $OG_{\psi_X}(N_c) = (OG_{N_c}, \psi_X)$ where $\psi = (\mathsf{q}_6 \vee \mathsf{q}_{14}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_9)$ as stated above. If we consider again open nets M_1 and M_2 (which are both strategies for N_c), then we get that M_1 matches with $OG_{\psi_X}(N_c)$ and it is hence a $Cover_X$-strategy for N_c. In contrast, M_2 does not match with $OG_{\psi_X}(N_c)$, because it does not satisfy the global constraint. More precisely, there is no marking in $inner_{M_2}$ that matches with any of the nodes q_4, q_6, q_9, and q_{14}.

As another example, let $X = \{\mathsf{t}_1, \ldots, \mathsf{t}_{11}\}$, meaning all transitions of N_c should not be dead in $M \oplus N$. Then, $OG_{\psi_X}(N_c)$ has the following global constraint:

$$\begin{aligned} \psi_X = \quad & (\mathsf{q}_2 \vee \mathsf{q}_4 \vee \mathsf{q}_6) \wedge (\mathsf{q}_2 \vee \mathsf{q}_4 \vee \mathsf{q}_6) \wedge (\mathsf{q}_2 \vee \mathsf{q}_4 \vee \mathsf{q}_6) \wedge (\mathsf{q}_4 \vee \mathsf{q}_{12}) \\ & \wedge (\mathsf{q}_6 \vee \mathsf{q}_{14}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_9) \wedge (\mathsf{q}_6 \vee \mathsf{q}_{11}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_{12}) \\ & \wedge (\mathsf{q}_4 \vee \mathsf{q}_6 \vee \mathsf{q}_9 \vee \mathsf{q}_{14}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_6 \vee \mathsf{q}_9 \vee \mathsf{q}_{14}) \wedge (\mathsf{q}_6 \vee \mathsf{q}_{11}) \end{aligned}$$

which is equivalent to

$$\psi_X = (\mathsf{q}_2 \vee \mathsf{q}_4 \vee \mathsf{q}_6) \wedge (\mathsf{q}_4 \vee \mathsf{q}_{12}) \wedge (\mathsf{q}_6 \vee \mathsf{q}_{11}) \wedge (\mathsf{q}_4 \vee \mathsf{q}_6 \vee \mathsf{q}_9 \vee \mathsf{q}_{14}).$$

3.3 Discussion

In the following we will compare (ordinary) operating guidelines and operating guidelines with global constraint. We further discuss some complexity issues.

Comparing an operating guideline OG_N for N and an operating guideline with global constraint $OG_{\psi_X}(N)$ for N, we identify that both operating guidelines have the same underlying automaton. This is caused by the fact that each $Cover_X$-strategy for N is also a strategy for N. Furthermore, if the most permissive strategy for N is not a $Cover_X$-strategy for N, then the set of $Cover_X$-strategies is empty.

Computing OG_N is proportional in time to the product of the number of states of N and an over-approximation of its most permissive strategy [9]. For

$OG_{\psi_X}(N)$ the time complexity does not change, because all information necessary for annotating the states $q \in Q$ with the nodes of N and setting up the global constraint have to be computed for OG_N anyway. In order to increase efficiency, it is sufficient to annotate each state q only with open net nodes of X.

The space complexity of OG_N is proportional to the product of the number of states of N and its most permissive strategy [9]. If we compute $OG_{\psi_X}(N)$, then this complexity increases due to ψ_X. The global constraint is a conjunction of at most $|X|$ disjunctions where each disjunction may consist of at most $|Q|$ literals. Hence, the size of the global constraint is at most $O(|X| \cdot |Q|)$. The example suggests that the size of the global constraint will be much smaller in practice.

Although the time and space complexity of OG_N is high, experimental results have shown that the calculation of OG_N is feasible in practical applications both for time and space (see [5], for instance). Based on the complexity considerations for $OG_{\psi_X}(N)$ we conclude that the calculation of $OG_{\psi_X}(N)$ will be feasible in practical applications, too.

Matching an open net M with OG_N is proportional in time to the number of states in $M \oplus N$ [9]. If we match M with $OG_{\psi_X}(N)$, we additionally have to check whether the global constraint is satisfied by the assignment γ_M. This can be done in linear time w.r.t. the size of the constraint.

As the space complexity and the matching complexity for the proposed notion of operating guidelines with global constraint only marginally increase in comparison with ordinary operating guidelines, we can conclude that this novel notion is a well-suited instrument for service composition.

4 Related Work

The work presented in this paper is mainly inspired by the notion of soundness for workflow nets [2]. Soundness guarantees the absence of deadlocks, livelocks, and dead transitions. In this paper, we adopt the idea of soundness to our service model open nets. Given an open net N, we are interested in all open nets M such that the composition $M \oplus N$ is deadlock-free and certain places and transitions of N are covered. Here, cover means that these places can be marked and the transitions are not dead in $M \oplus N$. So far in contrast to soundness, our approach is limited in the sense that the composed system may contain livelocks.

There is also some relation to the research fields testing and computer-aided verification. In these fields testing/checking the coverage of certain activities is also a known and important sanity check, see [11], for instance.

Besides the relation to soundness, covering open net nodes can also be seen as a behavioral constraint for services. In [6] the authors introduced two kinds of behavioral constraints: to *enforce* and to *exclude* a set of open net nodes. A strategy M for N enforces (excludes) a transition t of N if every (no) run in $M \oplus N$ includes a t-step. Covering a transition t is thus equivalent to *not exclude* t. However, cover cannot be expressed by the approach proposed in [6], because the authors model a constraint as a constraint open net C and compose C and N.

5 Conclusion

We proposed an approach to guarantee the coverage of certain activities in services. Our approach is inspired by the notion of soundness on the one hand and by the work on behavioral constraints for services on the other hand. We have shown that with the operating guideline of a service N we can decide if a partner service M is designed in a way such that a given set of activities of N can be covered in the composition of M and N. We further presented a finite representation of all partner services M by extending our notion of an operating guideline with a global constraint. The results presented in this paper have been implemented in our analysis tool Fiona[1] [10]. The proposed approach can also be applied to industrial service models specified in WS-BPEL [12]. To this end the compiler BPEL2oWFN[2] [10] can be used to translate such a WS-BPEL process into an open net. The resulting net can then be analyzed by Fiona.

The advantage of the proposed approach is that the global constraint can be calculated based on the information that is already present when calculating the operating guideline. In addition, the global constraint does only marginally increase the complexity of matching a service with the operating guideline with global constraint. Thus operating guidelines with global constraint are a well-suited instrument for service composition.

In ongoing work plan to deal with a stricter correctness criterion that also excludes livelocks. That way, we can close the gap to soundness. Furthermore, an open net N can be substituted by an open net N' w.r.t. the coverage of X if and only if every $Cover_X$-strategy for N is also a $Cover_X$-strategy for N' (i.e. $Strat_{Cover_X}(N') \supseteq Strat_{Cover_X}(N)$). To this end we are working on an algorithm to automatically decide substitutability w.r.t. the coverage of X.

Acknowledgements. The authors wish like to thank Robert Danitz and Janine Ott for their work on the implementation of Fiona. Christian Stahl is funded by the DFG project "Substitutability of Services" (RE 834/16-1). Karsten Wolf is supported by the DFG within grant "Operating Guidelines for Services" (WO 1466/8-1).

References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (2007)
2. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
3. van der Aalst, W.M.P., Weske, M.: The P2P approach to Interorganizational Workflows. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–156. Springer, Heidelberg (2001)
4. Leymann, F., Roller, D., Schmidt, M.T.: Web services and business process management. IBM Systems Journal 41(2), 198–211 (2002)

[1] Available at `http://www.service-technology.org/fiona`
[2] Available at `http://www.service-technology.org/bpel2owfn`

5. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
6. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer, Heidelberg (2007)
7. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer, Heidelberg (1985)
8. Massuthe, P., Schmidt, K.: Operating Guidelines – an Automata-Theoretic Foundation for the Service-Oriented Architecture. In: Cai, K., Ohnishi, A., Lau, M. (eds.) Proceedings of the Fifth International Conference on Quality Software (QSIC 2005), Melbourne, Australia, pp. 452–457. IEEE Computer Society, Los Alamitos (2005)
9. Massuthe, P., Wolf, K.: An Algorithm for Matching Non-deterministic Services with Operating Guidelines. International Journal of Business Process Integration and Management (IJBPIM) 2(2), 81–90 (2007)
10. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
11. Kupferman, O.: Sanity Checks in Formal Verification. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 37–51. Springer, Heidelberg (2006)
12. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0, April 11 2007. OASIS Standard, OASIS (2007)

Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance

Niels Lohmann

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

Abstract. Many work has been conducted to analyze service choreographies to assert manyfold correctness criteria. While errors can be *detected* automatically, the *correction* of defective services is usually done manually. This paper introduces a graph-based approach to calculate the minimal edit distance between a given defective service and synthesized correct services. This edit distance helps to automatically fix found errors while keeping the rest of the service untouched. A prototypic implementation shows that the approach is applicable to real-life services.

Keywords: Choreographies, graph correction, correction of services, verification of services, service automata, operating guidelines, BPEL.

1 Introduction

In service-oriented computing [1], the correct interplay of distributed services is crucial to achieve a common goal. Choreographies [2] are a means to document and model the complex global interactions between services of different partners. BPEL4Chor [3] has been introduced to use BPEL [4] to describe and execute choreographies. Recently, a formal semantics for BPEL4Chor was introduced [5], offering tools and techniques to verify BPEL-based choreographies.

Whereas it is already possible to automatically *check* choreographies for deadlocks or to synthesize participant services [6], no work was conducted in supporting the *fixing* of existing choreographies. This is especially crucial, because fixing incorrect services is usually cheaper and takes less time than re-designing and implemeting a correct service from scratch. In addition, information on how to adjust an existing service can help the designers understand the error more easily compared to confronting them with a whole new synthesized service.

As the running example for this paper, consider the example choreography visualized in BPMN [7] in Fig. 1. It describes the interplay of a travel agency, a customer service, and an airline reservation system. The travel agency sends an offer to the client which either rejects it or books a trip. In the latter case, the travel agency orders a ticket at the airline service which either sends a confirmation or a refusal message to the customer. The choreography contains a design flaw as the customer service does not receive the refusal message. This leads to a deadlock in case the airline refuses the ticket order.

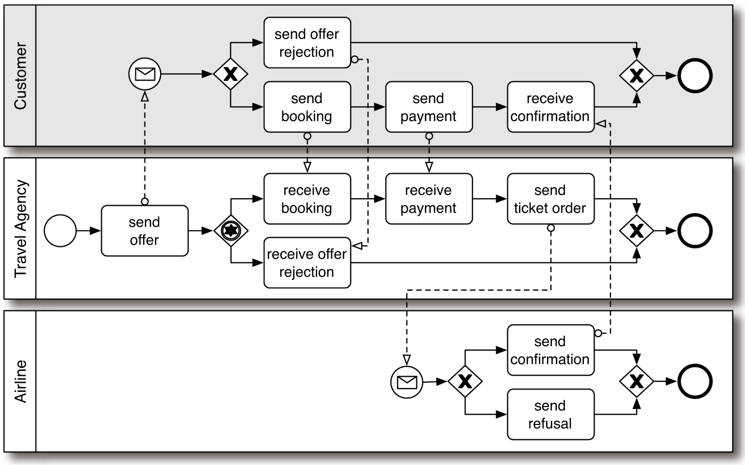


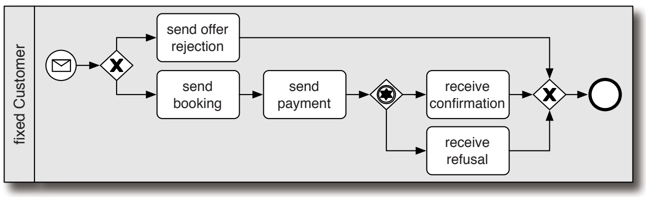
Fig. 1. Choreography between travel agency, airline, and customer. The choreography can deadlock, because the customer does not receive a refusal message from the airline.

This deadlock can be detected using state-of-the-art model checking tools which provide a trace to the deadlocking state. In the concrete example, a trace would be (send offer, receive offer, send booking, send payment[1], receive booking, receive payment, send ticket order, receive ticket order, send refusal). This trace, however, gives no information which service has to be changed in which manner to avoid the deadlock. Thus, an iteration of manual corrections followed by further deadlock checks is necessary to finally remove the deadlock. Though it is obvious how to correct the flawed example, the manual correction of choreographies of a larger number of more complex services is tedious, if not impossible.

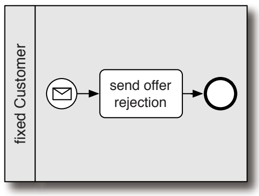
Moreover, even for this simple choreography exists a variety of possibilities to fix the customer's service. Figure 2 depicts two possible corrections to avoid the deadlock. Though both services would avoid the choreography to deadlock, the service in Fig. 2(a) is to be preferred over that in Fig. 2(b) as it is "more similar" to the original service. Though this preference is psychological and is unlikely to be proven formally, the usage of similarities is widley accepted (cf. [8]). The tool chain presented in [6,5] synthesizes a participant service independently of an existing incorrect service and might produce correct, yet unintuitive results such as the service in Fig. 2(b).

The goal of this paper is to formalize, systematize, and to some extend automatize the fixing of choreographies as it has been illustrated above. We thereby combine existing work on characterizing all correctly interacting partners of a service with similarity measures and edit distances known in the field of graph correction. These approaches are recalled in Sect. 2 and 3. In Sect. 4, we define an edit distance that aims at finding the *most similar* service from the set of all fitting services. To support the modeler, we further derive the required edit

[1] We assume asynchronous (i. e., non-blocking) communication.



(a) add receipt of refusal message



(b) delete booking branch

Fig. 2. Two possible corrections of the customer service to achieve deadlock freedom

actions needed to correct the originally incorrect service. In Sect. 5, we present experimental results conducted with a proof of concept implementation. Section 6 discusses related work. Finally, Sect. 7 is dedicated to a conclusion and gives directions for future research.

2 Service Models

2.1 Service Automata and Operating Guidelines

To formally analyze services, a sound mathematical model is needed. In the area of workflows and services, Petri nets are a widely accepted formalism [9]. They combine a graphical notation with a variety of analysis methods and tools. For real-life service description languages such as BPEL or BPEL4Chor exists a feature-complete Petri net semantics [10,5]. To simplify the presentation, we abstract from the structure of a service and complex aspects such as data or fault handling, and focus on the external behavior (also known as the *business protocol*) of services in this paper. To this end, we use service automata [11] to model the external behavior services.[2]

A *service automaton* is a finite automaton with a set Q of states, a set $F \subseteq Q$ of final states, an initial state $q_0 \in Q$, an interface I for asynchronous message passing, and a partial transition function $\delta : Q \times I \to Q$. In this paper, we only consider deterministic service automata and require that final states are sink states; that is, have no outgoing transitions. For $\delta(q, a) = q'$ we also write $q \xrightarrow{a} q'$. Throughout this paper, we use $\mathcal{S}$ to denote service automata. We further assume that all services in this paper share a common interface I. This common interface can be achieved by joining all participants' interfaces.

Figure 3(a) depicts a service automaton modeling the external behavior of the customer service of Fig. 1. The edges are labeled with messages sent to (preceded with "!") or received from (preceded with "?") the environment: The interface of the service automaton is $\{!\mathsf{booking}, ?\mathsf{confirmation}, ?\mathsf{offer}, !\mathsf{payment}, !\mathsf{rejection}\}$.

[2] Due to the close relationship (cf. [12]) between Petri nets and automata, there exist techniques to transform back and forth between the two formalisms.

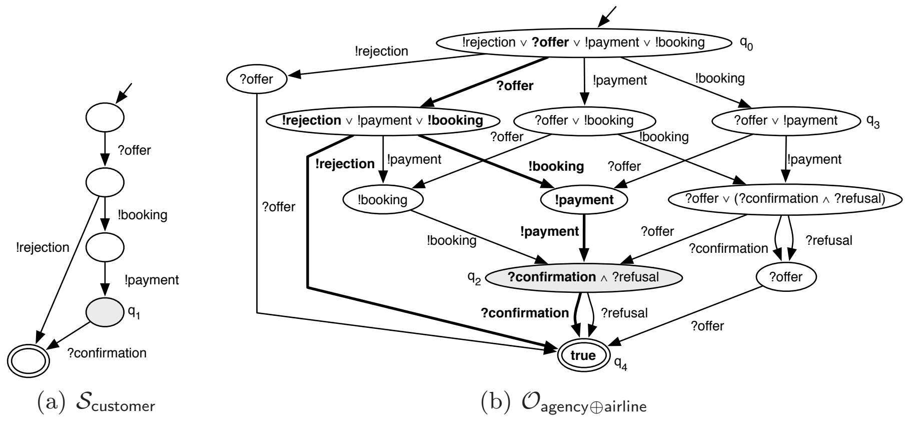


Fig. 3. A service automaton $\mathcal{S}_{\text{customer}}$ modeling the customer service of Fig. 1 (a) and the operating guideline $\mathcal{O}_{\text{agency}\oplus\text{airline}}$ of the composition of travel agency and airline service (b). The service automaton does not match the OG, because q_2's formula is not satisfied.

As services are usually not considered in isolation, their interplay has to be taken into account in verification. A necessary correctness criterion is *controllability* [13]. A service $\mathcal{S}$ is controllable if there exists a partner service $\mathcal{S}'$ such that their composition $\mathcal{S} \oplus \mathcal{S}'$ (i. e., the choreography of the service and the partner) is free of deadlocks.

Controllability can be decided constructively: If a correctly interacting partner service for $\mathcal{S}$ exists, it can be automatically synthesized [13,6]. Furthermore, it has been proven that there exists one distinguished partner service $\mathcal{S}^*$ that is *most permissive*; that is, it simulates any other correctly interacting partner service. The converse does, however, not hold; not every simulated service is a correct partner service itself. To this end, the most permissive partner service can be annotated with Boolean formulae expressing which states and transitions can be omitted and which parts are mandatory. This annotated most permissive partner service is called an *operating guideline* (OG) [11]. We denote OGs with $\mathcal{O}$ and use $\varphi(q)$ to denote the Boolean formula annotated to state q of the OG.

Figure 3(b) depicts the OG of the composition of the travel agency and the airline. The disjunction of the OG's initial q_0 state means that a partner service must send a rejection, receive an offer, send a payment or send a booking in its initial state. This is possible due to asynchronous communication. The service automaton of Fig. 3(a) is simulated by the OG and fulfills all but one formula (satisfied literals are depicted bold in Fig. 3(b)). It does not satisfy the formula $\varphi(\mathsf{q}_2) = (\mathsf{?confirmation} \wedge \mathsf{?refusal})$ of the OG's state q_2, because the service automaton does not receive a refusal message in the simulated state q_1.

2.2 Fixing Deadlocking Choreographies

Consider a deadlocking choreography of n participants, $\mathcal{S}_1 \oplus \cdots \oplus \mathcal{S}_n$. As mentioned earlier, a deadlock trace usually does not give sufficient information *how* to fix *which* service to achieve deadlock freedom. To find a candidate service that can be changed such that whole choreography is deadlock-free, we can perform the following steps:

– Firstly, we check for each service the necessary correctness criterion: If a service taken for itself is not controllable, then there exists no environment in which that service runs correctly — especially not the choreography under consideration. In that case, that service has to be radically overworked towards controllability, which is not topic of this paper.
– Secondly, we remove one participant, say $\mathcal{S}_i$. The resulting choreography $Chor_i = \mathcal{S}_1 \oplus \cdots \oplus \mathcal{S}_{i-1} \oplus \mathcal{S}_{i+1} \oplus \cdots \oplus \mathcal{S}_n$ can be considered as one large service with an interface. If it is controllable, then there exists a service $\mathcal{S}'_i$ which interacts deadlock-freely with the other participants of the choreography; that is, $Chor_i \oplus \mathcal{S}'_i$ is deadlock-free. In [5], a complete tool chain for this participant synthesis was presented for BPEL-based choreographies.

As motivated in the introduction, the mere replacement of $\mathcal{S}_i$ by $\mathcal{S}'_i$ is not desirable, because $\mathcal{S}'_i$ totally ignores the structure of $\mathcal{S}_i$ and might be very different to the original, yet incorrect service $\mathcal{S}_i$. Instead of synthesizing *any* fitting service (such as the service in Fig. 2(b)), we are interested in a corrected service that is most similar to $\mathcal{S}_i$. To this end, we can use the OG of $Chor_i$, because it characterizes the set of *all* fitting partners. Figure 4 illustrates this.

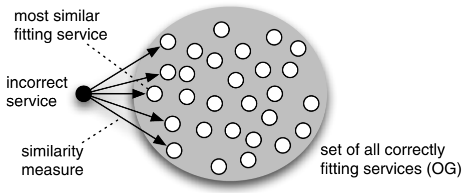


Fig. 4. The OG as characterization of all correct services can be used to find the most similar service

Beside the corrected services of Fig. 2, the OG characterizes 2002 additional (acyclic and deterministic) partner services.[3] Though all are correct, we are interested in the service that most similar to the incorrect customer service. Instead of iteratively check all candidates, we will define a similarity measure that exploits the OG's compact representation to efficiently find the desired service of Fig. 2(a).

[3] The set of cyclic or nondeterministic partner services might be infinite.

3 Graph Similarities

Graph similarities are widely used in many fields of computer science, for example for pattern recognition [14] or in bio informatics. Cost-based distance measures adapt the *edit distance* known from string comparison [15,16] to compare labeled graphs (e. g., [17]). They aim at finding the minimal number of modifications (i. e., adding, deleting, and modifying nodes and edges) needed to achieve a graph isomorphism.

Distance measures aiming at graph isomorphism have the drawback that they are solely based on the *structure* of the graphs. They focus on the syntax of the graphs rather than their semantics. When a graph (e. g., a service automaton) models the *behavior* of a system, similarity of graphs should focus on simulation of behavior rather than on a high structural similarity. Figure 5 illustrates that structural and behavioral similarity is not necessarily related.

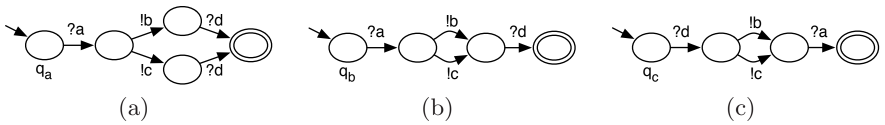


Fig. 5. Service automata (a) and (b) simulate each other, but have an unsimilar structure. Service automata (b) and (c) have a very similar structure, but rather unsimilar behavior.

Sokolsky et el. [18] address this problem (a similar approach is presented in [19]), motivated by finding computer viruses by comparing a program with a library of control flow graphs. In that setting, classical simulation is too strict, because two systems that are equal in all but one edge label *behave* very similar, but there exists no simulation relation between them. To this end, Sokolsky et al. introduce a *weighted quantitative simulation* function to compare states of two graphs. Whenever the two graphs cannot perform a transition with same labels, one graph performs a special stuttering step ε, which is similar to τ-steps in stuttering bisimulation [20]. To "penalize" stuttering, a label similarity function assings low similarity between ε and any other label. This label similarity function $L : (I \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\}) \to [0, 1]$ assigns a value that expresses the similarity between the labels of the service automata[4] under consideration. For example, $L(?\mathsf{a}, ?\mathsf{b})$ describes the similarity of a ?a-labeled transition of service automaton $\mathcal{S}_1$ and a ?b-labeled transition of service automaton $\mathcal{S}_2$. Furthermore, a discount factor $p \in [0, 1]$ describes the local importance of similarity compared to the similarity of successor states.

Definition 1 (Weighted quantitative simulation, [18]). *Let* $\mathcal{S}_1 = [Q_1, \delta_1, F_1, q_{0_1}, I]$, $\mathcal{S}_2 = [Q_2, \delta_2, F_2, q_{0_2}, I]$ *be service automata. A* weighted quantitative simulation *is a function* $S : Q_1 \times Q_2 \to [0, 1]$*, such that:*

[4] We adjusted the definitions of to service automata. The original definition in [18] bases on labeled directed graphs. We do not consider node labels in this paper.

$$S(q_1, q_2) = \begin{cases} 1, & \textit{if } q_1 \in F_1, \\ (1-p) + \max(W_1(q_1,q_2), W_2(q_1,q_2)), & \textit{otherwise,} \end{cases}$$

$$W_1(q_1, q_2) = \max_{q_2 \xrightarrow{b} q_2'} \Big(L(\varepsilon, b) \cdot S(q_1, q_2') \Big),$$

$$W_2(q_1, q_2) = \frac{p}{n} \cdot \sum_{q_1 \xrightarrow{a} q_1'} \max \left(L(a, \varepsilon) \cdot S(q_1', q_2), \max_{q_2 \xrightarrow{b} q_2'} (L(a,b) \cdot S(q_1', q_2')) \right),$$

and n is the number of edges leaving q_1.

The weighted quantitative simulation function S recursively compares the states from the two service automata and finds the maximal similar edges. Thereby, W_1 describes the similarity gain by stuttering of graph $\mathcal{S}_1$ and W_2 the tradeoff between simultaneous transitions of $\mathcal{S}_1$ and $\mathcal{S}_2$ and stuttering of graph $\mathcal{S}_2$. Both, the discount factor p and the label similarity function L, can be chosen freely to adjust the result of the similarity algorithm. The choice of the parameters is, however, out of scope of this paper.

As an example, consider the service automata in Fig. 5 and assume a discount factor $p = 0.5$ and a label similarity function L that assigns 1.0 to equal labels and 0.5 to any other label pair. Then $S(\mathsf{q_a}, \mathsf{q_b}) = 1.0$ (the weighted quantitative simulation is a generalization of the classical simulation) and $S(\mathsf{q_b}, \mathsf{q_c}) = 0.75$ which indicates the differences in the behaviors.

4 A Matching-Based Edit Distance

The algorithm to calculate weighted quantitative simulation can be used as a similarity measure for service automata or OGs, but has two drawbacks: Firstly, it is not an edit distance. It calculates a value that expresses the similarity between the service automata, but gives no information about the modification actions needed to *achieve* simulation. Secondly, it does not take formulae of the OG into account. Therefore, a high similarity between a service automaton and an OG would not guarantee deadlock freedom as the example of Fig. 3 demonstrates: The service automaton of the customer is perfectly simulated by the OG but the overall choreography deadlocks.

4.1 Simulation-Based Edit Distance

Before we consider the OG's formulae, we show how the similarity result of the algorithm of [18] can transformed into an edit distance. Given two states q_1 and q_2, Def. 1 determines the best simulation between the transitions of q_1 and q_2. In addition, one service automaton can stutter (i. e., remain in the same state). The weighted quantitative simulation function calculates the best label matching to maximize the similarity between the root nodes of the service automata. From the transition pairs belonging to the maximum, we can derive according edit actions (cf. Table 1).

Table 1. Deriving edit actions from transition pairs of Def. 1

transition of $\mathcal{S}_1$	transition of $\mathcal{S}_2$	resulting edit action	similarity
a	a	keep transition a	$L(\mathsf{a},\mathsf{a})$
a	b	modify transition a to b	$L(\mathsf{a},\mathsf{b})$
a	ε (stutter)	delete transition a	$L(\mathsf{a},\varepsilon)$
ε (stutter)	a	insert transition a	$L(\varepsilon,\mathsf{a})$

These edit actions define basic edit actions whose similarity is determined by the edge similarity function L. To simplify the representation of a large number of edit actions, the basic edit actions may be grouped to macros to express more complex operations such as swapping or moving of edges and nodes, duplicating of subgraphs, or partial unfolding of loops.

The simulation-based edit distance does not respect the OG's formulae. One possibility to achieve a matching would be to first calculate the most similar simulating service using the edit distance for Def. 1 and then to simply add and remove all nodes and edges necessary in a second step. Using the weighted quantitative simulation function of Def. 1, the resulting edit actions (cf. Table 1) simply inserts or removes edges to present nodes rather than to new nodes. This approach does in general not work to achieve matching with an OG. See Fig. 6 for a counterexample. However, also the insertion of nodes would not determine the most similar partner service, because this may result in sub-optimal solutions as Fig. 7 illustrates.

4.2 Combining Formula-Checking and Graph Similarity

Due to the suboptimal results achieved by a-posteriori formula satisfaction by node insertion, we need to modify the algorithm of [18] not to statically take the outgoing transitions of an OG's state into account, but also check any formula-fulfilling subset of outgoing transitions. Therefore, we need some additional definitions to base formula satisfaction and to cover the dynamic presence of OG transitions.

Definition 2 (Satisfying label set, label permutation). *Let* $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ *be a service automaton and* $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ *an OG, and let* $q_1 \in Q_\mathcal{S}$ *and* $q_2 \in Q_\mathcal{O}$*.*

- *Define* $Sat(\varphi(q_2)) \subseteq \mathcal{P}(I \cap \{b \mid \exists q_2' \in Q_\mathcal{O} : q_2 \xrightarrow{b} q_2'\})$ *to be the set of all sets of labels of transitions leaving* q_2 *that satisfy formula* φ *of state* q_2*.*
- *For* $\beta \in Sat(\varphi(q_2))$*, define* $perm(q_1, q_2, \beta) \subsetneq \big((I \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\})\big)$ *to be a label permutation of* q_1*,* q_2 *and* β *such that:*
  (a) if $q_1 \xrightarrow{a} q_1'$*, then* $(a, c) \in perm(q_1, q_2, \beta)$ *for a label* $c \in \beta \cup \{\varepsilon\}$*,*
  (b) if $q_2 \xrightarrow{b} q_2'$ *and* $b \in \beta$*, then* $(d, b) \in perm(q_1, q_2, \beta)$ *for a label* $d \in I \cup \{\varepsilon\}$*,*
  (c) $(\varepsilon, \varepsilon) \notin perm(q_1, q_2, \beta)$*, and*
  (d) if $(a, b) \in perm(q_1, q_2, \beta)$*, then* $(a, c), (d, b) \notin perm(q_1, q_2, \beta)$ *for all labels* $c \in \beta \cup \{\varepsilon\}$ *and all labels* $d \in I \cup \{\varepsilon\}$*.*
- *Define* $Perms(q_1, q_2, \beta)$ *to be the set of all label permutations of* q_1*,* q_2 *and* β*.*

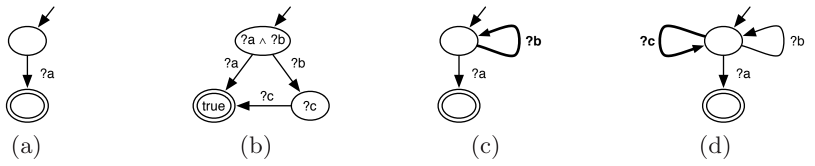


Fig. 6. Matching cannot be achieved solely by transition insertion. The service automaton (a) does not match with the OG (b) because of a missing ?b-branch. In service automaton (c), a loop edge was inserted. However, the state reached by ?b in the OG requires a ?c-branch to be present. After inserting this edge (d), the resulting service automaton is not simulated by the OG (b).

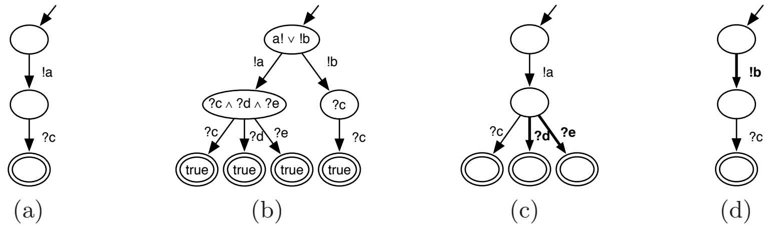


Fig. 7. Adding states to a simulating service automaton may yield sub-optimal results. The service automaton (a) does not match with the OG (b), because the formula (?c∧?d∧?e) is not satisfied. The OG, however, perfectly simulates the service automaton (a), and adding two edges achieves matching (c). However, changing the edge label of (a) from !a to !b also achieves matching, but only requires a single edit action (d).

The set *Sat* consists of all sets of labels that fulfill a state's formula. For example, consider the OG in Fig. 3(b): For state q_2 of the OG $\mathcal{O}_{\mathsf{agency}\oplus\mathsf{airline}}$, we have $Sat(\varphi(\mathsf{q}_2)) = \{\{\mathsf{?confirmation}, \mathsf{?refusal}\}\}$. Likewise, $Sat(\varphi(\mathsf{q}_3)) = \{\{\mathsf{?offer}\}, \{\mathsf{!payment}\}, \{\mathsf{?offer}, \mathsf{!payment}\}\}$.

The set *Perms* consists of all permutations of outgoing edges of two states. In a permutation, each outgoing edge of a state of the service automaton has to be present as first element of a pair (a), each outgoing edge of a state of the OG that is part of the label set β has to be present as second element of a pair (b). As the number of outgoing edges of both states may be different, ε-labels can occur in the pairs, but no pair $(\varepsilon, \varepsilon)$ is allowed (c). Finally, each edge is only allowed to occur once in a pair (d).

For $\beta = \{\mathsf{?confirmation}, \mathsf{?refusal}\}$ and state q_1 of the service automaton $\mathcal{S}_1$ in Fig. 3(a), $\{(\mathsf{?confirmation}, \mathsf{?confirmation}), (\varepsilon, \mathsf{?refusal})\}$ is one of the permutations in $Perms(\mathsf{q}_1, \mathsf{q}_2, \beta)$. Another permutation is $\{(\mathsf{?confirmation}, \mathsf{?refusal}), (\varepsilon, \mathsf{?confirmation})\}$. The permutations can be interpreted like the label pairs of the simulation edit distance: $(\mathsf{?confirmation}, \mathsf{?confirmation})$ describes a keeping of ?confirmation, $(\mathsf{?confirmation}, \mathsf{?refusal})$ describes changing ?confirmation to ?refusal, and $(\varepsilon, \mathsf{?refusal})$ the insertion of a ?refusal transition. The insertion and deletion has to be adapted to avoid incorrect or sub-optimal results (see Fig. 6–7).

Definition 3 (Subgraph insertion, subgraph deletion). *Let $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ be a service automaton and $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ an OG. Define*

$$ins(q_2) = \begin{cases} 1, & \text{if } q_2 \in F_\mathcal{O}, \\ (1-p) + \max\limits_{\beta \in Sat(\varphi(q_2))} \dfrac{p}{|\beta|} \cdot \sum\limits_{b \in \beta} L(\varepsilon, b) \cdot ins(\delta_\mathcal{O}(q_2, b)), & \text{otherwise}, \end{cases}$$

$$del(q_1) = \begin{cases} 1, & \text{if } q_1 \in F_\mathcal{S}, \\ (1-p) + \dfrac{p}{n} \cdot \sum\limits_{q_1 \xrightarrow{a} q_1'} L(a, \varepsilon) \cdot del(q_1'), & \text{otherwise}, \end{cases}$$

where n is the number of outgoing edges of q_1.

Function $ins(q_2)$ calculates the insertion cost of the optimal subgraph of the OG $\mathcal{O}$ beginning at q_2 which fulfills the formulae. Likewise, $del(q_1)$ calculates the cost of deletion of the whole subgraph of the service automaton $\mathcal{S}$ from state q_1. Both functions only depend on one of the graphs; that is, *ins* and *del* can be calculated independently from the service automaton and the OG, respectively. Definition 3 actually does not insert or delete nodes, but only calculates the similarity value of the resulting subgraphs. Only this similarity is needed to find the most similar partner service and the actual edit actions can be easily derived from the state from which nodes are inserted or deleted (cf. Table 1).

With Def. 2 describing means to respect the OG's formulae and Def. 3 coping with insertion and deletion, we can finally define the weighted quantitative matching function:

Definition 4 (Weighted quantitative matching). *Let $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ be a service automaton and $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ an OG. A* weighted quantitative matching *is a function $M : Q_\mathcal{S} \times Q_\mathcal{O} \to [0, 1]$, such that:*

$$M(q_1, q_2) = \begin{cases} 1, & \text{if } (q_1 \in F_\mathcal{S} \wedge q_2 \in F_\mathcal{O}), \\ (1-p) + W_1(q_1, q_2), & \text{otherwise}, \end{cases}$$

$$W_1(q_1, q_2) = \max_{\beta \in Sat(\varphi(q_2))} \max_{P \in Perms(q_1, q_2, \beta)} \frac{p}{|P|} \cdot \sum_{(a,b) \in P} W_2(q_1, q_2, a, b),$$

$$W_2(q_1, q_2, a, b) = \begin{cases} L(a, b) \cdot M(\delta_\mathcal{S}(q_1, a), \delta_\mathcal{O}(q_2, b)), & \text{if } (a \neq \varepsilon \wedge b \neq \varepsilon), \\ L(\varepsilon, b) \cdot ins(\delta_\mathcal{O}(q_2, b)), & \text{if } a = \varepsilon, \\ L(a, \varepsilon) \cdot del(\delta_\mathcal{S}(q_1, a)), & \text{otherwise}. \end{cases}$$

The weighted quantitative matching function is similar to the weighted quantitative simulation function (Def. 1). It recursively compares the states of the service automaton and the OG, but instead of statically taking the OG's edges into consideration, it uses the formulae and checks all satisfying subsets (W_1). Additionally, W_2 organizes the successor states determined by the labels a and b, or the insertion or deletion.

4.3 Matching-Based Edit Distance

Again, we can straight-forwardly extend the weighted quantitative matching function towards an edit distance, because the permutations give information how to modify the graph. Keeping and modification of transitions is handled as in Table 1, whereas adding and deletion of nodes can be derived from Def. 3. In fact, the weighted quantitative matching function is not a classical distance. It expresses the similarity between a service automaton and an OG (i. e., a characterization of many service automata) and is hence not symmetric. We still use the term "edit distance" to express the concept of a similarity measure from which edit actions can be derived.

Consider the example from Fig. 3. During the calculation of $M(\mathsf{q}_1, \mathsf{q}_2)$, the permutation $\{(\mathsf{?confirmation}, \mathsf{?confirmation}), (\varepsilon, \mathsf{?refusal})\}$ is considered. The first label pair denotes that the ?confirmation transition is kept unmodified. The second label pair denotes an insertion of a ?refusal transition. The value of this insertion is defined by

$$\begin{aligned} L(\varepsilon, \mathsf{?refusal}) \cdot ins(\delta_{\mathcal{O}_{\mathsf{agency} \oplus \mathsf{airline}}}(\mathsf{q}_2, \mathsf{?refusal})) &= L(\varepsilon, \mathsf{?refusal}) \cdot ins(\mathsf{q}_4) \\ &= L(\varepsilon, \mathsf{?refusal}) \end{aligned}$$

and only depends on the similarity function L.

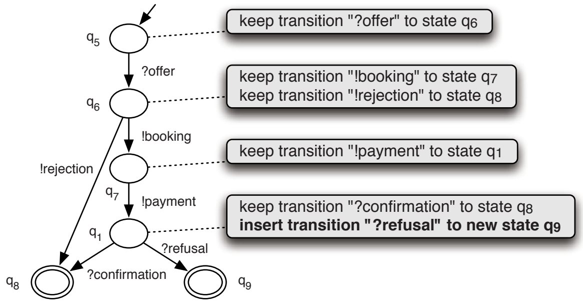

keep transition "?offer" to state q6
keep transition "!booking" to state q7
keep transition "!rejection" to state q8
keep transition "!payment" to state q1
keep transition "?confirmation" to state q8
insert transition "?refusal" to new state q9
q5
?offer
q6
!booking
!rejection
q7
!payment
q1
?refusal
?confirmation
q8
q9

Fig. 8. Matching-based edit distance applied to the customer's service

Figure 8 shows the result of the application of the matching-based edit distance to the service automaton of Fig 3(a). The states are annotated with edit actions. The service automaton was automatically generated from a BPEL process and the state in which a modification has to be made can be mapped back to the original BPEL activity. In the example, a `receive` activity has to be replaced by a `pick` activity with an additional `onMessage` branch to receive the refusal message.

5 Complexity Considerations and Experimental Results

The original simulation algorithm of [18] to calculate a weighted quantitative simulation between two service automata $\mathcal{S}_1$ and $\mathcal{S}_2$ (cf. Def. 1) needs to check

$O(|Q_{\mathcal{S}_1}| \cdot |Q_{\mathcal{S}_2}|)$ state pairs. The extension to calculate the matching between a service automaton $\mathcal{S}$ and an OG $\mathcal{O}$ (cf. Def. 4) takes the OG's formulae and the resulting label permutations into consideration. The length of the OG's formulae is limited by the maximal degree of the nodes which again is limited by the interface I. Thus, for each state pair, at most $2^{|I|}$ satisfying assignments have to be considered. The number of permutations is again limited by the maximal node degree such that at most $|I|!$ permutations have to be considered for each state pair and assignment. This results in $O(|Q_{\mathcal{S}}| \cdot |Q_{\mathcal{O}}| \cdot 2^{|I|} \cdot |I|!)$ comparisons.

Though the extension towards a formula-checking edit distance yields a high worst-case complexity, OGs of real-life services tend to have quite simple formulae, a rather small interface (compared to the number of states), and a low node degree. As a proof of concept, we implemented the edit distance in a prototype.[5] It takes an acyclic deterministic service automaton and an acyclic OG[6] as input and calculates the edit actions necessary to achieve a matching with the OG. The prototype exploits the fact that a lot of subproblems overlap, and uses dynamic programming techniques [21] to cache and reuse intermediate results which significantly accelerates the runtime. We evaluated the prototype with models of some real-life services. In most cases, the edit distance could be calculated within few seconds. The experiments were conducted on a 2.16 GHz notebook. Memory consumption is not listed as it never exceeded 10 MB. Table 2 summarizes the results.

Table 2. Experimental results

service	interface	states SA	states OG	search space	time (s)
Online Shop	16	222	153	10^{2033}	4
Supply Order	7	7	96	10^{733}	1
Customer Service	9	104	59	10^{108}	3
Internal Order	9	14	512	$> 10^{4932}$	195
Credit Preparation	5	63	32	10^{36}	2
Register Request	6	19	24	10^{25}	0
Car Rental	7	49	50	10^{144}	6
Order Process	8	27	44	10^{222}	0
Auction Service	6	13	395	10^{12}	0
Loan Approval	6	15	20	10^{17}	0
Purchase Order	10	137	168	$> 10^{4932}$	391

The first seven services are derived from BPEL processes of a German consulting company; the last four services are taken from the current BPEL specification [4]. The services were translated into service automata using the compiler BPEL2oWFN.[7] For these service automata, the OGs were calculated using the tool Fiona.[8] For some services, a partner service was already available; for the other services, we synthesized a partner service with Fiona. As we can

[5] Available at `http://service-technology.org/rachel`.

[6] Operating guidelines are deterministic by construction.

[7] Available at `http://service-technology.org/bpel2owfn`.

[8] Available at `http://service-technology.org/fiona`.

see, the services' interfaces are rather small compared to their number of states. It is worth mentioning that the complexity of the matching is independent of the fact whether the service automaton matches the OG or not. We used existing partner services in the case study to process services of realistic size.

Column "search space" of Table 2 lists the number of acyclic deterministic services characterized by the OG. All these services are correct partner services and have to be considered when finding the most similar service. The presented algorithm exploits the compact representation of the OG and allows to efficiently find the most similar service from more than 10^{2000} candidates.

For most services, the calculation only takes a few seconds. The "Internal Order" and "Purchase Order" services are exceptions. The OGs of these services have long formulae with a large number of satisfying assignments (about ten times larger than those of the other services) yielding a significantly larger search space. Notwithstanding the larger calculation time, the service fixed by the calculated edit actions is correct by design, and the calculation time is surely an improvement compared to iterative manual correction.

6 Related Work

The presented matching edit distance is related to several aspects of current research in many areas of computer science:

Automated debugging. In the field of model checking, the explanation of errors by using distance metrics (cf. [8]) has received a lot of attention. Compared to the approach presented in this paper, these works focus on the explanation and location of single errors in classical C (i.e., low-level) programs. The derived information is used to support the debugging of an erroneous program.

Service matching. Many works exists to discover a similar partner service. An approach to match BPEL processes using an algorithm based on subgraph isomorphism is presented in [22]. Other approaches such as [23,24] use ontologies and take the semantics of activities into account, but do not focus much on the behavior or message exchange. In [25], the behavior of a service is represented as a language of traces which allows for string edit distances to compare services. This approach, however, cannot be used in the setting of communicating services where the moment of branching is crucial to avoid deadlocks.

Service similarity and versioning. The change management of business processes and services is subject of many recent works. An overview of what can differ between otherwise similar services is given in [26,27]. The reported differences go beyond the behavioral level and also take authorization aspects under consideration. [28] gives an overview of frequent change patterns occurring in the evolution of a business process model. Beside the already mentioned basic operations (adding, changing and removing of edges or nodes), complex operations such as extracting sub processes are presented. With a *version preserving graph*, a technique to represent different versions of a process model is introduced in [29]. This technique was made independent of a change log in [30]. Again, versioning relies on the structure of the model rather than on its behavior.

Service mediation. Instead of changing a service to achieve deadlock freedom in a choreography, it would also be possible to use a service mediator (sometimes called adapter) to fix a choreography (e. g., [31,32]). Service mediation is rather suited to fit existing services, whereas our approach aims at supporting the design and modelling phase of a service choreography. Still, a mediator between the customer service on the one hand and the travel agency and the airline service on the other hand (cf. Fig. 1) would have to receive the airline's refusal message and create a confirmation message for the customer which is surely unintended. Furthermore, several service mediation approaches such as [33] assume total perception of the participants' internal states during runtime.

The difference between all mentioned related approaches and the setting of this paper is that these approaches either focus on low-level programs or mainly aim at finding *structural* (certainly not simulation-based) differences between *two* given services and are therefore not applicable to find the most similar service from a large set (cf. Table 2) of candidates.

7 Conclusion and Future Work

We presented an edit distance to compute the edit actions necessary to correct a faulty service to interact deadlock-freely in a choreography. The edit distance (i. e., the actions needed to fix the service) can be automatically calculated using a prototypic implementation. Together with translations from [10] and to [34] BPEL processes and the calculation of the characterization of all correct partner services (the operating guideline) [6,11], a continuous tool chain to analyze and correct BPEL-based choreographies is available. As the edit distance itself bases on service automata, it can be easily adapted to other modeling languages such as UML activity diagrams [35] or BPMN [7] using Petri net or automaton-based formalisations.

However, a lot of questions still remain open. First of all, the choice *which* service causes the deadlock and hence needs to be fixed is not always obvious and needs further investigation. For instance, the choreography of Fig. 1 could also have been fixed by adjusting the airline service. Another aspect to be considered in future research is the choice of the *cost function* used in the algorithm, because it is possible to set different values for any transition pairs. Semantic information on message contents (e. g., derived from an ontology) and relationships between messages can be incorporated to refine the correction. For example, the insertion of the receipt of a confirmation message can be penalized less than the insertion of sending an additional payment message.

Another important field of research is to further increase the performance of the implementation by an early omission of suboptimal edit actions. For instance, heuristic guidance metrics such as used in the A* algorithm [36] may greatly improve runtime performance. Finally, a translation of the matching edit distance of Def. 4 into a linear optimization problem [37] may also help to cope with cyclic and nondeterministic services.

Acknowledgements. The author wishes to thank Oleg Sokolsky for the sources of his implementation, Christian Stahl for his feedback on an earlier version of this paper, and the anonymous referees for their valuable comments. Niels Lohmann is funded by the DFG project "Operating Guidelines for Services" (WO 1466/8-1).

References

1. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Commun. ACM 44(4), 71–77 (2001)
2. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. IJCIS 13(4), 337–368 (2004)
3. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, pp. 296–303. IEEE, Los Alamitos (2007)
4. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0, April 11, 2007. OASIS Standard, OASIS (2007)
5. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. LNCS, vol. 4937, pp. 46–60. Springer, Heidelberg (2008)
6. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
7. OMG: Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, Object Management Group (2006), http://www.bpmn.org
8. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. STTT 8(3), 229–247 (2006)
9. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
10. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
11. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
12. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
13. Schmidt, K.: Controllability of open workflow nets. In: EMISA 2005. LNI, vol. P-75, pp. 236–249, GI (2005)
14. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. on SMC 13(3), 353–362 (1983)
15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Dokl. 10(8), 707–710 (1966)
16. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21(1), 168–173 (1974)
17. Tsai, W., Fu, K.: Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans. on SMC 9(12), 757–768 (1979)
18. Sokolsky, O., Kannan, S., Lee, I.: Simulation-based graph similarity. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 426–440. Springer, Heidelberg (2006)
19. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S.M., Zave, P.: Matching and merging of statecharts specifications. In: ICSE, pp. 54–64. IEEE Computer Society, Los Alamitos (2007)

20. Namjoshi, K.S.: A simple characterization of stuttering bisimulation. In: Ramesh, S., Sivakumar, G. (eds.) FST TCS 1997. LNCS, vol. 1346, pp. 284–296. Springer, Heidelberg (1997)
21. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
22. Corrales, J.C., Grigori, D., Bouzeghoub, M.: BPEL processes matchmaking for service discovery. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 237–254. Springer, Heidelberg (2006)
23. Wu, J., Wu, Z.: Similarity-based Web service matchmaking. In: IEEE SCC, pp. 287–294. IEEE Computer Society, Los Alamitos (2005)
24. Bianchini, D., Antonellis, V.D., Melchiori, M.: Evaluating similarity and difference in service matchmaking. In: EMOI-INTEROP. CEUR Workshop Proceedings, CEUR-WS.org, vol. 200 (2006)
25. Günay, A., Yolum, P.: Structural and semantic similarity metrics for Web service matchmaking. In: Psaila, G., Wagner, R. (eds.) EC-Web 2007. LNCS, vol. 4655, pp. 129–138. Springer, Heidelberg (2007)
26. Dijkman, R.M.: A classification of differences between similar BusinessProcesses. In: EDOC, pp. 37–50. IEEE Computer Society, Los Alamitos (2007)
27. Dijkman, R.: Diagnosing differences between business process models. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, pp. 132–147. Springer, Heidelberg (2008)
28. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
29. Zhao, X., Liu, C.: Version management in the business process change context. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 198–213. Springer, Heidelberg (2007)
30. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 244–260. Springer, Heidelberg (2008)
31. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
32. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
33. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007, pp. 993–1002. ACM, New York (2007)
34. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. LNI, vol. P-127, pp. 57–72, GI (2008)
35. OMG: Unified Modeling Language (UML), Version 2.1.2. Technical report, Object Management Group (2007), http://www.uml.org
36. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans.Syst. Sci. and Cybernetics SSC-4(2), 100–107 (1968)
37. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & sons, Chichester (1998)

Predicting Coupling of Object-Centric Business Process Implementations

Ksenia Wahler and Jochen M. Küster

IBM Zurich Research Laboratory, Säumerstr. 4
8803 Rüschlikon, Switzerland
{ryn,jku}@zurich.ibm.com

Abstract. Object-centric approaches for business process implementation distribute process logic among several interacting components, each representing a life cycle of an object. One of the challenges is to manage the component coupling, because highly-coupled components are difficult to distribute, maintain and adapt. Existing techniques that derive a component for each object that changes state in a given process do not consider component interdependencies and run the risk of producing components that are highly coupled. To make coupling explicit and manageable during component identification, we propose an approach for computing the expected coupling of an object-centric implementation for a given process model prior to actually deriving this implementation.

Keywords: Coupling, object life cycle, object-centric and data-driven processes, state machines.

1 Introduction

Most existing languages for business process modeling (e.g. BPMN [3]) and implementation (e.g. BPEL [1]) are *activity-centric*, because they represent processes as a set of activities connected by control-flow elements to indicate the order of activity execution. In recent years however, a line of alternative *object-centric* approaches for modeling and implementing business processes has been proposed, which include artifact-centric modeling [6,15], adaptive business objects [14], data-driven modeling [11] and proclets [20]. Activities in the process are distributed among several *components*, each representing an *object life cycle* that defines possible states of a particular object and transitions between these states. Interaction between such *object life cycle components* ensures that the overall process logic is correctly implemented. Object-centric implementations can be used for *distributed* process execution and can lead to a more *maintainable* and *adaptable* implementation than activity-centric approaches, as the behavior of one object can be partially changed without influencing the rest of the process [10]. However, the more dependencies and interactions there are between the object life cycle components, the costlier becomes their distribution and the more complicated it is to change their behavior.

One of the challenges in object-centric process implementation is therefore the management of component interdependencies, commonly referred to as *coupling* in software engineering [7]. Several object-centric approaches advocate deriving object life cycles from activity-centric process models that specify the process logic to be implemented. The existing derivation methods [6,18,19] do not explicitly address object life cycle interdependencies and hence run the risk of producing components that are highly coupled. Component refactoring, e.g. moving some behavior from one component to another or merging components, is one approach to reduce coupling. However, as a result the process model can get out of sync with its implementation, which challenges the propagation of any subsequent process model changes to the implementation. This problem can be alleviated by making the developer aware of the expected coupling before component derivation, so that the process model can be adapted until a desired level of coupling is achieved. Realization of this approach requires the computation of the expected component coupling based on a given process model.

The problem addressed in this paper is therefore the prediction of the expected coupling of an object-centric implementation based on a given process model. We first review the mapping of the most common *workflow patterns* [21] to object life cycle components in order to identify how properties of a process model influence the coupling of the derived components. We then show that given a process model, it is possible to compute the object life cycle component pairs that require interaction by analyzing the control flow between activities that change the state of objects. Finally, we use this information to compute the expected coupling of the object life cycle components.

We implement object life cycle components using Business State Machines (BSMs) [5]. BSMs are introduced in Sect. 2, along with an illustrative example and the coupling metric used. In Sect. 3, we demonstrate how workflow patterns can be implemented using BSMs and study how solutions for different patterns contribute to the overall coupling. These observations are formalized in Sect. 4, where we define how to compute the expected coupling based on a given process model. In Sect. 5, we discuss the generalization of our approach. Related work and conclusions are presented in Sect. 6 and 7, respectively.

2 Example and Background

As an illustrative example, we use a process designed for the organization of alumni events at the IBM Zurich Research Laboratory. An abridged BPMN [3] model for this process is shown in Fig. 1. After the approval of the budget, the date for the event is fixed and then two things happen in parallel: the program, invitations and web site are prepared; and catering is organized. After all these have completed, the alumni day is hosted. The process model contains three sub-processes: Fix Date, Prepare And Send Invitations and Develop Web Site.

All activities of a business process generally transform some objects by changing their state to contribute to the final goal of the process. For each atomic activity in the alumni day process, we indicate the state-changing objects[1]. For

[1] We use the notation given on p.94 in [3] for object outputs of an activity and a shorthand notation for objects that are both inputs and outputs of an activity.

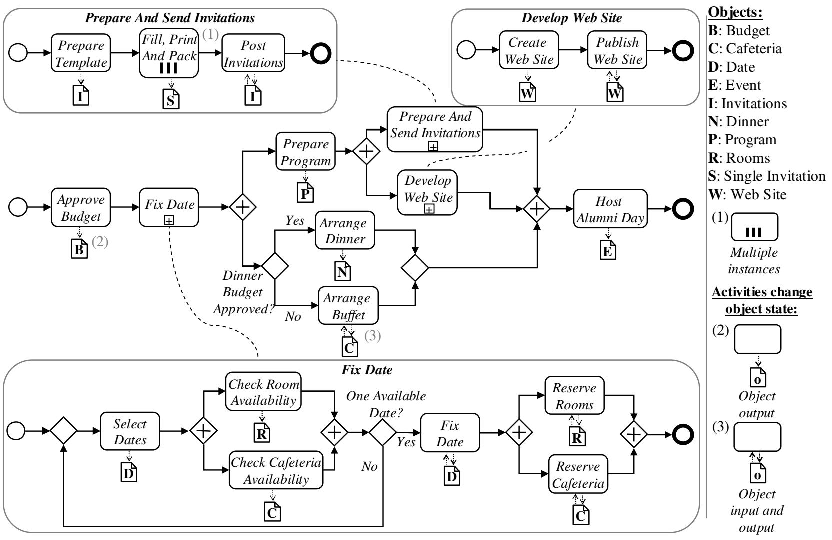


Fig. 1. Process Model for Alumni Day Organization

example, the Create Web Site activity produces a Web Site object in state Drafted and Publish Web Site changes the state of the Web Site object from Drafted to Published (states are omitted in this diagram). In the Prepare And Send Invitations subprocess, Prepare Template creates an Invitations object in state TemplatePrepared, and then multiple instances of the Fill, Print And Pack activity are performed in parallel, each creating a Single Invitation object. Once all instances of Fill, Print And Pack have completed, Post Invitations updates the state of Invitations to Posted.

In an object-centric implementation of the alumni day process, the process logic is split into ten object life cycle components, assuming an approach in which one component is derived for each state-changing object. We implement each object life cycle component as a Business State Machine (BSM) [5]. A simple example of a BSM is shown in Fig. 2.

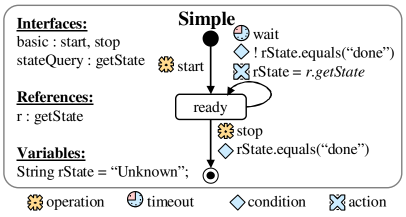


Fig. 2. Example BSM

A BSM is a finite state automaton, tailored for execution in a service-oriented environment. Each BSM can have several of the following: *interfaces*, *references* and *variables*. The Simple BSM in Fig. 2 has two interfaces: basic comprising *operations* start and stop, and stateQuery with the getState operation. These are the three operations that can be invoked on this BSM. Simple also has one reference r, referencing an interface of another BSM, with one operation getState. Operations in addition have parameters, which we omit here. Simple has one variable rState, initialized to the literal "Unknown".

State transitions in BSMs follow the *event-condition-action* paradigm. A transition can be triggered either by an expiration of a *timeout* or by an invocation of an *operation* defined in one of the BSM's interfaces. Once a transition has been triggered, its associated *condition*, if any, is evaluated. If the condition evaluates to true or there is no condition, the *action* associated with the transition, if any, is performed and the target state of the transition is entered. An action either invokes an operation on one of the BSM's references or performs some other processing specified in a custom language, such as Java. For example, once the Simple BSM is in state ready, a self-transition is triggered repeatedly after expiration of the timeout wait. Each time the transition is triggered and rState is not equal to "done", the getState operation is invoked on r (invocation is indicated using italics in the diagrams). The operation getState is implicitly handled by every BSM and returns the BSM's current state. Invocation of the stop operation on Simple results in a transition to the final state only if rState is equal to "done".

At runtime, each BSM instance is associated with a *correlation ID*. The runtime engine creates a new BSM instance if it receives a call to an operation associated with an initial transition of some BSM and this operation call specifies a correlation ID that does not correspond to an existing BSM instance.

For the implementation of the alumni day process, we distribute the process activities among ten BSMs (Budget, Cafeteria, Date, etc). We make a simplifying assumption that one activity changes the state of exactly one object, as in the example process model. Each activity is placed into the BSM that represents the state-changing object for this activity. In Sect. 6, we explain how our approach can be extended to handle activities that change the state of several objects.

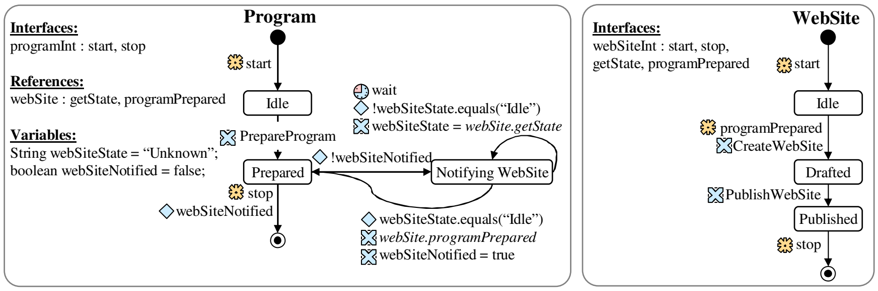


Fig. 3. Example BSMs

Partial implementations of the Program and WebSite BSMs are shown in Fig. 3. Activities that change the state of these two objects are mapped to actions associated with state transitions in the BSMs, e.g. the PrepareProgram and CreateWebSite actions. These actions can be implemented to invoke service operations, human tasks, etc. In the process model, the Prepare Program activity must complete before the Create Web Site activity can execute. *Synchronization* of the Program and WebSite BSMs is implemented to preserve this dependency: After the PrepareProgram action has been performed in the Program BSM and the Prepared state has been reached, the Program BSM transits to state Notifying WebSite. In this state, the Program BSM queries the state of the WebSite BSM repeatedly. Once the WebSite BSM has reached the Idle state, the Program BSM notifies the WebSite BSM that it has reached the state Prepared by invoking the programPrepared operation. After this, the Program BSM transits back to state Prepared, and the WebSite BSM can perform the CreateWebSite action. In a complete BSM implementation of the alumni day process, many such synchronizations need to be implemented, e.g. Program also needs to synchronize with the Invitations BSM.

Aside from additional states and transitions within BSMs, synchronization also leads to interface bindings between the BSMs. We use the *Service Component Architecture (SCA)* [2], which is a service-oriented component framework, to represent these bindings. Each BSM is an implementation of an *SCA component* (used interchangeably with component from now on). An *assembly model* in SCA is a representation of directed communication channels, called *wires*, between components. The assembly model for the BSMs from Fig. 3 is shown in Fig. 4. Synchronization of the Web Site and Program BSMs requires that the components are connected by a wire in the assembly model.

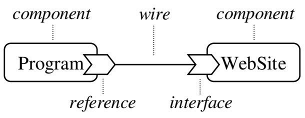


Fig. 4. Assembly Model

Definition 1 (Assembly model). *An* assembly model *is a tuple $M = (C, \phi)$, where C is the set of components in M and $\phi \subseteq C \times C$ is the wire relation between components.*

In the context of SCA, we use the term coupling to refer to interdependencies of components in an assembly model. We quantify the coupling of an assembly model by defining the *interface coupling* metric, adapted from existing work on quality metrics in the business process domain [17].

Definition 2 (Interface coupling). *Given an assembly model $M = (C, \phi)$, its interface coupling is defined as follows:*

$$p(M) = \begin{cases} 0 & \textit{if } |C| = 0 \textit{ or } 1 \\ \frac{|\phi|}{|C| \times (|C|-1)} & \textit{otherwise} \end{cases} \tag{1}$$

Interface coupling represents the ratio between the actual number of wires and the maximum possible number of wires between the components in the assembly model. A coupling value of 0 means that there is no interaction at all between the components. This implies that the distribution of these components does not incur any communication costs, and the implementation of each component can be maintained and its behavior adapted at run time with no side-effects on the other components. On the contrary, a coupling value of 1 means that every component interacts with every other component. The distribution of such components will incur high communication costs, and maintenance or adaptation of one component affects the behavior of all other components. The interface coupling of the assembly model shown in Fig. 4 is $\frac{1}{2\times 1} = 0.5$. More refined coupling metrics could also be used here, e.g. to take into account the number of operations in the component interfaces connected to wires or the number of operation calls inside the BSMs.

In the following section, we explore how the implementation of different workflow patterns using BSMs introduces wires between the BSM components and thus contributes to the coupling of the resulting assembly model.

3 Implementing Workflow Patterns Using BSMs

Workflow patterns [21] are a well-established benchmark for exploring how common process behaviors can be represented in different business process modeling and implementation languages. In this section, we show how the basic control-flow patterns, WP1-WP5, can be modeled using BSMs. In addition, we provide a solution for WP14, as it can be used to represent the processing of object collections, commonly occurring in process models. We provide BSM solutions on an exemplary basis, similar to existing evaluations of other languages (e.g. [22]). We discuss the requirements that each pattern has with respect to the synchronization of BSMs and its contribution to the coupling of the overall implementation.

WP1 Sequence. Several activities are executed one after another in this pattern, as illustrated with two examples[2] in Fig. 5. In E1, ActivityA and ActivityB change the state of the same object o1, whereas in E2 ActivityA and ActivityB change the state of different objects, o1 and o2.

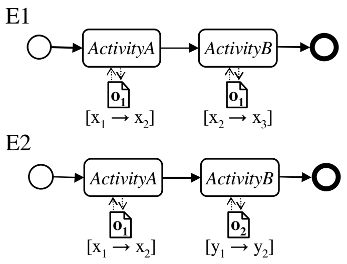


Fig. 5. WP1 Examples

The solution for E1 is straightforward, see Fig. 6(a) (interfaces and references are omitted). It comprises one component, shown in the assembly model at the bottom of the diagram. A solution for E2 is shown in Fig. 6(b), where BSMs o1 and o2 represent the life cycles of objects o_1 and o_2, respectively.

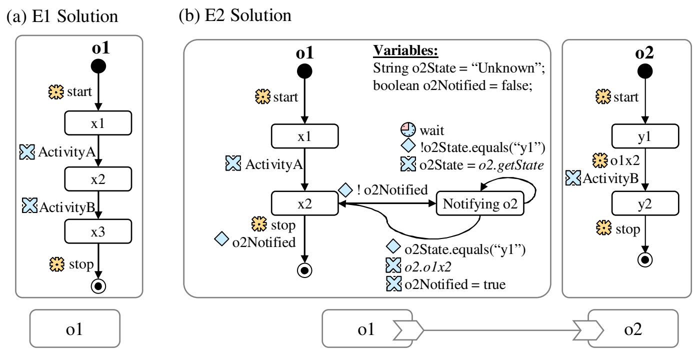


Fig. 6. WP1 Solutions

[2] We use a shorthand of the form $[state_{src} \rightarrow state_{tgt}]$, based on the notation given on p.94 in [3], to show how an activity changes the state of an object.

Once ActivityA has been performed by o1, o1 notifies o2 that it has reached state x2 by first ensuring that o2 is in state y1 and then invoking the o1x2 operation on o2. Once o1x2 has been invoked on o2, ActivityB is performed by o2. The resulting assembly model has an interface coupling of $\frac{1}{2\times 1} = 0.5$.

WP1 Synchronization Requirements: A generic instance of WP1 comprises activities $a_1, ..., a_n$, which change the states of objects $o_1, ..., o_n$, respectively. A pair of activities a_i, a_{i+1}, with $1 \leq i < n$, requires a synchronization of BSM o_i and BSM o_{i+1} if and only if $o_i \neq o_{i+1}$. We introduce the *control handover* synchronization category for such synchronizations, since they represent the handover of control between BSMs. Each such control handover requires a wire from BSM o_i to BSM o_{i+1} to be present in the assembly model. The introduction of these wires contributes to the overall coupling of the resulting assembly model.

WP2 Parallel Split & WP3 Synchronization. In WP2, several activities are executed simultaneously or in any possible order, and in WP3, several parallel threads are joined together into a single control thread. An example containing an instance of both of these workflow patterns is shown in Fig. 7. In E3, each activity changes the state of a different object. Note that we do not only consider block-structured process models, but examine these two patterns together for the sake of conciseness.

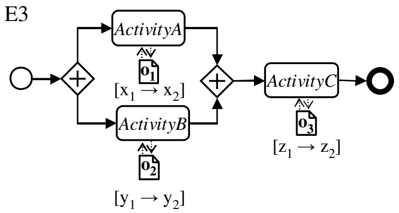


Fig. 7. WP2 & WP3 Example

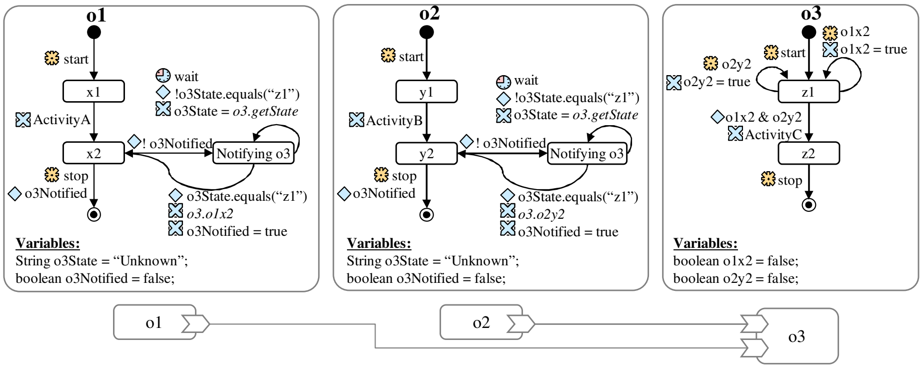


Fig. 8. WP2 & WP3 E3 Solution

A solution for E3 is shown in Fig. 8. As by default all BSMs are executed concurrently, no explicit parallel split is required. Synchronization of the threads is performed using notifications, similar as in the E2 solution in Fig. 6(b). BSM o3 waits to receive notifications from both o1 and o2 (operation calls o1x2 and o2y2) before performing ActivityC. The interface coupling of the assembly model for this solution is $\frac{2}{3\times 2} \approx 0.33$.

WP2 & WP3 Synchronization Requirements: As instances of WP2 do not require any interaction between BSMs, they do not contribute wires to the assembly model and have no effect on the coupling. A generic instance of WP3 comprises activities $a_1, ..., a_n$ that all need to complete before activity a_{n+1} can begin execution. Assuming that $a_1, ..., a_n, a_{n+1}$ change the states of objects $o_1, ..., o_n, o_{n+1}$, respectively, a pair of activities a_i, a_{n+1}, with $1 \leq i \leq n$, requires a synchronization of BSMs if and only if $o_i \neq o_{n+1}$. These synchronizations also fall into the control handover category, introduced for WP1.

WP4 Exclusive Choice & WP5 Simple Merge. In WP4, one out of several activities is executed based on the outcome of a decision, and in WP5, several alternative threads are joined into one control thread without synchronization. An example containing instances of these patterns is shown in Fig. 9.

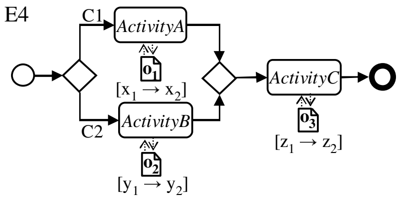


Fig. 9. WP4 & WP5 Example

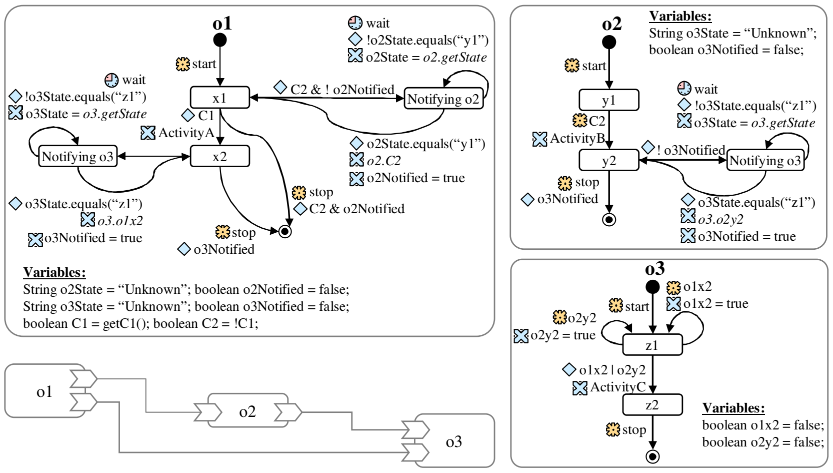


Fig. 10. WP4 & WP5 E4 Solution

In a solution for this pattern, the decision needs to be placed into one of the BSMs, as shown in Fig. 10 where it is placed in BSM o1 (two transitions going out of state x1 with conditions C1 and C2). Once the decision has been evaluated in o1[3], either ActivityA is performed (C1 is true) or o2 is notified and ActivityB is performed in o2 (C2 is true). The merging of alternative control threads is implemented similarly to the synchronization solution in Fig. 8, except that BSM o3 performs ActivityC as soon as it receives one of the operation calls,

[3] For simplicity, we initialize C1 and C2 in the variable definitions here. In a real implementation, they would be evaluated in state x1.

o1x2 or o2y2. The interface coupling of the assembly model is $\frac{3}{3\times2} = 0.5$. These three components have a higher coupling value than those in Fig. 8, because of an additional wire between o1 and o2 required for communicating the decision outcome.

WP4 & WP5 Synchronization Requirements: A generic instance of WP4 comprises a decision d and activities $a_1, ..., a_n$, which change the states of objects $o_1, ..., o_n$, where one of these activities is executed depending on the evaluation of the conditions of d. We assume that the evaluation of d can be assigned to an object o_i, where $1 \leq i \leq n$. BSM o_i requires synchronization with each BSM o_j, where $1 \leq j \leq n$ and $o_i \neq o_j$. Since such synchronizations do not represent control handovers, we introduce a new synchronization category called *decision notification* for such synchronizations. Instances of WP5 require control handover synchronizations, similar to instances of WP1 and WP3.

WP14 Multiple Instances with a priori Run-Time Knowledge. In WP14, multiple instances of the same activity are created, all of which need to complete before a subsequent activity can be executed. The number of instances is not known at design time, but is determined at run time before activity instances are created. This pattern can be used to represent the processing of object collections, as shown in the example in Fig. 11. In E5, a collection of o2 objects is processed by multiple instances of ActivityB. In this example, each activity instance creates a new object o2 in state y1. Once all instances of ActivityB have completed, ActivityC is executed. We show this particular example here, because it corresponds to the Prepare And Send Invitations sub-process in Fig. 1, where Invitations and Single Invitation objects take the role of o1 and o2, respectively.

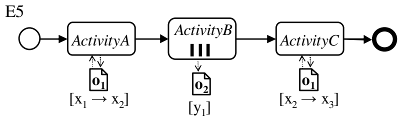


Fig. 11. WP14 Example

A solution for E5 is shown in Fig. 12. After performing ActivityA, o1 transits to state x2 and then to Creating o2s, where it creates n instances of the o2 BSM by repeatedly invoking the start operation with a fresh correlation ID. Each o2

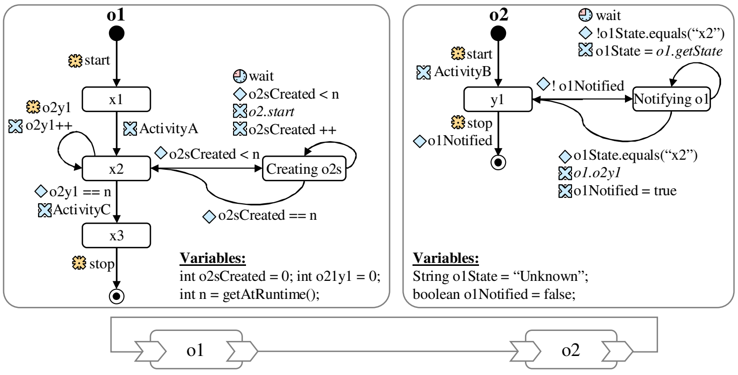


Fig. 12. WP14 E5 Solution

instance performs ActivityB and then notifies o1 that it has reached state y1. Once o1 has received notifications from all o2 instances, it performs ActivityC and transits to state x3. The interface coupling for the assembly model is $\frac{2}{2\times 1} = 1$.

WP14 Synchronization Requirements: A generic instance of WP14 comprises activities a_1, a_2, a_3, which change the states of objects o_1, o_2, o_3, where activity a_2 is to be instantiated multiple times. Provided that a_1 and a_3 are not themselves multiple instance activities, the following control handovers are always required: from BSM o_1 to instances of BSM o_2, and from instances of BSM o_2 to BSM o_3. Although the number of synchronizations at run time will vary, the contribution to the coupling is constant, as two wires, (o_1, o_2) and (o_2, o_3), are introduced into the assembly model to enable the synchronizations (this also holds if $o_1 = o_3$). The case where $o_1 = o_2$ and $o_2 = o_3$ is an exception, as in this case only one wire (o_2, o_2) would be introduced into the assembly model. For simplification, we do not consider this case in the remainder of the paper.

In this section, we have demonstrated how workflow patterns can be implemented using BSMs and discussed the requirements of each pattern for the synchronization of BSMs. In the next section, we show how the number of control handovers and decision notifications can be computed for a given process model, and then used to compute the expected coupling of a BSM implementation.

4 Predicting Coupling of BSM Implementations

We assume that the process model provided as a specification for a BSM implementation comprises instances of WP1-WP5 and WP14 only and has each activity associated with one state-changing object, as in the alumni day process model in Fig. 1. As a sub-process hierarchy in a given process model can be flattened for processing, we use the following definition for a process model.

Definition 3 (Process model). *A* process model *is a tuple $P = (G, O, \sigma)$:*

- *$G = (N, E)$ is a directed graph, in which each node $n \in N$ is either a start node, stop node, activity, fork, join, decision, or merge. As a shorthand, we use N_A and N_D to denote activities and decisions in N, respectively.*
- *O is the set of objects whose states are changed by activities $a \in N_A$.*
- *$\sigma \subseteq N_A \times O$ is the state-changing relation between activities and objects. We use o_a to denote the object whose state is changed by activity $a \in N_A$, i.e. $(a, o_a) \in \sigma$.*

Given a process model P, the number of components in the assembly model of its BSM implementation is equal to the number of objects whose states are changed in P, assuming a simple mapping. In Sect. 3, we showed that the number of wires between the components depends on the control handover and decision notification synchronizations between the BSMs. As all the synchronizations required by different patterns fall into these two categories, we directly compute all object pairs that require such synchronizations, instead of first identifying workflow pattern instances in a given process model.

A control handover is required whenever an activity in the process model that changes the state of one object has a direct successor activity[4] that changes

[4] Only edges and gateways connect an activity and its direct successor activity.

the state of another object. A decision notification is required between the object assigned to evaluate a decision and all the objects whose state is changed by the direct successor activities of that decision. To compute the objects that require control handovers and decision notifications, we propagate the information about state-changing objects *downstream* from each activity to its direct successor activities and *upstream* from direct successor activities of decisions.

Definition 4 (Downstream and upstream control objects). *Given a process model $P = (G, O, \sigma)$ where $G = (N, E)$, each edge $e \in E$ is associated with* downstream *and* upstream control objects*, $dco(e), uco(e) \subseteq O$ respectively, defined as follows:*

$$dco(e) = \begin{cases} \emptyset & \text{if } e \text{ is the outgoing edge of the start node} \\ \{o_a\} & \text{if } e \text{ is the outgoing edge of activity } a \in N_A \\ \bigcup_{i=1}^{m} dco(e_i) & \text{otherwise, where } e_1, ..., e_m \text{ are the incoming edges} \\ & \text{of node } n, \text{ which has } e \text{ as its outgoing edge} \end{cases} \tag{2}$$

$$uco(e) = \begin{cases} \emptyset & \text{if } e \text{ is the incoming edge of the stop node} \\ \{o_a\} & \text{if } e \text{ is the incoming edge of activity } a \in N_A \\ \bigcup_{i=1}^{m} uco(e_i) & \text{otherwise, where } e_1, ..., e_m \text{ are the outgoing edges} \\ & \text{of node } n, \text{ which has } e \text{ as its incoming edge} \end{cases} \tag{3}$$

Downstream and upstream control objects can be computed for a given process model using data flow analysis techniques [9]. For example, to compute the downstream control objects, $dco(e)$ is initialized to an empty set for each edge e and then the nodes in the process model are traversed, evaluating the dco equations (Equation 2) for each outgoing edge of the traversed node. Reverse postorder traversal ensures that in the absence of cycles each node is visited once. In the presence of cycles, the nodes are traversed repeatedly until a fixpoint is reached, i.e. an iteration when no dco values are updated. Fig. 13 shows the alumni day process model with the downstream and upstream control objects indicated above and below each edge, respectively[5]. The set of object pairs that need to perform control handover is then defined as follows.

Definition 5 (Control handover object pairs). *Given a process model $P = (G, O, \sigma)$ where $G = (N, E)$ and each of the edges $e_1, ..., e_n$ is an incoming edge of some activity $a \in N_A$, the set of directed* object pairs *that require BSMs to perform* control handover *is defined as follows:*

$$O^{ch}(P) = \bigcup_{i=1}^{n} (dco(e_i) \times uco(e_i)) \setminus \{(o, o) \mid o \in O\} \tag{4}$$

For example, the incoming edge of the AD activity gives rise to two control handover object pairs: (R,N) and (C,N); and the incoming edge of the AB2 activity gives rise to only one control handover object pair: (R,C).

Next we define object pairs that require decision notification between BSMs. Given a decision d, we denote its outgoing edges by E_d^{out} and assume that the evaluation of d can be assigned to the object $co(d)$, which is one of the upstream control objects of some edge in E_d^{out}.

[5] Activity names are abbreviated in Fig. 13.

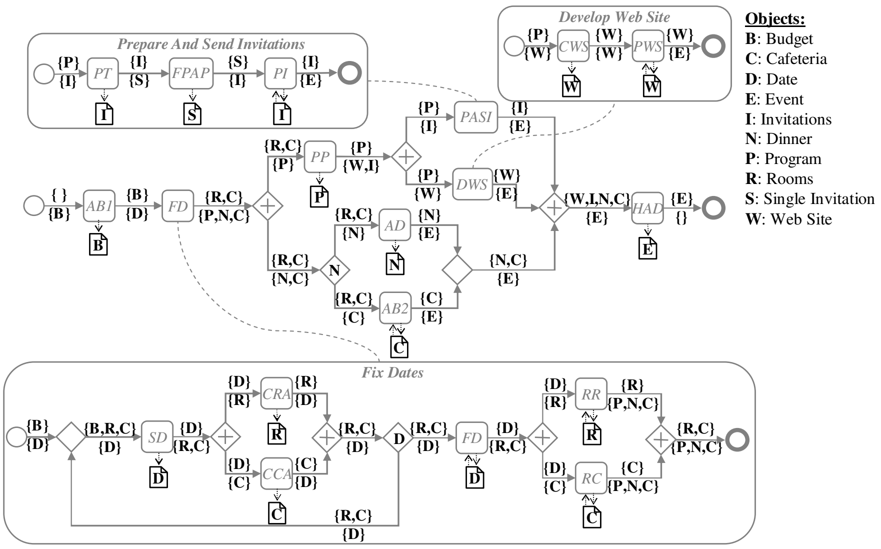


Fig. 13. Downstream and Upstream Control Objects in a Process Model

Definition 6 (Decision notification object pairs). *Given a process model $P = (G, O, \sigma)$ where $G = (N, E)$, the set of directed* object pairs *that require* decision notification *between BSMs is defined as follows:*

$$O^{dn}(P) = \bigcup_{d \in N_D} \left(co(d) \times \bigcup_{e \in E_d^{out}} uco(e) \right) \setminus \{(o, o) \mid o \in O\} \tag{5}$$

The decision in the parent alumni day process model is assigned to object N and gives rise to one decision notification object pair: (N,C). The decision in the Fix Date sub-process is assigned to object D. It does not introduce any decision notification object pairs, as the sets of upstream control objects for both edges going out of the decision are the same: {D}.

The predicted assembly model for a BSM implementation of a given process model can now be constructed by introducing a component for each object and a wire for each of the control handover and decision notification object pairs.

Definition 7 (Predicted assembly model for a BSM implementation). *Given a process model $P = (G, O, \sigma)$, the* predicted assembly model for a BSM implementation *of P is defined as follows:*

$$M_P = (C_P, \phi_P) \tag{6}$$

- *where $C_P = \{c_{o_1}, ..., c_{o_n}\}$ is the set of components, with one component c_{o_i} for each object $o_i \in O$ where $1 \leq i \leq n$,*
- *and $\phi_P = \{(c_{o_1}, c_{o_2}) \in C_P \times C_P \mid (o_1, o_2) \in O^{ch}(P) \cup O^{dn}(P)\}$ is the wire relation between components.*

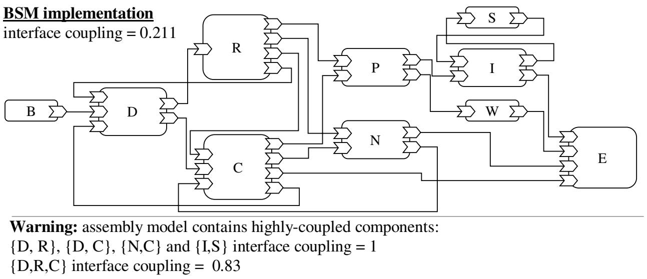

BSM implementation
interface coupling = 0.211
R
S
P
I
B
D
W
N
E
C
Warning: assembly model contains highly-coupled components:
{D, R}, {D, C}, {N,C} and {I,S} interface coupling = 1
{D,R,C} interface coupling = 0.83

Fig. 14. Predicted Assembly Model for the Alumni Day Process

The assembly model for the alumni day process model is shown in Fig. 14. It can be seen that each distinct control handover and decision notification object pair, such as (R,N) or (C,N), introduces a wire in the predicted assembly model.

The interface coupling is computed for the entire assembly model and for all component subsets according to Definition 2. A configurable upper bound is used to assess the predicted coupling values. This upper bound can be evolved as a best practice by developers, i.e. first initialized to some value and then refined in further iterations or projects based on the experience gained in deploying and maintaining object-centric implementations. Empirical evaluations can also help in determining a generic guideline for this upper bound. In Fig. 14, the overall interface coupling is $\frac{19}{10\times 9} \approx 0.211$, which would not give a reason for concern, assuming for example an upper bound of 0.8. However, component sets {D,R}, {D,C}, {N,C}, {I,S} and {D,R,C} have a coupling value higher than 0.8 and would thus be brought to the attention of the developer, as shown in Fig. 14.

Once the expected coupling is predicted using the proposed approach, the developer should decide how to deal with each set of highly-coupled components. High coupling may be tolerated for components that have a stable design and do not require distributed deployment. Otherwise, the process model should be revised in such a way that the expected coupling between components is reduced. Possible revisions include identification of objects that can be represented by a merged life cycle and refactoring control flow in the process model. Object life cycle merger should be applicable only for those objects that have a strong semantic relationship. For example, the Dinner (N) and Cafeteria (C) life cycles, which give rise to the highly-coupled component set {N,C}, can be merged to produce a Catering life cycle. In order to alleviate component coupling by process model refactoring, the number of control handovers and decision notifications should be reduced. In the alumni day process, the decision Dinner Budget Approved? and the activities connected to its outgoing edges could take place directly after the Reserve Cafeteria activity, without waiting for the Reserve Event Rooms activity to complete. This refactoring would reduce the coupling of the {R,C} and {R,N} component sets. After each life cycle merger and process model refactoring, the coupling computations need to be repeated and shown to the developer.

5 Discussion

In this paper, we have shown how the coupling of an object-centric implementation using BSMs can be predicted using a given process model. We assumed that each activity in the process model changes the state of one object. An activity that changes the state of several objects would be placed into several BSMs, which would need to synchronize, thus contributing to component coupling. Our current approach can be extended to handle such activities by adding a new synchronization category, *activity synchronization*, and providing a definition for computing the object pairs requiring such a synchronization (similar to Definitions 5 and 6). The approach can be further extended to handle workflow patterns other than WP1-WP5 and WP14 by investigating BSM solutions for these patterns, identifying pattern requirements for synchronization of BSMs, and extending Definitions 5, 6 and 7.

Although our approach was demonstrated using SCA and BSMs, it can be generalized to other component frameworks (not necessarily based on services) and other object-centric approaches. For example, adaptive business objects (ABO) [14] are based on communicating automata, and our approach is applicable once every ABO has been encapsulated in a component and communication channels between the components have been made explicit. In data-driven modeling [11], object life cycles are synchronized by so-called external state transitions. To compute the coupling, each life cycle can be seen as a component, and communication channels need to be introduced between components whose life cycles are connected by external state transitions. Proclets [20] use WF-nets to represent object life cycles and make use of explicit communication channels. Although more advanced communication options, such as multicast and broadcast, are supported in proclets, our approach is still applicable.

6 Related Work

In component-based development, the coupling has been used for component identification [8] and refactoring [4]. For example, a statistics technique called clustering analysis to form components that have high cohesion and low coupling is used in [8]. Such approaches are complementary to what we propose in this paper, as they can help to identify how the highly-coupled components in the predicted assembly model of a BSM implementation can be alleviated.

Many different categories or types of coupling have been identified in software engineering [7]. Given source code or a component model, it is usually straightforward to calculate the different coupling values, as the metrics are defined directly in terms of source code or component model elements. In our approach, we determine how the control flow in a given process model influences the coupling of the resulting BSM implementation before actually deriving the BSMs. So far we have focused on the so-called interface coupling of SCA components; however other types of coupling, such as data coupling, could also be considered.

A tight correlation between the semantic relationships of objects and synchronization of their life cycles has been identified in manufacturing processes [11,16].

In manufacturing, objects are naturally coupled by the "part-of" relationship. Our approach is valuable also in this context, as it can identify whether the implementation components have dependencies other than those resulting from the semantic relationships between objects.

In workflow management, several approaches have been proposed for decentralizing workflows with the goal of optimizing their execution [12,13]. For example, the approach in [12] involves minimizing the loads and number of synchronization messages exchanged between the distributed workflow components. Although in our approach we also strive to reduce the number of dependencies between components, execution optimization is not our primary focus. The object life cycle components dealt with in this paper need to be refined and maintained by developers, whereas workflow decentralization happens once a workflow is deployed and its results are not exposed to the developers.

7 Conclusions and Future Work

We have presented an approach for predicting the coupling of an object-centric implementation for a given process model. Our example showed that deriving one component for each state-changing object can produce highly-coupled components, which are difficult to distribute, maintain and adapt. The predicted coupling information allows the developer to take preventive actions to arrive at a better decomposition of the final implementation. Although our approach has been demonstrated using BSMs, it is possible to generalize it to other languages suitable for object-centric process implementation.

We are currently extending the approach with the prediction of cohesion and complexity metrics. We expect that the incorporation of these two metrics with coupling will not only offer deeper insights into object-centric implementations, but will also facilitate a comparison of activity-centric and object-centric implementation approaches.

References

1. Business Process Execution Language for Web Services, Version 1.1. Joint specification by BEA, IBM, Microsoft, SAP and Siebel Systems (2003)
2. SCA Service Component Architecture, Assembly Model Specification, SCA Version 1.00, Open SOA Collaboration Specification (March 2007)
3. Business Process Modeling Notation Specification, V1.1., formal/2008-01-17. OMG Document (January 2008)
4. Abreu, F.B., Pereira, G., Sousa, P.: A Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems. In: Proc. of the Conf. on Software Maintenance and Reengineering, pp. 13–22. IEEE Computer Society, Los Alamitos (2000)
5. Beers, G., Carey, J.: WebSphere Process Server Business State Machines concepts and capabilities, Part 1: Exploring basic concepts. IBM developerWorks (October 2006)

6. Bhattacharya, K., Guttman, R., Lyman, K., et al.: A Model-Driven Approach to Industrializing Discovery Processes in Pharmaceutical Research. IBM Systems Journal 44(1), 145–162 (2005)
7. Briand, L.C., Daly, J.W., Wüst, J.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering 25(1), 91–121 (1999)
8. Lee, J.K., Seung, S.J., Kim, S.D., Hyun, W., Han, D.H.: Component Identification Method with Coupling and Cohesion. In: Proc. of the 8th Asia-Pacific Conf. on Software Engineering, pp. 79–86. IEEE Computer Society, Los Alamitos (2001)
9. Muchnick, S.: Advanced Compiler Design and Implementation. Morgan Kaufmann, San Francisco (1997)
10. Müller, D., Reichert, M., Herbst, J.: Flexibility of Data-Driven Process Structures. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 181–192. Springer, Heidelberg (2006)
11. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
12. Muth, P., Wodtke, D., Weißenfels, J., Kotz Dittrich, A., Weikum, G.: From Centralized Workflow Specification to Distributed Workflow Execution. Journal of Intelligent Information Systems 10(2), 159–184 (1998)
13. Nanda, M.G., Chandra, S., Sarkar, V.: Decentralizing Execution of Composite Web Services. In: Proc. of the 19th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications, pp. 170–187. ACM, New York (2004)
14. Nandi, P., Kumaran, S.: Adaptive Business Object - A New Component Model for Business Integration. In: Proc. of the 8th Int. Conf. on Enterprise Information Systems, pp. 179–188 (2005)
15. Nigam, A., Caswell, N.S.: Business Artifacts: An Approach to Operational Specification. IBM Systems Journal 42(3), 428–445 (2003)
16. Reijers, H.A., Limam, S., van der Aalst, W.M.P.: Product-Based Workflow Design. Journal of Management Information Systems 20(1), 229–262
17. Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and Coupling Metrics for Workflow Process Design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004)
18. Ryndina, K., Küster, J.M., Gall, H.: Consistency of Business Process Models and Object Life Cycles. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 80–90. Springer, Heidelberg (2007)
19. Ryndina, K., Küster, J.M., Gall, H.: A Tool for Integrating Object Life Cycle and Business Process Modeling. In: Proc. of the BPM Demonstration Program at the 5th Int. Conf. on Business Process Management. CEUR-WS (2007)
20. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems 10(4), 443–481 (2001)
21. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
22. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of Web Services Composition Languages: The Case of BPEL4WS. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 200–215. Springer, Heidelberg (2003)

Instantiation Semantics for Process Models

Gero Decker[1] and Jan Mendling[2]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
gero.decker@hpi.uni-potsdam.de
[2] Queensland University of Technology, Brisbane, Australia
j.mendling@qut.edu.au

Abstract. Although several process modeling languages allow one to specify processes with multiple start elements, the precise semantics of such models are often unclear, both from a pragmatic and from a theoretical point of view. This paper addresses the lack of research on this problem and introduces the CASU framework. The contribution of this framework is a systematic description of design alternatives for the specification of instantiation semantics of process modeling languages. We classify six of the most prominent languages by the help of this framework. Our work provides the basis for the design of new correctness criteria as well as for the formalization of EPCs and extension of BPMN. It complements research such as the workflow patterns.

1 Introduction

Process modeling techniques have been widely adopted by businesses and other organizations for documenting their operations. In this context, process models describe business activities along with their temporal and logical relationships within business processes of the organization, either as reflection of the status quo or as a road map for change. Process models are also used for configuring information systems, in particular workflow systems, that create and handle singular cases (or instances) according to the rules defined in the model.

There are several business process modeling languages that define the basic elements for constructing individual business process models. In this paper we consider the six most prominent ones and assume that the reader has some basic understanding of their syntax and semantics. They are in historical order:

- Petri nets (PN) [1], a formalism to specify processes with concurrency. In particular, we will focus on Open Workflow Nets (oWFNs) [2] which extend Workflow nets [3] with interface places.
- Event-driven Process Chains (EPCs) [4], the business process modeling language used within the ARIS framework and the respective toolset [5].
- UML Activity Diagrams (UAD) [6], the process modeling language of UML.
- Yet Another Workflow Language (YAWL) [7], the workflow language that builds on the workflow patterns analysis [8].
- Business Process Execution Language for Web Services (BPEL) [9], the proposed OASIS standard for web service composition and execution.

– Business Process Modeling Notation (BPMN) [10], the OMG standard notation for describing business processes.

In practice these languages tend to be used in different design phases: while executable processes and workflows are often defined as BPEL or YAWL models, Petri nets and UAD specify processes in a way that easily supports software development. EPCs and BPMN are meant to serve as a high-level description of business operations. For an introduction to these languages refer to [11].

In this paper we focus on the problem of process instantiation and its representation in process models. This problem is little understood in theory and practice, and it poses a considerable challenge for mapping conceptual models to executable processes. In particular, such conceptual models tend to have a significant amount of control flow errors like deadlocks [12,13]. 57% of these errors in the SAP Reference Model, i.e. 102 out of 178 [14, p.150], can be traced back to an unsound combination of multiple start and end events. The BPMN specification acknowledges that the semantics of multiple start events are often unclear [10, p.36]. Even though there has been a considerable amount of academic contributions on the formalization of control flow constructs in all the six mentioned process modeling languages, these works tend to abstract from the problem of process instantiation. Most notably, the original workflow patterns [8] do not cover any instantiation patterns. A revised set of control-flow patterns [15] discusses the effect of external signals on the execution of a process instance (WCP-23 Transient Trigger, WCP-24 Persistent Trigger), but not on instantiation.

Against this background, this paper provides a twofold contribution. First, we define a conceptual framework to describe process instantiation semantics as assumed in different process modeling languages. This framework is called CASU since it builds on four pillars: instantiation creation (C), control threads activation (A), event subscription (S), and unsubscription (U). Second, we use this framework to classify and compare the instantiation semantics of the six mentioned modeling languages. In particular, this comparison reveals additional problems of mapping BPMN to BPEL beyond those discussed in [16,17,18].

The remainder of this paper is structured as follows. In Section 2 we discuss how process instantiation is represented in Petri nets, EPCs, UML-AD, YAWL, BPEL, and BPMN. We define a set of general concepts to make the approaches comparable. Then, Section 3 introduce the CASU framework to describe different instantiation semantics of process modeling languages. We provide a classification of the languages according to this framework. Section 4 discusses the implications of this work, in particular, its relationship to existing research on the verification of process models as well as potential directions for EPC formalization and BPMN extension. Finally, Section 5 concludes the paper.

2 Background on Process Instantiation

Process Instantiation refers to the action and the rules of creating an instance from a process model. Instantiation requires an initial state to be identified for the newly created instance. In this section we discuss explicit and implicit

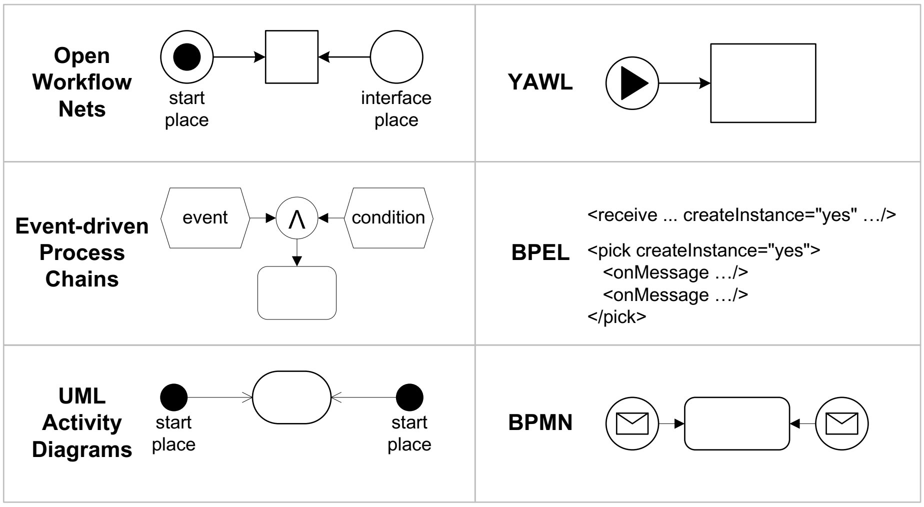


Fig. 1. Entry points in different process modeling languages

definition of an initial state, the role of entry points for it, i.e. start places, start conditions, and start events, as well as the basic architecture for instantiation.

Most prominently, process instantiation requires the definition of the initial state for the new instance. This initial state becomes the starting point for allowed state transitions in the life cycle of the instance. In general there are two ways of defining the initial state of a process instance: explicitly or implicitly. A definition of an initial state is *explicit* if the initial state is part of the definition of the process model. The initial state of a Petri net is traditionally defined explicitly: Murata defines a Petri net as a 5-tuple including places, transitions, arcs, weight function, and the initial marking [19]. The definition of an initial state is *implicit* if it has to be derived from what we call *entry points* of a process model, i.e. model elements without any control flow arc pointing to them. Note that this notion of entry point only refers to the structure of the process model.

Entry points are related to different concepts in process modeling languages, most prominently start places, start conditions, and start events (see Figure 1). In the simplest case, an entry point is a *start place* that receives a control token at the time of instantiation. A *start condition* is a statement about the environment of a process that can be either true or false at a given point in time. Depending on whether that condition yields true or false, the respective entry point is activated, and thus becomes part of the initial state of the instance. Entry points can also refer to *start events*. An event in that sense can be understood as a record of an activity in a system [20]. Therefore, every event has a defined time of occurrence. Start events are special events that respond to records related to a process type.

In some cases the initial state can be derived as unambiguously from entry points as by giving the initial marking of a Petri net explicitly. Modeling languages like Workflow nets and YAWL restrict the number of entry points to one unique node such that the initial state assigns a single token to the unique start

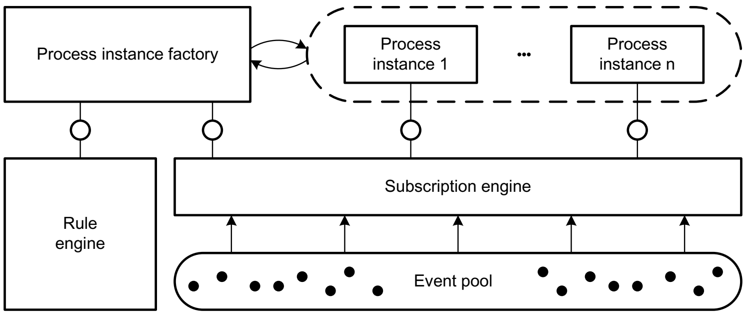


Fig. 2. Process execution environment

place. Things are less clear if there are *multiple entry points* in the model. Open Workflow nets extend Workflow nets with an interface: syntactically they are classified as start events according to our definition of an entry point. Yet, they cannot trigger the creation of a new instance. In UML Activity Diagrams the initial state is derived unambiguously by assigning a control token to each initial node [6]. Note that receive activities in UAD are no entry points according to our definition since they have to receive control from an incoming flow to be activated. In contrast to the mentioned languages, the original definition of EPCs [4] does not define a notion of state. Therefore, it is not a priori clear how a combination of entry points, i.e. EPC start events, maps to an initial state. An unambiguous way of deriving the initial state in this case would be to activate all entry points while creating an instance. In contrast to that, the initial state of an EPC is defined non-deterministically [21,22]. The start events of an EPC are often used to represent both events and conditions [23, p.134]. As a consequence, different initial states are allowed, but there is at most informal information, e.g. in the text labels of the start events, that gives hints when and which initial state has to be used. In BPMN start events can be used (they are optional) to describe instantiation. The specification distinguishes subtypes for message, timer, rule, link, and multiple start events [10]. If there are multiple start events modeled they should be interpreted as independent events, i.e. each event creates a new instance. Still, it is possible to describe the dependency upon multiple events if the events flow to the same activity in the process and this activity specifies the event dependency. In BPEL alternative start events can be defined using the pick activity [9]. Multiple message activities can have the "createInstance" attribute set to "yes". Upon instantiation subscriptions are issued for all those receive and onMessage elements that did not trigger instantiation and that do not belong to the same pick element the triggering activity belonged to.

As a conceptual framework for those cases where start events apply, we assume a subscription infrastructure including a rule engine involved in process instantiation. Process instances and process instance factories can subscribe for particular events and can have conditions evaluated. Process instance factories typically have durable subscriptions for events, i.e. the subscription takes place at deployment time of a process model and unsubscription at the moment of undeployment. As the name indicates, process instance factories create process instances as a result of certain event occurrences. Subscriptions by process instances have a shorter life span. Subscription can only become effective after the moment of process instantiation. Unsubscription can take place any time during the life time of a process instance, however, it must be before termination.

Figure 2 illustrates the subscription framework. Events occur in an event pool and can be observed by a subscription engine. Subscriptions and unsubscriptions can be issued by the different process instances and by the process instance factory. The subscription engine in turn notifies them upon availability of a corresponding event which in turn is then consumed.

3 A Framework for Process Instantiation

This section discusses different design choices for defining process instantiation semantics. We establish a framework building on four aspects of instantiation that have to be specified by a process modeling language:

Creation (C): When has a new instance to be created?
Activation (A): Which entry points are activated?
Subscription (S): For which start events are subscriptions created?
Unsubscription (U): How long are subscriptions kept?

Based on the first letters we refer to the framework as the CASU framework.

3.1 When to Create a New Instance?

In essence we can distinguish cases where the process model does not specify when an instance has to be created (C-1), where the process model defines conditions before an instance can be created (C-2 and C-3), and where the process model specifies in response to which event an instance is created (C-4 and C-5). Please note that it is not reasonable to create an instance when a condition is true. While an event is consumed, a condition would remain true and trigger a cascade of new instances before it becomes false at some stage.

C-1 Ignorance. The process model is ignorant of instantiation condition. The instantiation of a process instance is controlled by the process environment, and no triggering events are defined.

Example: A process model describes that supply needs must be identified before a request for quote is set up. However, it is not defined what triggers the first activity. Figure 3 shows a corresponding YAWL net.

C-2 Single Condition Filter. The start condition of a process model specifies under which circumstances it is possible to create a new process instance.

Example: The start condition of a loan process model specifies that the applicant must be of full age.

C-3 Multi Condition Filter. Multiple start conditions define a complex condition when a process is allowed to be instantiated.

Example: A loan process model of another bank defines two start conditions: the applicant must be of full age and credit card owner (Figure 5).

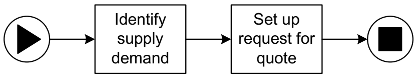


Fig. 3. C-1 Ignorance in a YAWL net

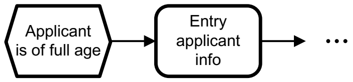


Fig. 4. C-2 Single condition filter in EPC

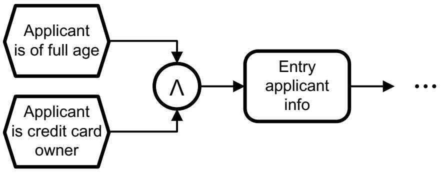


Fig. 5. C-3 Multi condition filter in EPC

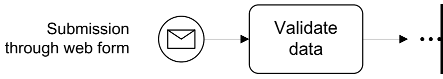


Fig. 6. C-4 Single event trigger in a BPMN diagram

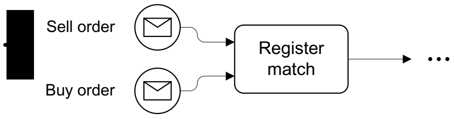


Fig. 7. C-5 Multi event trigger in a BPMN diagram

C-4 Single Event Trigger. The consumption of one start event triggers instantiation.

Example: A Police process model describes that citizens can file charges via a website, triggering instantiation by submitting the web form (Figure 6).

C-5 Multi Event Trigger. Consumption of multiple events triggers instantiation. There is a potential race between different process definitions (factories) in case of overlapping event types. When the last required event becomes available, the instance is created and all required events are consumed at one point in time.

Example: Buy and sell events arising from the stock market are automatically correlated triggering trade processes (Figure 7).

3.2 Which Entry Points Are Activated?

There are different ways to express in a process model which entry points are activated at instantiation time. An initial state (A-1) defines explicitly the activation. Depending on the type of entry points the activation can be specified implicitly: all start places (A-2), true conditions (A-3), occurred events (A-4), or a combination of the latter (A-5).

A-1 Initial State. The process model explicitly defines the state each process instance is initially in.

Example: A model includes an initial marking with several tokens. One of them represents a semaphore, the others two streams of control (Figure 8).

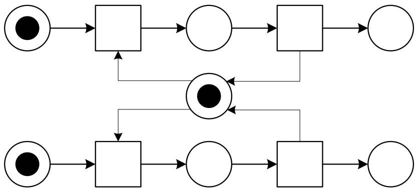


Fig. 8. A-1 Initial state in a Petri net

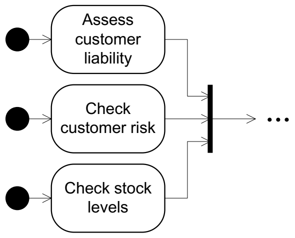


Fig. 9. A-2 All start places in a UML Activity Diagram

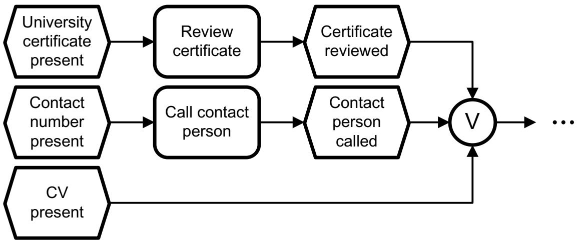


Fig. 10. A-3 True conditions as an EPC

A-2 All Start Places. The process model implicitly defines an initial state through its structure: all start places receive a token at instantiation.

Example: An ordering process model has three start nodes, each receiving a token upon instantiation. These tokens enable the three parallel activities "assess customer liability", "check customer risk" and "check stock levels" (Figure 9).

A-3 True Conditions. The environment checks conditions at instantiation and activates the respective start condition nodes.

Example: A job application process model contains the following start conditions: "University certificate present", "contact number present" and "CV present". Only if a contact phone number is present, the former employer is called for getting further information on the candidate. The university certificate must be reviewed if present and the contact person is called if a phone number is present (Figure 10).

A-4 Occurred Events. In this case all consumed events (one or more) are mapped to an activation of control threads in the process model. There may be start events that do not belong to this set.

Example: An invoice management process model describes four start events (Figure 11): "paper invoice received", "electronic invoice received", "delivery notification received" and a timer event "second Tuesday of the month". Once a pair of corresponding invoice and delivery notification have arrived or an invoice has arrived and the timer event has occurred, a process instance is

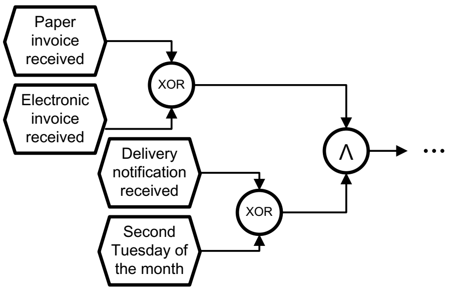

Paper invoice received
Electronic invoice received
XOR
Delivery notification received
Second Tuesday of the month
XOR
Λ
...

Fig. 11. A-4 Occurred events as an EPC

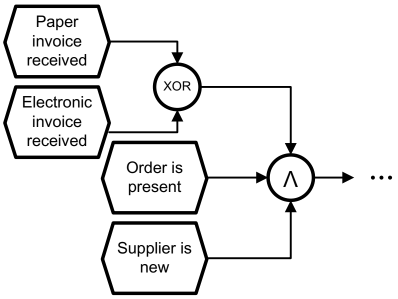


Fig. 12. A-5 Occurred events plus conditions as an EPC

created. Upon instantiation those control threads are activated that originate in the respective start events.

A-5 Occurred Events plus Conditions. In this case all consumed events map to activated control threads. Additionally, branches can be activated if start conditions yield true at instantiation time.

Example: In a second invoice management process model (Figure 12), the start events “paper invoice received” and “electronic invoice received” appear again. Additionally, there are start conditions “order is present” and “supplier is new”. For each start condition that is fulfilled upon instantiation the corresponding control thread is activated.

3.3 For Which Non-activated Start Events Are Subscriptions Created?

When there are start events a decision has to be made whether event subscriptions are made for those remaining start events that did not lead to the instantiation of process. We distinguish the case of subscriptions being created for all of the remaining start events (S-1), for none of them (S-2), or for those that are required for proper execution (S-3).

S-1 All Subscriptions. For those start events that are not activated at instantiation time, there is an event subscription created for the process instance. I.e., the remainder branches may be activated later by respective events.

Example: A couple applies for a mortgage. With opening the case, there are already several events subscriptions activated that matter later like providing sketch of the house, sale contract, etc. (Figure 13).

S-2 No Subscriptions. In this case there are no event subscriptions created for the process instance. I.e., an entry point thread will be either activated at instantiation time or never.

Example: A stock purchase process can be triggered by either a customer representative directly entering the purchase request or by the customer entering the request in a web form (Figure 14).

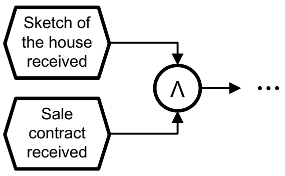


Fig. 13. S-1 All subscriptions as an EPC

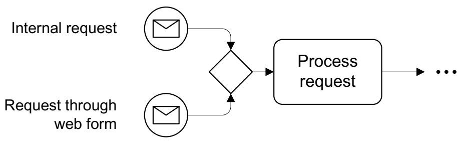


Fig. 14. S-2 No subscriptions in a BPMN diagram

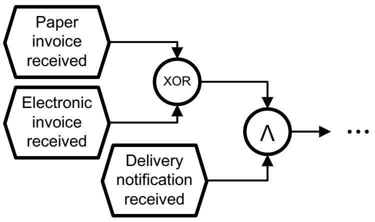


Fig. 15. S-3 Reachable subscriptions as an EPC

S-3 Reachable Subscription. Only those event subscriptions are activated that might be required later to complete the process instance properly.

Example: In an invoice management process model similar to that of A-4 there are three start events: the receipt of a paper invoice, of an electronic invoice or a delivery notification can trigger instantiation. As only one invoice is needed for proper termination, only a subscription for the delivery notification is issued in case the receipt of an invoice triggered instantiation. In the other case, subscriptions for both invoice types are issued (Figure 15). BPEL provide respective functionality with the pick as a start activity.

3.4 How Long Are Subscriptions Kept?

There may be different ways to unsubscribe for events. In the simplest case, they are kept until consumption (U-1) or at least until the process terminates (U-2). Earlier unsubscriptions can be defined based on timers (U-3), on events (U-4), or on proper completion (U-5). Listing 1 shows respective concepts in BPEL.

U-1 Until Consumption. The process cannot terminate before all event subscriptions have led to the consumption of a respective event. A subscription of an instance is never deactivated.

Example: A process model describes the activities of a logistics hub, where containers with RFID tags arrive while routing information for the containers is fed into the system through a different channel. Either the container or its routing information might arrive first, but the process cannot terminate before both are there.

Listing 1. Unsubscription in BPEL

```
<scope>
 <eventHandlers><onAlarm name="timeout" .. /></eventHandlers>
 <flow>
  <receive name="rcvDeliveryNot" .. createInstance="yes">
   <correlations><correlation set="inv" initiate="join"/></correlations>
  </receive>
  <sequence><pick createInstance="yes">
   <onMessage name="rcvEInvoice" .. >
    <correlations><correlation set="inv" initiate="join"/></correlations>
   .. </onMessage>
   <onMessage name="rcvPInvoice" .. >
    <correlations><correlation set="inv" initiate="join"/></correlations>
   .. </onMessage>
  </pick> ..
  <exit/></sequence>
 </flow>
</scope>
```

U-2 Until Termination. As soon as the process fulfills a termination condition, all subscriptions are deactivated, and the process terminates. This seems to be often assumed by EPC modelers.

Example: A BPEL process reaches an exit activity terminating all subscriptions.

U-3 Timer-based. After a certain period of time after instantiation, all or individual event subscriptions are cancelled.

Example: A timeout of a pick activity in a BPEL process deactivates an event subscription.

U-4 Event-based. If one of alternative events is consumed, the others are not more considered, and deactivated.

Example: In an invoice management process model similar to that of A-4 is represented in BPEL. A pick as a start activity defines alternative start events: the receipt of a paper or an electronic invoice or of a paper or an electronic delivery notification. If the receipt of an invoice triggered instantiation and a paper delivery notification arrives, the subscription for an electronic delivery notification is removed. This also applies for the other combinations.

U-5 Proper Completion. An event gets deactivated when proper completion is guaranteed for the current marking if the event is not consumed.

Example: A process model describes how reallocation of passengers to flights of partner airlines works at an airport's service desk. This process model has two start events "Passenger arrives at service desk" and "flight voucher arrives". Immediately upon arrival of the passenger a seat is allocated and immediately upon arrival of an electronic flight voucher this voucher is checked for validity. As vouchers can also be issued in paper form, the passenger might carry this voucher with him. As soon as the voucher is checked, the flight is billed and the passenger can board the aircraft.

3.5 A Classification of Instantiation Semantics

Before assessing the six process modeling languages, the interrelationships between the patterns presented in the CASU framework need to be discussed briefly. While patterns A-1 (Initial state) and A-2 (All start places) do not require support for any particular of the C-patterns, A-3 (True conditions) requires the specification of single or multiple start conditions (C-2, C-3). All patterns related to start events (A-4, A-5, all S-patterns and all U-patterns) rely on the possibility to specify single or multiple event triggers (C-4, C-5). The results of the classification are summarized below in Table 1.

Open Workflow Nets (oWFN) are a particular class of Petri nets that are ignorant of the circumstances of their instantiation (C-1). Furthermore, they define an initial state (A-1). They also include a distinct set of interface places that can be used for message passing. The input places of the interface follow all subscription semantics (S-1) that are kept until completion (U-2).

Start events (also called triggers) are used in *Event-driven Process Chains (EPC)* to represent when a process starts. The cases C-4 (Single Event Trigger) with XOR-join and C-5 (Multiple Event Trigger) with AND-join are described in [5] and [23], but no formalization is available. Although not recommended, the decomposition of EPCs often leads to subprocesses that have conditions as start nodes (C-2 and C-3), e.g. if the subprocess starts immediately after a

Table 1. Instantiation in different process modeling languages

Patterns	oWFN	EPCs	UAD	YAWL	BPEL	BPMN
C-1 Ignorance	+	+	+	+	+	+
C-2 Single Condition Filter	–	+	–	–	–	–
C-3 Multi Condition Filter	–	+	–	–	–	–
C-4 Single Event Trigger	–	+	–	–	+	+
C-5 Multi Event Trigger	–	+	–	–	–	+
A-1 Initial State	+	–	–	–	–	–
A-2 All Start Places	–	–	+	+	–	–
A-3 True Conditions	–	+	–	–	–	–
A-4 Occurred Events	–	+	–	–	+	+
A-5 Occurred Events plus Cond.	–	+	–	–	–	–
S-1 All Subscriptions	+	∅	–	–	+	–
S-2 No Subscriptions	–	∅	–	–	–	+
S-3 Reachable Subscription	–	∅	–	–	+	–
U-1 Until Consumption	–	∅	–	–	+	–
U-2 Until Termination	+	∅	–	–	+	–
U-3 Timer-based	–	∅	–	–	+	–
U-4 Event-based	–	∅	–	–	+	–
U-5 Proper Completion	–	∅	–	–	–	–

decision [5, pp.250] or to express external dependencies [5, pp.131]. If the trigger or condition is not made explicit in the start event label, the EPC remains ignorant of the instantiation (C-1). Depending on which of multiple start events and start conditions apply the respective initial state is derived (A-3, A-4, A-5). The whole area of subscription (S-Patterns) and unsubscription (U-Patterns) related to non-activated entry points has not been explicitly defined for EPCs. There seem to be some inconsistent interpretations that need to be resolved in future work: While Rump assumes that there are no subscriptions [21], the concept of external dependency appears to suggest either S-1 (all subscriptions) or S-3 (reachable subscriptions) [5, pp.131]. In neither case unsubscription is discussed. Table 1 reflects this ambiguity by using the $\emptyset$ character.

Although *UML Activity Diagrams (UAD)* include event consumption and event production as first-class citizens of the language, these concepts are not used in the context of process instantiation. The events required for process instantiation are beyond the scope of UAD models. That way UAD only supports C-1 (Ignorance) among the C-patterns. The start nodes are essentially start places that all receive a token upon instantiation (A-2). The remaining patterns A-3 through A-5, the S-patterns and the U-patterns are not supported.

Yet Another Workflow Language (YAWL) concentrates on the control and data flow within process instances. There is one distinguished "start condition" per process model, however, definitions of how and when instantiation takes place are not part of YAWL models. The notion of start conditions or start events are not present. Therefore, it does not support C-1 through C-4, A-3 through A-5, none of the S-patterns as well as none of the U-patterns. The initial state of a process instance is implicitly given: there is exactly one start place that receives a token upon instantiation (A-2). That way, YAWL is similar to UAD in terms of instantiation semantics with an additional restriction to exactly one start place. Workflow nets share the same instantiation profile with YAWL.

The *Business Process Execution Language (BPEL)* completely lacks the notion of start conditions (C-2, C-3). Process instantiation might be undefined in the case of abstract BPEL (C-1) and must be defined for executable BPEL processes. Here, instantiation is always triggered through individual message receipts (C-4), described in incoming message activities (receive or pick) having the attribute "createInstance" set to yes. Defining combinations of messages that are required for instantiation is not possible (C-5). The start state of a process instance is solely determined by the one start event that triggered process instantiation. Therefore, BPEL does not support A-1, A-2, A-3 and A-5. Subscriptions are issued for all those incoming message activities that have not been involved in process instantiation (S-1). Whenever an onMessage branch of a pick element receives the initial message, no subscriptions are issued for the other onMessage branches of the same pick element. That way, BPEL supports S-3. As illustrated in Listing 1, BPEL supports patterns U-1 through U-4. Termination before having received all start messages can be achieved through the exit element or through throwing exceptions. Timer-based unsubscription (U-3) can be realized by surrounding message activities with a scope that has an onAlarm event handler attached. Event-based unsubscription happens in the context of pick elements (U-4). Beyond these triggers for unsubscription, BPEL does not support pattern U-5.

The *Business Process Modeling Notation (BPMN)* does not include the notion of start conditions. The only entry points available are start events. C-4 (Single Event) is the default case for BPMN processes, where an individual event specified in the model leads to process instantiation. However, no specification might be given, that way realizing Ignorance (C-1). The BPMN specification mentions a special case realizing C-5: Multiple start events are connected to an activity indicating that all start events must have occurred before the activity can start [10, p.36]. BPMN also supports A-4 (Occurred Events) via event-based gateways. However, if C-5 applies there is a slightly different token flow in comparison with the standard semantics of BPMN: While typically each token flowing into an activity leads to a separate activity instance, only one activity instance is created in the presence of the C-5 scenario. All other A-patterns are not supported, in particular, neither A-1 (Initial State), nor A-2 (All Start Places), and the notion of start conditions is absent (A-3 and A-5). Although the BPMN specification is slightly ambiguous regarding multiple start events, we interpret that each start event consumption will lead to a separate process instantiation. No subscriptions for other start events are issued within a newly created process instance (S-2). As a result, BPMN does not support patterns S-1, S-3 and none of the U-patterns.

4 Discussion

In this section we discuss the implications of this research. First, we focus on the suitability of correctness criteria. We then give directions for a formalization of EPC instantiation semantics before finally identifying potential extensions to BPMN. Please note that the formalizations of process modeling languages that we are aware of tend to abstract from the complexity of the instantiation problem, e.g. [24,25,22].

Several *correctness criteria* for process models are available including soundness, relaxed soundness, EPC soundness, and controllability. For an overview see [11]. The classical *soundness* property demands a process to complete properly and to have no dead transitions [3]. It can be used to check process models with unique start and end elements such as Workflow nets and YAWL nets. Multiple start nodes in UAD can be bundled with an AND-join such that it becomes also applicable for them. The *relaxed soundness* property can be used for languages with multiple entry points such as EPCs. It basically requires (1) that an OR-split is introduced to bundle all start elements, and (2) checks whether each node participates in at least one execution sequence that leads to proper completion [26]. The property of *EPC soundness* is stricter: it demands that for every start element there exists an initial marking that guarantees proper completion [22]. This property assumes pattern S-2 (no subscriptions). For oWFNs the property of *controllability* was defined to deal with interface places. An oWFN is essentially controllable if there exists a strategy to interact with it such that it terminates properly [2]. The interesting characteristic of this property is that it is basically applicable for any combinations of subscription and unsubscription patterns including those that consider reachability and proper completion. Still it does not distinguish models for which only one particular strategy exists from those which permit different strategies.

In Section 3.5 we already mentioned that a specification of *EPC instantiation semantics* is missing. The concept of external dependency [5] and its representation as a start event highlights the need to discuss subscription semantics in detail. A start event with external dependency semantics does not trigger the creation of an instance, but defines a point of synchronization with an event from outside the process. We basically see two options to support such external dependencies: either S-3 (reachable subscriptions) or based on S-1 (all subscriptions) with U-5 (proper completion). In the case of S-3 those event subscriptions are activated that might be required later to complete the process instance properly. In this case a reachability graph analysis, e.g. using [22], would be required. While this solution would prevent some deadlocks at AND-joins that merge paths from start events, it still allows problems with lack of synchronization. In case of S-1 with U-5 some of the latter problems can be avoided since events get unsubscribed if no more needed. Beyond this aspect of the semantics, one has to carefully select a state representation for the subscriptions. If a subscription is defined like a special activity that is active, this has consequences for downstream OR-joins: they keep waiting for the event to occur, potentially forever if the event cannot occur anymore. Therefore, it would be preferable that event subscriptions were not visible in the state representation of this case.

The status of *BPMN* as a standards proposal raises questions how and whether it should support more of the CASU patterns. An important consideration in this regard is most likely to extend it such that it remains consistent with the current semantics. The support of the start condition patterns (C-2 and C-3) would require either the introduction of a new element or the redefinition of the rule events. These options either affect the metamodel or the current semantics which is both undesirable. If the creation support (C-1 to C-5) remains unchanged, also the activation patterns (A-1 to A-5) stay the same. With respect to the subscriptions (S-1 and S-3) there are basically two options: either changing the instantiation semantics of BPMN, or to add a subscription attribute to start events. The latter seems more attractive from a consistency perspective. Using a keepSubscription attribute, one would be able to specify S-1 (all subscriptions) and S-2 (no subscriptions). By using a further attribute subscriptionGroup one would be able to specify instantiation behavior similar to BPEL: all start events with the same subscription group assigned would correspond to message receive activities within one pick element. If there is only one start event for a group, it corresponds to a plain receive activity. Clearly, these concepts require correlation mechanisms such as identified in [27]. Unsubscriptions could equally be captured by additional attributes, e.g. by setting subscriptionTimeout (U-3) and properTermination (U-5) attributes. An event-based unsubscription (U-4) can be handled using the previously mentioned subscriptionGroup: as soon as an event of the group occurs, the others are unsubscribed. As BPEL has a more sophisticated profile in terms of subscription and unsubscription patterns (cf. [1]), an extension of BPMN with these aspects could simplify the automatic transformation between the languages.

5 Conclusions

Up to now there has been hardly any research dedicated to instantiation semantics of process models. In this paper we have addressed this research gap and introduced the CASU framework. This framework distinguishes the specification of when to create a new process instance (C), of which control threads to be activated upon instantiation (A), of which remaining start events to subscribe for (S), and of when to unsubscribe from these events (U). It builds on general, language-independent concepts and offers a tool for the systematic description and comparison of instantiation semantics of process modeling languages. As such it complements other works such as the workflow patterns.

Based on the CASU framework, we have classified six of the most prominent languages according to their instantiation semantics. In particular, the different profiles of BPMN and BPEL reveal a source of mapping problems between these two languages that has not been identified before. Furthermore, we have shown that the framework provides a basis to discuss the suitability of correctness criteria, the formalization of EPCs, and potential extensions to BPMN. In future research we aim to utilize the CASU framework for analyzing control-flow errors in EPCs. This could lead to new insights regarding which instantiation semantics process modelers assume. In this regard, the explicit description of instantiation semantics by the help of the CASU framework might eventually help to reduce ambiguity and the number of errors in conceptual process modeling.

References

1. Petri, C.: Fundamentals of a Theory of Asynchronous Information Flow. In: 1962 IFIP Congress, pp. 386–390. North-Holland, Amsterdam (1962)
2. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting ws-bpel processes using flexible model generation. Data Knowl. Eng. 64, 38–54 (2008)
3. van der Aalst, W.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
4. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
5. Davis, R.: Business Process Modelling With Aris: A Practical Guide (2001)
6. Object Management Group: UML 2.0 Superstructure Specification (2005)
7. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems 30, 245–275 (2005)
8. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)

9. Alves, A., et al.: Web services business process execution language 2.0 (2007)
10. Object Management Group: Business Process Modeling Notation (2006)
11. Weske, M.: Business Process Management. Springer, Heidelberg (2007)
12. Mendling, J., Verbeek, H., van Dongen, B., van der Aalst, W., Neumann, G.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. Data & Knowledge Engineering 64, 312–329 (2008)
13. Mendling, J., Neumann, G., van der Aalst, W.: Understanding the occurrence of errors in process models based on metrics. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 113–130. Springer, Heidelberg (2007)
14. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
15. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22 (2006)
16. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.: Translating standard process models to bpel. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 417–432. Springer, Heidelberg (2006)
17. Mendling, J., Lassen, K., Zdun, U.: Transformation strategies between block-oriented and graph-oriented process modelling languages. International Journal of Business Process Integration and Management 3 (2008)
18. van der Aalst, W., Lassen, K.: Translating unstructured workflow processes to readable bpel: theory and implementation. Inf. Softw. Techn. 50, 131–159 (2008)
19. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77, 541–580 (1989)
20. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading (2001)
21. Rump, F.: Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozess-ketten - Formalisierung, Analyse und Ausführung von EPKs. Teubner (1999)
22. Mendling, J., van der Aalst, W.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 439–453. Springer, Heidelberg (2007)
23. Scheer, A.W., Thomas, O., Adam, O.: Process Modeling Using Event-Driven Process Chains. In: Process Aware Information Systems, pp. 119–146 (2005)
24. Eshuis, R., Wieringa, R.: Tool support for verifying UML activity diagrams. IEEE Trans. Software Eng. 30, 437–447 (2004)
25. Puhlmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 145–160. Springer, Heidelberg (2006)
26. Dehnert, J., Aalst, W.: Bridging The Gap Between Business Models And Workflow Specifications. International J. Cooperative Inf. Syst. 13, 289–332 (2004)
27. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)

A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows

Xiao Liu, Jinjun Chen, and Yun Yang

Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Melbourne, Australia 3122
{xliu,yyang,jchen}@swin.edu.au

Abstract. In scientific workflow systems, temporal consistency is critical to ensure the timely completion of workflow instances. To monitor and guarantee the correctness of temporal consistency, temporal constraints are often set and then verified. However, most current work adopts user specified temporal constraints without considering system performance, and hence may result in frequent temporal violations that deteriorate the overall workflow execution effectiveness. In this paper, with a systematic analysis of such problem, we propose a probabilistic strategy which is capable of setting coarse-grained and fine-grained temporal constraints based on the weighted joint distribution of activity durations. The strategy aims to effectively assign a set of temporal constraints which are well balanced between user requirements and system performance. The effectiveness of our work is demonstrated by an example scientific workflow in our scientific workflow system.

Keywords: Scientific Workflow, Temporal Constraints, Temporal Constraint Setting, Probabilistic Strategy.

1 Introduction

Scientific workflow is a new special type of workflow that often underlies many large-scale complex e-science applications such as climate modelling, structural biology and chemistry, medical surgery or disaster recovery simulation [18][25]. Real world scientific as well as business processes normally stay in a temporal context and are often time constrained to achieve on-time fulfilment of certain scientific or business targets. Furthermore, scientific workflows are usually deployed on the high performance computing infrastructures, e.g. cluster, peer-to-peer and grid computing, to deal with huge number of data intensive and computation intensive activities [24][25]. Therefore, as an important dimension of workflow QoS (Quality of Service) constraints, temporal constraints are often set to ensure satisfactory efficiency of scientific workflow executions [5][9]. Temporal constraints mainly include three types, i.e. upper bound, lower bound and fixed-time. An upper bound constraint between two activities is a relative time value so that the duration between them must be less than or equal to it. As discussed in [6], conceptually, a lower bound constraint is symmetrical to an upper bound constraint and a fixed-time constraint can be viewed as a special case of upper bound constraint, hence they can be treated similarly. Therefore, in this paper, we focus on upper bound constraints only. As an important means to facilitate temporal QoS, many efforts have been dedicated to temporal verification for workflows in recent years. Different approaches for checkpoint selection and dynamic temporal verification are proposed to improve the efficiency of temporal verification with given temporal constraints [6][10][17]. However, with the assumption that temporal constraints are pre-defined, most papers focus on run-time temporal verification while neglecting the fact that efforts put at run-time will be mostly in vain without build-time setting of high quality temporal constraints. The reason is obvious since the purpose of temporal verification is to identify potential violations of temporal constraints to minimise the exception handling cost. Therefore, if temporal constraints are of low quality themselves, temporal violations are highly expected no matter how much efforts have been put by temporal verification.

The task of setting temporal constraints described in this paper is to assign a set of coarse-grained and fine-grained upper bound temporal constraints to scientific workflows. Here, *coarse-grained constraints* refer to those assigned to the entire workflow or workflow segments, while *fine-grained constraints* refer to those assigned to individual activities. However, although coarse-grained constraints can be deemed as the collection of fine-grained constraints, they are not in a simple relationship of linear culmination and decomposition. To ensure on-time fulfilment of workflow instances, both coarse-grained and fine-grained temporal constraints are required, especially when scientific workflows are deployed on dynamic computing infrastructures, e.g. grid, where the performance of the underlying resources is highly uncertain [18]. Here, the quality of temporal constraints can be measured by at least two criteria: 1) well balanced between user requirements and system performance; 2) well supported for both overall coarse-grained control and local fine-grained control. A detailed illustration will be presented in Section 2.

In this paper, a probabilistic strategy for setting both coarse-grained and fine-grained temporal constraints is proposed. With a novel probability based temporal consistency which utilises the weighted joint distribution of activity durations, our strategy supports an iterative and interactive negotiation process between the client (e.g. a user) and the service provider (e.g. a workflow system) for setting coarse-grained temporal constraints. Afterwards, fine-grained temporal constraints associated with each activity can be derived automatically. In addition, the weighted joint distribution of four basic Stochastic Petri Nets [1] (SPN) based building blocks, i.e. sequence, iteration, parallelism and choice, is presented to enhance the efficiency of calculating the overall weighted joint distribution through their compositions. The effectiveness of our strategy is further demonstrated by an example scientific workflow of weather forecast in our scientific workflow management system, i.e. SwinDeW-G (**Swin**burne **De**centralised **W**orkflow for **G**rid) [23].

The remainder of the paper is organised as follows. Section 2 presents a motivating example and the problem analysis. Section 3 proposes novel probability based temporal consistency. Section 4 presents the probabilistic strategy for setting temporal constraints. Section 5 further demonstrates the setting process with the motivating example to verify the effectiveness of our strategy. Section 6 introduces the implementation of the strategy in our scientific workflow system. Section 7 presents the related work. Finally, Section 8 addresses our conclusion and future work.

2 Motivating Example and Problem Analysis

In this section, we introduce a weather forecast scientific workflow to demonstrate the problem of setting temporal constraints. In addition, two basic requirements for setting high quality temporal constraints are presented.

The entire weather forecast workflow contains hundreds of data intensive and computation intensive activities. Major data intensive activities include the collection of meteorological information, e.g. surface data, atmospheric humidity, temperature, cloud area and wind speed from satellites, radars and ground observatories at distributed geographic locations. These data files are transferred via various kinds of network. Computation intensive activities mainly consist of solving complex meteorological equations, e.g. meteorological dynamics equations, thermodynamic equations, pressure equations, turbulent kinetic energy equations and so forth which require high performance computing resources. Due to the space limit, it is not possible to present the whole forecasting process in detail. Here, we only focus on one of its segments for radar data collection. As depicted in Figure 1, this workflow segment contains 12 activities which are modeled by SPN with additional graphic notations as illustrated in Sections 4 and 6. For simplicity, we denote these activities as X_1 to X_{12}. The workflow process structures are composed with four SPN based building blocks, i.e. a choice block for data collection from two radars at different locations (activities $X_1 \sim X_4$), a compound block of parallelism and iteration for data updating and pre-processing (activities $X_6 \sim X_{10}$), and two sequence blocks for data transferring (activities X_5, X_{11}, X_{12}).

It is evident that the duration of these scientific workflow activities are highly dynamic in nature due to their data complexity and the computation environment. However, to ensure the weather forecast can be broadcast on time, every scientific workflow instances must be completed within a specific time duration. Therefore, a set of temporal constraints must be set to monitor the overall workflow execution time. For our example workflow segment, to ensure that the radar data can be collected in time and transferred for further processing, at least one overall upper bound temporal constraint is required. However, a coarse-grained temporal constraint is not effective enough to ensure fine-grained workflow execution, i.e. the completion time of each activity. It is evidently that without the support of local enforcements, the overall workflow duration can hardly be guaranteed. For example, we set a two hour temporal constraint for this radar data collection process. But due to some technical problems, the connection to the two radars are broken and blocked in a state of retry

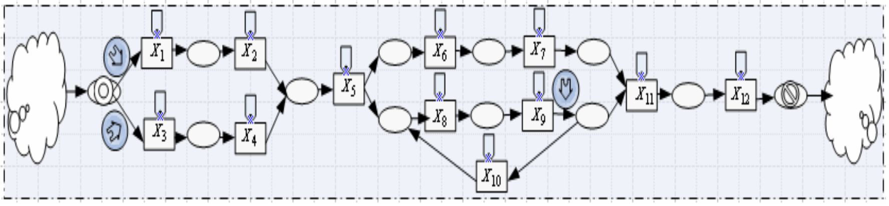


Fig. 1. Example scientific workflow segment

and timeout for more than 30 minutes whilst its normal duration should be far less. Therefore, the two hour overall temporal constraint for this workflow segment will probably be violated since its subsequent activities normally require more than 90 minutes to accomplish. However, no actions were taken yet due to the ignorance of the fine-grained temporal constraints on these connection activities. The exception handling cost for compensation of this time deficit, e.g. workflow re-scheduling and recruitment of additional resources, is hence inevitable. That is why we also need to set fine-grained temporal constraints to each activity. Specifically, for this example workflow segment, an overall coarse-grained temporal constraint and 12 fine-grained temporal constraints for activities X_1 to X_{12} are required to be set.

However, setting temporal constraints is not a straightforward task, many factors such as workflow structures, system performance and user requirements should be taken into consideration. Here, we present the basic requirements of the setting strategy by analysing two criteria for high quality temporal constraints.

1) Temporal constraints should be well balanced between user requirements and system performance. It is common that clients often suggest coarse-grained temporal constraints based on their own interest while with limited knowledge about the actual performance of workflow systems. With our example, it is not rational to set a 60 minutes temporal constraint to the segment which normally needs two hours to finish. Therefore, user specified constraints are normally prone to cause frequent temporal violations. To address this problem, a negotiation process between the client and the service provider who is well aware of the system performance is desirable to achieve balanced coarse-grained temporal constraints that both sides are satisfied with.

2) Temporal constraints should facilitate both overall coarse-grained control and local fine-grained control. As analysed above, this criterion actually means that the strategy should support setting both coarse-grained temporal constraints and fine-grained temporal constraints. However, although the overall workflow process is composed of individual workflow activities, coarse-grained temporal constraints and fine-grained temporal constraints are not in a simple relationship of linear culmination and decomposition. Meanwhile, it is impractical to set fine-grained temporal constraints manually for a large amount of activities in scientific workflows. Since coarse-grained temporal constraints can be obtained through the negotiation process, the problem to be addressed here is how to automatically derive the local fine-grained temporal constraints for each activity.

To conclude, the basic requirements for setting high quality temporal constraints can be simply put as effective negotiation for coarse-grained temporal constraints and automatic assignment for fine-grained temporal constraints. However, to our best knowledge, very little efforts have been dedicated to set high quality temporal constraints in scientific workflows. In this paper, we propose a probabilistic strategy which targets at the two requirements.

3 Probability Based Temporal Consistency

In this section, we propose a novel probability based temporal consistency which utilise the weighted joint distribution of workflow acitivity durations to facilitate setting temporal constraints. To define the weighted joint distribution of workflow acitivity durations, we first present two assumptions on the probability distribution of individual activity duration.

Assumption 1: The distribution of activity durations can be obtained from workflow system logs. Without losing generality, we assume all the activity durations follow the normal distribution model, which can be denoted as $N(\mu, \sigma^2)$ where μ is the expected value and σ^2 is the variance where σ is the standard deviation [21].

Assumption 2: The activity durations are independent to each other.
For the convenience of analysis, assumption 1 chooses normal distribution to model the activity durations. If most of the activity durations follow non-normal distribution, e.g. Gamma distribution, lognormal distribution or Beta distribution [16], our strategy can still be applied in a similar way with minor differences of their joint distribution. Furthermore, as it is commonly applied in the area of system simulation and performance analysis, assumption 2 requires that the activity durations be independent from each other to facilitate the analysis of joint normal distribution. For those which do not follow the above assumptions, they can be treated by normal transformation and correlation analysis [16], or moreover, they can be ignored first when calculating joint distribution and then added up afterwards.

Furthermore, we present an important formula of joint normal distribution.

Formula 1: If there are n independent variables of $X_i \sim N(\mu_i, \sigma_i^2)$ and n real numbers θ_i, where n is a limited natural number, then the joint distribution of these variables can be obtained with the following formula [21]:

$$Z = \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n = \sum_{i=1}^{n} \theta_i X_i \sim N\left(\sum_{i=1}^{n} \theta_i \mu_i, \sum_{i=1}^{n} \theta_i^2 \sigma_i^2 \right) \qquad (1)$$

Based on this formula, we define the weighted joint distribution of workflow acitivity durations as follows.

Definition 1: (Weighted joint distribution). For a scientific workflow process SW which consists of n activities, we denote the activity duration distribution of activity a_i as $N(\mu_i, \sigma_i^2)$ with $1 \le i \le n$. Then the weighted joint distribution is defined as $N(\mu_{sw}, \sigma_{sw}^2) = N\left(\sum_{i=1}^{n} w_i \mu_i, \sum_{i=1}^{n} w_i^2 \sigma_i^2 \right)$, where w_i stands for the weight of activity a_i that denotes the choice probability or iteration times associated with the workflow path where a_i belongs to.

The weight of each activity with different kinds of workflow structures will be further illustrated in Section 4 by the calculation of weighted joint distribution for basic SPN based building blocks. The weighted joint distribution enables us to analyse the completion time of the entire workflow from an overall perspective. Here, we need to define some notations. For a workflow activity a_i, its maximum duration and minimum duration are defined as $D(a_i)$ and $d(a_i)$ respectively. For a scientific workflow process SW which consists of n activities, its upper bound temporal constraint is denoted as $U(SW)$ with the value of $u(SW)$ [6][10]. In addition, we employ the "3σ"rule which has been widely used in statistical data analysis to specify the possible intervals of activity durations. The "3σ"rule depicts that for any sample comes from normal distribution model, it has a probability of 99.73% to fall into the range of $[\mu-3\sigma, \mu+3\sigma]$ which is a systematic interval of 3 standard deviation around the mean [21]. Therefore, in this paper, we define the maximum duration as $D(a_i)=\mu_i+3\sigma_i$ and the minimum duration as $d(a_i)=\mu_i-3\sigma_i$. Accordingly, samples from the system logs which are above $D(a_i)$ or below $d(a_i)$ are hence discarded as outliers. Now, we propose the definition of probability based temporal consistency which is based on the weighted joint distribution of activity durations. To be noted that, since we deal with setting temporal constraints in this paper, here we only present the definition of build-time temporal consistency.

Definition 2: (Probability based temporal consistency).

At build-time stage, $U(SW)$ is said to be:

1) Absolute Consistency (AC), if $\sum_{i=1}^{n} w_i(\mu_i+3\sigma_i) \le u(SW)$;

2) Absolute Inconsistency (AI), if $\sum_{i=1}^{n} w_i(\mu_i-3\sigma_i) \ge u(SW)$;

3) $\alpha\%$ Consistency ($\alpha\%$ C), if $\sum_{i=1}^{n} w_i(\mu_i+\lambda\sigma_i) = u(SW)$.

Here w_i stands for the weight of activity a_i, λ ($-3 \le \lambda \le 3$) is defined as the $\alpha\%$ confidence percentile with the cumulative normal distribution function of $F(\mu_i+\lambda\sigma_i)=\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{\mu_i+\lambda\sigma_i} \hbar^{-(x-\mu_i)^2 / 2\sigma_i^2} \bullet dx = \alpha\%$ ($0<\alpha<100$). As depicted in Figure 2, if we apply the "3σ"rule to the conventional discrete multiple temporal consistency [7], i.e. strong consistency (SC), weak consistency (WC), weak inconsistency (WI) and strong inconsistency (SI), the two discrete states of WI and WC are actually replaced by continuous consistency states $\alpha\%$ C which compose a Gaussian curve the same as the cumulative normal distribution [21]. The other two consistency states outside the interval are basically the same but also with continuous values infinitely approaching 100% or 0% respectively. However, in order to distinguish them from conventional strong consistency and strong inconsistency, we name them absolute consistency (AC) and absolute inconsistency (AI). Evidently, the prerequisite for this definition is the calculation of weighted joint distribution.

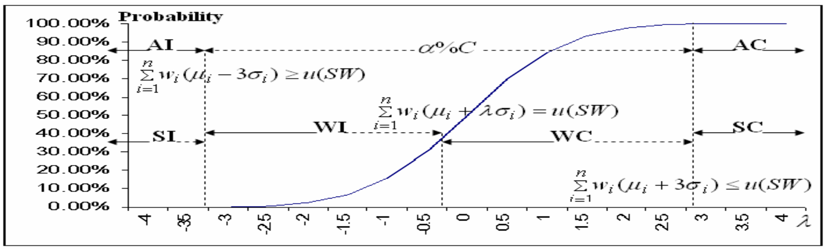


Fig. 2. Probability based temporal consistency

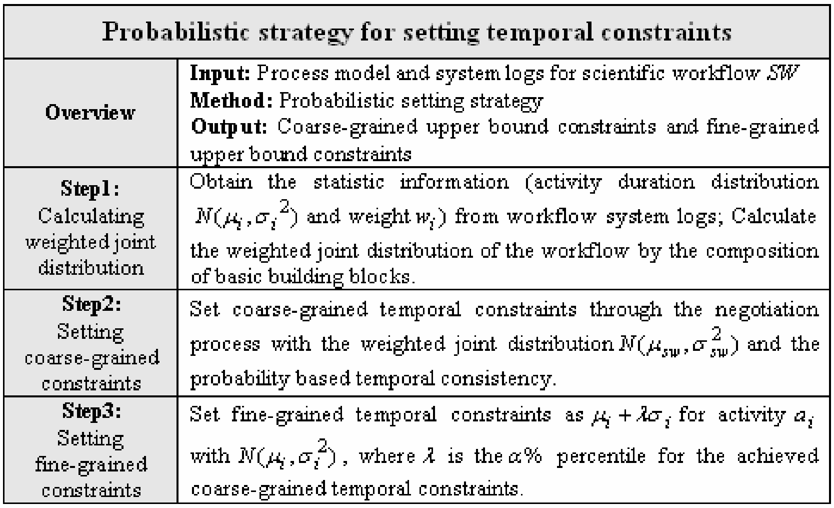
The purpose of probability based temporal consistency is to facilitate the effectiveness of setting temporal constraints. The reason why conventional discrete multiple states based temporal consistency is not suitable here can be explained from two aspects. First, clients normally cannot distinguish between qualitative expressions such as weak consistency and weak inconsistency due to the lack of background knowledge, and thus deteriorates the negotiation process for setting coarse-grained temporal constraints. Second, since each discrete temporal consistency state is actually defined with a coarse-grained interval, it cannot support fine-grained setting. Hence, we propose the probability based continuous temporal consistency. A probability value such as 80% or 90% gives much more sense than a qualitative expression, and any fine-grained temporal consistency state is represented by a unique probability value rather than the previous coarse-grained qualitative expression for each interval. Therefore, the probability based temporal consistency supports the setting of both coarse-grained and fine-grained temporal constraints.

4 Probabilistic Strategy for Setting Temporal Constraints

In this section, we present our probabilistic strategy for setting temporal constraints. The strategy aims to effectively achieve a set of coarse-grained and fine-grained temporal constraints which are well balanced between user requirements and system performance. As depicted in Table 1, the strategy requires the input of process model and system logs. It consists of three steps, i.e. calculating weighted joint distribution of activity durations, setting coarse-grained temporal constraints and setting fine-grained temporal constraints. We illustrate them accordingly as follows.

The first step is to calculate weighted joint distribution. The statistic information, i.e. activity duration distribution and activity weight, can be obtained from system logs by statistical analysis [1][21]. Here, to illustrate and facilitate the calculation of the weighted joint distribution, we analyse basic SPN based building blocks,

Table 1. Probabilpistic setting strategy



Probabilistic strategy for setting temporal constraints	
Overview	**Input:** Process model and system logs for scientific workflow *SW* **Method:** Probabilistic setting strategy **Output:** Coarse-grained upper bound constraints and fine-grained upper bound constraints
Step1: Calculating weighted joint distribution	Obtain the statistic information (activity duration distribution $N(\mu_i, \sigma_i^2)$ and weight w_i) from workflow system logs; Calculate the weighted joint distribution of the workflow by the composition of basic building blocks.
Step2: Setting coarse-grained constraints	Set coarse-grained temporal constraints through the negotiation process with the weighted joint distribution $N(\mu_{sw}, \sigma_{sw}^2)$ and the probability based temporal consistency.
Step3: Setting fine-grained constraints	Set fine-grained temporal constraints as $\mu_i + \lambda\sigma_i$ for activity a_i with $N(\mu_i, \sigma_i^2)$, where λ is the α% percentile for the achieved coarse-grained temporal constraints.

i.e. sequence, iteration, parallelism and choice. These four building blocks consist of basic control flow patterns and are widely used in workflow modelling and structure analysis [2][4][19]. Most workflow process models can be easily built by their compositions, and similarly for the weighted joint distribution of most workflow processes. Here, SPN based modelling is employed to incorporate time and probability attributes with additional graphic notations, e.g. stands for the probability of the path and stands for the normal duration distribution of the associated activity. For simplicity, we consider two paths for the iteration, parallelism and choice building blocks, except the sequence building block which has only one path by nature. However, the results can be extended to more paths in a similar way.

1) Sequence building block. As depicted in Figure 3, the sequence building block is composed by adjacent activities from a_i to a_j in a sequential relationship which means the successor activity will not be executed until its predecessor activity is finished. The weight for each activity in the sequence building block is 1 since they only need to be executed once. Therefore, according to Formula 1, the weighted joint distribution is $Z = \sum_{k=i}^{j} X_k \sim N\left((\sum_{k=i}^{j} \mu_k), (\sum_{k=i}^{j} \sigma_k^2) \right)$.

2) Iteration building block. As depicted in Figure 4, the iteration building block contains two paths which are executed iteratively until certain end conditions are satisfied. If the probability of meeting the end conditions for a single iteration is γ as denoted by the probability notation, then the lower path is expected to be executed for $1/r$ times and hence the upper path is executed for $(1/r)+1$ times. Accordingly, the weight for each activity in the iteration building block is the expected execution times of the path it belongs to. Therefore, the weighted joint distribution here is $Z=((1/\gamma)+1)\left(\sum_{p=i}^{j} X_p\right)+(1/\gamma)\left(\sum_{q=k}^{l} X_q\right) \sim N\left(\left(((1/\gamma)+1)(\sum_{p=i}^{j}\mu_p)+(1/\gamma)(\sum_{q=k}^{l}\mu_q)\right), \left(((1/\gamma)+1)^2(\sum_{p=i}^{j}\sigma_p^2)+(1/\gamma)^2(\sum_{q=k}^{l}\sigma_q^2)\right)\right)$.

3) Parallelism building block. As depicted in Figure 5, the parallelism building block contains two paths which are executed in a parallel relationship. The overall completion time of the parallelism building block is dominated by the path with the longer duration. Hence the joint distribution of this building block equals the joint distribution of the path with a lager expected total duration, that is if $\sum_{p=i}^{j} \mu_p \geq \sum_{q=k}^{l} \mu_q$ then $Z = \sum_{p=i}^{j} \mu_p$, otherwise $Z = \sum_{q=k}^{l} \mu_q$. Evidently, the weight for each activity on the

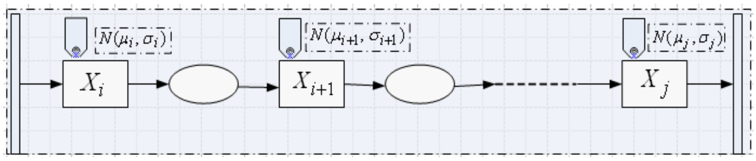


Fig. 3. Sequence building block

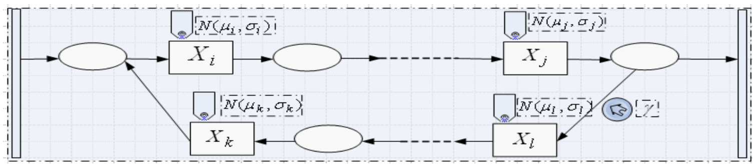


Fig. 4. Iteration building block

path with longer duration is 1 while on the other path is 0 since they do not contribute to the joint distribution. Therefore, the weighted joint distribution of this block is $Z = \begin{cases} \sum_{p=i}^{j} X_p \sim N\left(\sum_{p=i}^{j} \mu_p, \sum_{p=i}^{j} \sigma_p^2\right), & if \sum_{p=i}^{j} \mu_p \geq \sum_{q=k}^{l} \mu_q \\ \sum_{q=k}^{l} X_q \sim N\left(\sum_{q=k}^{l} \mu_q, \sum_{q=k}^{l} \sigma_q^2\right), & otherwise \end{cases}$.

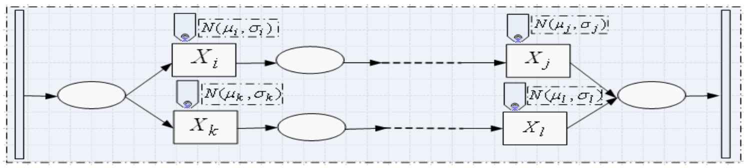


Fig. 5. Parallelism building block

4) Choice building block. As depicted in Figure 6, the choice building block contains two paths in an exclusive relationship which means only one path will be executed at run-time. The probability notation denotes that the probability for the choice of the upper path is β and hence the probability for the lower path is $1-\beta$. The weight for each activity in the choice building block is hence the probability of the path it belongs to. Therefore, the weighted joint distribution is $Z=\beta(\sum_{p=i}^{j} X_p)+(1-\beta)(\sum_{q=k}^{l} X_q)\sim N\left(\beta(\sum_{p=i}^{j}\mu_p)+(1-\beta)(\sum_{q=k}^{l}\mu_q), \beta^2(\sum_{p=i}^{j}\sigma_p^2)+(1-\beta)^2(\sum_{q=k}^{l}\sigma_q^2)\right)$.

The second step is to set coarse-grained temporal constraints. Based on the four basic building blocks, the weighted joint distribution of an entire workflow or workflow segment can be obtained efficiently to facilitate the negotiation process for setting coarse-grained temporal constraints. Here, we denote the obtained weighted joint distribution of the target scientific workflow (or workflow segment) SW as $N(\mu_{sw},\sigma_{sw}^2)$ and assume the minimum threshold is $\beta\%$ for the probability consistency which implies client's acceptable bottom-line probability for timely completion of the workflow instance. The actual negotiation process starts with the client's initial suggestion of an upper bound temporal constraint of $u(SW)$ and the evaluation of the corresponding temporal consistency state by the service provider. If $u(SW)=\mu_{sw}+\lambda\sigma_{sw}$ with λ as the $\alpha\%$ percentile, and $\alpha\%$ is below the threshold

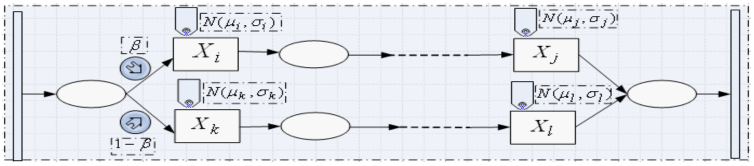


Fig. 6. Choice building block

of $\beta\%$, then the upper bound temporal constraint needs to be adjusted, otherwise the negotiation process terminates. The subsequent process is the iteration that the client proposes new upper bound temporal constraint which is less constrained as the previous one and the service provider re-evaluates the consistency states, until it reaches or is above the minimum probability threshold.

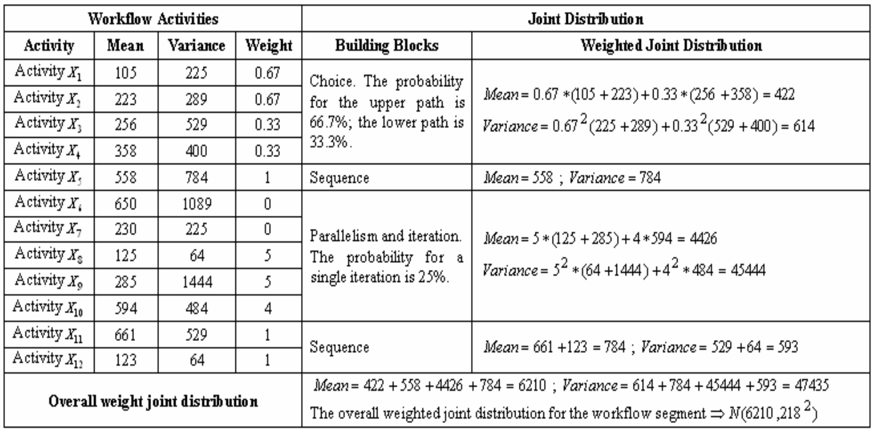
The third step is to set fine-grained temporal constraints. In fact, this process is straight forward with the probability based temporal consistency. Since our temporal consistency actually defines that if all the activities are executed with the duration of $\alpha\%$ probability and their total weighted duration equals their upper bound constraint, we say that the workflow process is $\alpha\%$ consistency at build-time. For example, if the obtained probability consistency is 90% with the percentile λ of 1.28 (the percentile value can be obtained from any normal distribution table or most statistic program [21]), it means that all activities are expected for the duration of 90% probability. Therefore, after the achievement of the coarse-grained temporal constraint, the fine-grained temporal constraint for activity a_i with $N(\mu_i, \sigma_i^2)$ is derived as $\mu_i + \lambda\sigma_i$ automatically. In the case of 90% consistency, it is $\mu_i + 1.28\sigma_i$.

5 Evaluation

In this section, we evaluate the effectiveness of our strategy by further illustrating the motivating example introduced in Section 2. The process model is the same as depicted in Figure 1. As presented in Table 1, the first step is to calculate the weighted joint distribution. Based on statistical analysis and the "3σ"rule, the normal distribution model and its associated weight for each activity duration are specified through statistical analysis of accumulated system logs. Therefore, as depicted in Table 2, the weighted joint distribution of each building block can be derived instantly with their formulas proposed in Section 4. We obtain the weighted joint distribution for the workflow segment as $N(6210, 218^2)$ with second as the basic time unit. The detailed specification of the workflow segment is presented in Table 2.

The second step is the negotiation process for setting an overall upper bound temporal constraint for this workflow segment. Here, we assume that the client's expected minimum threshold of the probability consistency state be 80%. The client starts to propose an upper bound temporal constraint of 6300s, based on the weighted joint distribution of $N(6210, 218^2)$ and the cumulative normal distribution function, the service provider can obtain the percentile as $\lambda = 0.41$ and reply with the probability of 66% which is lower than the threshold. Hence the service provider advises the

Table 2. Specification of the workflow segment

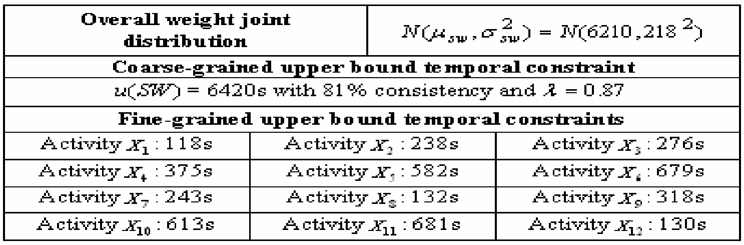


Workflow Activities				Joint Distribution	
Activity	Mean	Variance	Weight	Building Blocks	Weighted Joint Distribution
Activity X_1	105	225	0.67	Choice. The probability for the upper path is 66.7%; the lower path is 33.3%.	$Mean = 0.67*(105+223)+0.33*(256+358) = 422$; $Variance = 0.67^2(225+289)+0.33^2(529+400) = 614$
Activity X_2	223	289	0.67		
Activity X_3	256	529	0.33		
Activity X_4	358	400	0.33		
Activity X_5	558	784	1	Sequence	$Mean = 558$; $Variance = 784$
Activity X_6	650	1089	0	Parallelism and iteration. The probability for a single iteration is 25%.	$Mean = 5*(125+285)+4*594 = 4426$; $Variance = 5^2*(64+1444)+4^2*484 = 45444$
Activity X_7	230	225	0		
Activity X_8	125	64	5		
Activity X_9	285	1444	5		
Activity X_{10}	594	484	4		
Activity X_{11}	661	529	1	Sequence	$Mean = 661+123 = 784$; $Variance = 529+64 = 593$
Activity X_{12}	123	64	1		
Overall weight joint distribution				$Mean = 422+558+4426+784 = 6210$; $Variance = 614+784+45444+593 = 47435$ The overall weighted joint distribution for the workflow segment $\Rightarrow N(6210, 218^2)$	

client to relax the temporal constraint. Afterwards, for example, the client proposes a series of new candidate upper bound temporal constraints one after another, e.g. 6360s, 6390s and 6400s, and the service provider replies with 75%, 79% and 81% as the corresponding temporal consistency states. Therefore, through this negotiation process, the final negotiation result could be an upper bound temporal constraint of 6400s with a probability consistency state of 81%.

The third step is to set the fine-grained constrains for each workflow activity with the obtained overall upper bound constraint. As we mentioned in Section 4, the probability based temporal consistency defines that the probability for each expected activity duration is the same as the probability consistency state of the workflow process. Therefore, since the coarse-grained temporal constraint is 6400s with a probability consistency state of 81%, the probability of each activity duration is also 81%. According to the normal distribution, 81% means a percentile of $\lambda = 0.87$. Hence, the fine-grained temporal constraints for each activity can be calculated by $\mu + 0.87\sigma$. For example, the fine-grained upper bound temporal constraint for activity X_1 is $(105 + 0.87 * \sqrt{225}) = 118s$ and the upper bound constraint for activity X_{12} is $(123 + 0.87 * \sqrt{64}) = 130s$. The detailed results are presented in Table 3.

Table 3. The setting results



Overall weight joint distribution	$N(\mu_{sw}, \sigma_{sw}^2) = N(6210, 218^2)$	
Coarse-grained upper bound temporal constraint		
$u(SW) = 6420s$ with 81% consistency and $\lambda = 0.87$		
Fine-grained upper bound temporal constraints		
Activity X_1 : 118s	Activity X_2 : 238s	Activity X_3 : 276s
Activity X_4 : 375s	Activity X_5 : 582s	Activity X_6 : 679s
Activity X_7 : 243s	Activity X_8 : 132s	Activity X_9 : 318s
Activity X_{10} : 613s	Activity X_{11} : 681s	Activity X_{12} : 130s

To conclude, the above demonstration of the setting process evidently shows that our probabilistic strategy is effective for setting both coarse-grained and fine-grained temporal constraints. Meanwhile, it has met the two basic requirements proposed in Section 2, i.e. effective negotiation and automatic setting.

6 System Implementation

In this section, we introduce the implementation of the setting strategy in our SwinDew-G scientific grid workflow system.

6.1 SwinDeW-G Scientific Workflow System

SwinDeW-G is a peer-to-peer based scientific grid workflow system running on the SwinGrid (Swinburne service Grid) platform [23]. An overall picture of SwinGrid is depicted in Figure 7 (bottom plane) which contains many grid nodes distributed in different places. Each grid node contains many computers including high performance PCs and/or supercomputers composed of significant number of computing units. The primary hosting nodes include the Swinburne CITR (Centre for Information Technology Research) Node, Swinburne ESR (Enterprise Systems Research laboratory) Node, Swinburne Astrophysics Supercomputer Node, and Beihang CROWN (China R&D environment Over Wide-area Network) Node in China. They are running Linux, GT4 (Globus Toolkit) or CROWN grid toolkit 2.5 where CROWN is an extension of GT4 with more middleware, hence compatible with GT4. Currently, SwinDeW-G is deployed at all primary hosting nodes as exemplified in the top of plane of Figure 7 (top plane). In SwinDeW-G, a scientific workflow is executed by different peers that may be distributed at different grid nodes. As shown in Figure 6, each grid node can have a number of peers, and each peer can be simply viewed as a grid service.

As an important reinforcement for the overall workflow QoS, temporal verification is being implemented in SwinDeW-G. It currently supports dynamic checkpoint selection and temporal verification at run-time [9]. After the running of SwinDeW-G for a period of time, statistical analysis can be applied to accumulated system logs to obtain probability attributes. The probabilistic strategy for setting temporal constraints is being integrated into the scientific workflow modelling tool which supports SPN based modelling, composition of building blocks, temporal data analysis, interactive and automatic setting of temporal constraints.

6.2 SwinDeW-G Scientific Workflow Modelling Tool

Our probabilistic strategy for setting temporal constraints is being implemented into our SwinDeW-G scientific workflow system as an integrated component of the modelling tool. As shown in Figure 8(a), the modelling tool adopts SPN with additional graphic notations, e.g. ⌂ for probability, ▯ for activity duration, ☁ for a sub-process, ◎ for the start point and ⊘ for the end point of an upper bound temporal constraint, to support explicit representation of temporal information. It also supports the composition of the four basic building blocks and user specified ones. The component supports temporal data analysis from workflow system logs. Temporal data

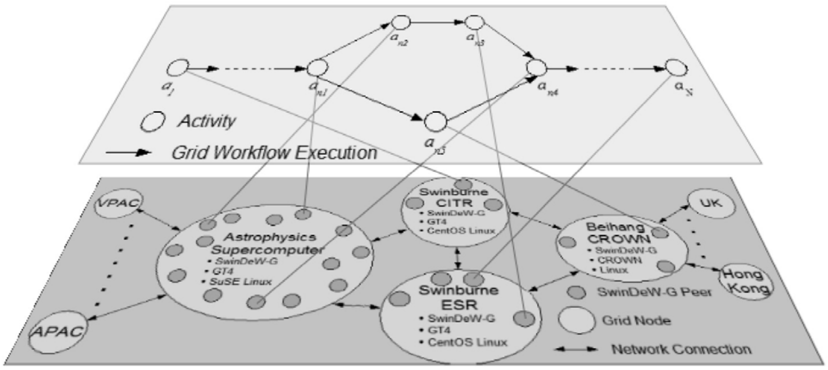
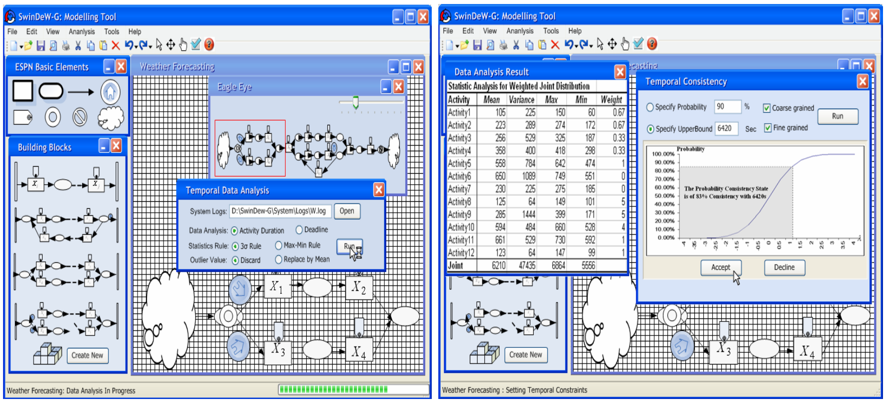


Fig. 7. Overview of SwinDeW-G environment

analysis follows the "3σ"rule and can generate the normal distribution model for each activity duration. The probability attributes for each workflow structure such as the choice probability and iteration times can also be obtained through statistical analysis on historic workflow instances from system logs. After temporal data analysis, the attributes for each activity, i.e. its mean duration, variance, maximum duration, minimum duration and weight are associated to the corresponding activity and explicitly displayed to the client. Meanwhile, the weighted joint distribution of the target process is obtained automatically with basic building blocks. As shown in Figure 8(b), with our probability based temporal consistency, the client can specify an upper bound temporal constraint and the system will reply with a probability for the consistency state. Based on the visualised results shown by a Gaussian curve (the cumulative normal distribution), the client can decide whether to accept or decline the results. If the client is not satisfied with the outcomes, he or she can specify a new value for evaluation until a satisfactory result is achieved. Evidently, the negotiation process between the client and the service provider is implemented as an interactive



Temporal Data Analysis

System Logs: D:\SwinDew-G\System\Logs\W.log

Data Analysis: [✓ Activity Duration] Deadline

Statistics Rule: [✓ 3σ Rule] Max-Min Rule

Outlier Value: [✓ Discard] Replace by Mean

Weather Forecasting: Data Analysis In Progress

Data Analysis Result

Statistic Analysis for Weighted Joint Distribution

Activity	Mean	Variance	Max	Min	Weight
Activity1	105	225	150	60	0.67
Activity2	223	289	274	172	0.67
Activity3	256	529	325	187	0.33
Activity4	358	400	418	298	0.33
Activity5	558	784	642	474	1
Activity6	650	1089	749	551	0
Activity7	230	225	275	185	0
Activity8	125	64	149	101	5
Activity9	285	1444	399	171	5
Activity10	594	484	660	528	4
Activity11	661	529	730	592	1
Activity12	123	64	147	99	1
Joint	6210	47435	6864	5556	

Temporal Consistency

Specify Probability 90 %

[✓ Specify UpperBound] 6420 Sec

[✓ Coarse grained]

[✓ Fine grained]

Weather Forecasting : Setting Temporal Constraints

(a) Temporal data analysis (b) Setting temporal constraints

Fig. 8. SwinDeW-G modelling tool

process between the system user and our developed program. After setting the coarse-grained temporal constraints, the fine-grained constraints for each activity areassigned automatically. These activity duration distribution models, coarse-grained and fine-grained temporal constraints are explicitly represented in the scientific workflow models and will be further deployed to facilitate the effectiveness of run-time temporal verification in scientific workflows.

7 Related Work

In this section, we review some related work on temporal constraints in both traditional workflows and non-traditional workflows. The work in [24] presents the taxonomy of grid workflow QoS constraints which include five dimensions, i.e. time, cost, fidelity, reliability and security. Some papers have presented an overview analysis of scientific or grid workflow QoS [5][14]. The work in [8] presents the taxonomy of grid workflow verification which includes the verification of temporal constraints. Generally speaking, there are two basic ways to assign QoS constraints, one is task-level assignment and the other is workflow-level assignment. Since the whole workflow process is composed by all individual tasks, an overall workflow-level constraint can be obtained by the composition of task-level constraints. On the contrary, task-level constraints can also be assigned by the decomposition of workflow-level constraints [24]. However, different QoS constraints have their own characteristics and require in depth research to handle different scenario.

As shown in our setting strategy, the primary information required for setting temporal constraints include the workflow process models, statistics for activity durations and the definition of temporal consistency. Scientific workflows require the explicit representation of temporal information, i.e. activity durations and temporal constraints to facilitate temporal verification. One of the classical modelling methods is the Stochastic Petri Net (SPN) [1][4] which incorporates time and probability attributes into workflow processes that can be employed to facilitate scientific workflow modelling. Activity duration, as one of the basic elements to measure system performance, is of significant value to workflow scheduling, performance analysis and temporal verification [8][18]. Most work obtains activity durations from workflow system logs and describes them by a discrete or continuous probability distribution through statistical analysis [1]. As for temporal consistency, traditionally, there are only binary states of consistency or inconsistency. However, as stated in [7], it argues that the conventional consistency condition is too restrictive and covers several different states which should be handled differently for the purpose of cost saving. Therefore, it divides conventional inconsistency into weak consistency, weak inconsistency and strong inconsistency and treats them accordingly. However, as we discussed in Section 3, multiple discrete temporal consistency is not quite effective in terms of negotiation and setting for temporal constraints.

Temporal constraints are not well emphasised in traditional workflow systems. However, some business workflow systems accommodate temporal information for the purpose of performance analysis. For example, Staffware provides the audit trail tool to monitor the execution of individual instances [2] and SAP business workflow system employs the workload analysis [22]. As for scientific workflow systems, according to the survey conducted in [24], Askalon [3], GrADS [11], GridBus [12] and

GridFlow [13] support temporal constraints and some other QoS constraints. Yet, to our best knowledge, only SwinDeW-G [23] has set up a series of strategies such as multiple temporal consistency states and efficient checkpoint selection to support dynamic temporal verification [7][9]. In overall terms, even though temporal QoS has been recognised as an important aspect in scientific workflow systems, the work in this area, e.g. the specification of temporal constraints and the support of temporal verification, is still in its infancy.

8 Conclusion and Future Work

In this paper, we have proposed a probabilistic strategy for setting temporal constraints in scientific workflows. The strategy aims to achieve a set of temporal constraints which are well balanced between user requirements and system performance. Hence, novel probability based temporal consistency which is defined by the weighted joint distribution of activity durations has been provided to support an effective negotiation process between the client and the service provider. In addition, the weighted joint distribution of four Stochastic Petri Nets based basic building blocks, i.e. sequence, iteration, parallelism and choice, has been presented to facilitate the setting process. With the probability based temporal consistency, well balanced overall coarse-grained temporal constraints can be achieved through the negotiation process, and afterwards, fine-grained temporal constraints for each activity can be derived instantly in an automatic fashion. A weather forecast scientific workflow has been first employed as a motivating example and then revisited with the detailed setting process to evaluate the effectiveness of our strategy. As an integrated component of the scientific workflow modelling tool in our SwinDeW-G scientific grid workflow system, the probabilistic strategy has been effectively implemented to support the setting of both coarse-grained and fine-grained temporal constraints in scientific workflows.

In the future, with our probability based temporal consistency, we will investigate corresponding run-time strategies for checkpoint selection, temporal verification and temporal-constraint adjustment, so as to improve the overall efficiency and effectiveness of temporal verification in scientific workflows.

Acknowledgments. The research work reported in this paper is partly supported by Australian Research Council under Discovery Grant DP0663841, Linkage Grant LP0669660, by Swinburne Dean's Collaborative Grants Scheme 2007-2008, and by Swinburne Research Development Scheme 2008.

References

1. van der Aalst, W.M.P., Hee, K.M.V., Reijers, H.A.: Analysis of Discrete-Time Stochastic Petri Nets. Statistica Neerlandica 54, 237–255 (2000)
2. van der Aalst, W.M.P., Hee, K.M.V.: Workflow Management: Models, Methods, and Systems. The MIT Press, Cambridge (2002)
3. Askalon Project (accessed March 1, 2008), `http://www.dps.uibk.ac.at/projects/askalon`
4. Bucci, G., Sassoli, L., Vicario, E.: Correctness Verification and Performance Analysis of Real-Time Systems Using Stochastic Preemptive Time Petri Nets. IEEE Trans. on Software Engineering 31(11), 913–927 (2005)

5. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics: Science, Service and Agents on the World Wide Web 1(3), 281–308 (2004)
6. Chen, J., Yang, Y.: Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. ACM Trans. on Autonomous and Adaptive Systems 2(2), Article 6 (June 2007)
7. Chen, J., Yang, Y.: Multiple States based Temporal Consistency for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In: Concurrency and Computation: Practice and Experience, vol. 19, pp. 965–982. Wiley, Chichester (2007)
8. Chen, J., Yang, Y.: A Taxonomy of Grid Workflow Verification and Validation. In: Concurrency and Computation: Practice and Experience, vol. 20, pp. 347–360 (2008)
9. Chen, J., Yang, Y.: Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In: Proc. of 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 2008, pp. 141–150 (2008)
10. Eder, J., Panagos, E., Rabinovich, M.: Time constraints in Workflow Systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 286–300. Springer, Heidelberg (1999)
11. GrADS Project (accessed March 1, 2008), `http://www.hipersoft.rice.edu/grads`
12. GridBus Project (accessed March 1, 2008), `http://www.gridbus.org`
13. GridFlow Project (accessed March 1, 2008), `http://gridflow.ca`
14. Hwang, S.Y., Wang, H., Tang, J., Srivastava, J.: A Probabilistic Approach to Modelling and Estimating the QoS of Web-Service-Based Workflows. Information Sciences 177, 5484–5503 (2007)
15. Kao, B., Garcia-Molina, H.: Deadline Assignment in a Distributed Soft Real-Time System. IEEE Trans. on Parallel and Distributed Systems 8(12), 1268–1274 (1997)
16. Law, A.M., Kelton, W.D.: Simulation Modelling and Analysis, 4th edn. McGraw-Hill, New York (2007)
17. Marjanovic, O., Orlowska, M.E.: On Modelling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems 1(2), 157–192 (1999)
18. Prodan, R., Fahringer, T.: Overhead Analysis of Scientific Workflows in Grid Environments. IEEE Trans. on Parallel and Distributed Systems 19(3), 378–393 (2008)
19. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View, BPM Center Report BPM-06-22 (2006)
20. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and Validation of Process Constraints for Flexible Workflows. Information Systems 30, 349–378 (2005)
21. Stroud, K.A.: Engineering Mathematics, 6th edn. Palgrave Macmillan, New York (2007)
22. Workflow System Administration, SAP Library (accessed March 1, 2008), `http://help.sap.com/saphelp_nw2004s/helpdata/en`
23. Yang, Y., Liu, K., Chen, J., Lignier, J., Jin, H.: Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G. In: Proc. of 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India, December 2007, pp. 51–58 (2007)
24. Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing 3, 171–200 (2005)
25. Yu, J., Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid Computing, Special Issue on Scientific Workflows. ACM SIGMOD Record 34(3), 44–49 (2005)
26. Zhuge, H., Cheung, T., Pung, H.: A Timed Workflow Process Model. Journal of Systems and Software 55(3), 231–243 (2001)

Workflow Simulation for Operational Decision Support Using Design, Historic and State Information

A. Rozinat[1], M.T. Wynn[2], W.M.P. van der Aalst[1,2], A.H.M. ter Hofstede[2], and C.J. Fidge[2]

[1] Information Systems Group, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.rozinat,w.m.p.v.d.aalst}@tue.nl

[2] Business Process Management Group, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{m.wynn,a.terhofstede,c.fidge}@qut.edu.au

Abstract. Simulation is widely used as a tool for analyzing business processes but is mostly focused on examining rather abstract steady-state situations. Such analyses are helpful for the initial design of a business process but are less suitable for operational decision making and continuous improvement. Here we describe a *simulation system for operational decision support* in the context of workflow management. To do this we exploit not only the workflow's *design*, but also logged data describing the system's observed *historic* behavior, and information extracted about the current *state* of the workflow. Making use of actual data capturing the current state and historic information allows our simulations to accurately predict potential near-future behaviors for different scenarios. The approach is supported by a practical toolset which combines and extends the workflow management system YAWL and the process mining framework ProM.

Keywords: Workflow Management, Process Mining, Short-term Simulation.

1 Introduction

Business process simulation is a powerful tool for process analysis and improvement. One of the main challenges is to create simulation models that *accurately* reflect the real-world process of interest. Moreover, we do not want to use simulation just for answering strategic questions but also for tactical and even operational decision making. To achieve this, different sources of simulation-relevant information need to be leveraged. In this paper, we present a new way of creating a simulation model for a business process supported by a workflow management system, in which we integrate design, historic, and state information.

Figure 1 illustrates our approach. We consider the setting of a *workflow system* that supports some *real-world process* based on a *workflow and organizational model*. Note that the workflow and organizational models have been

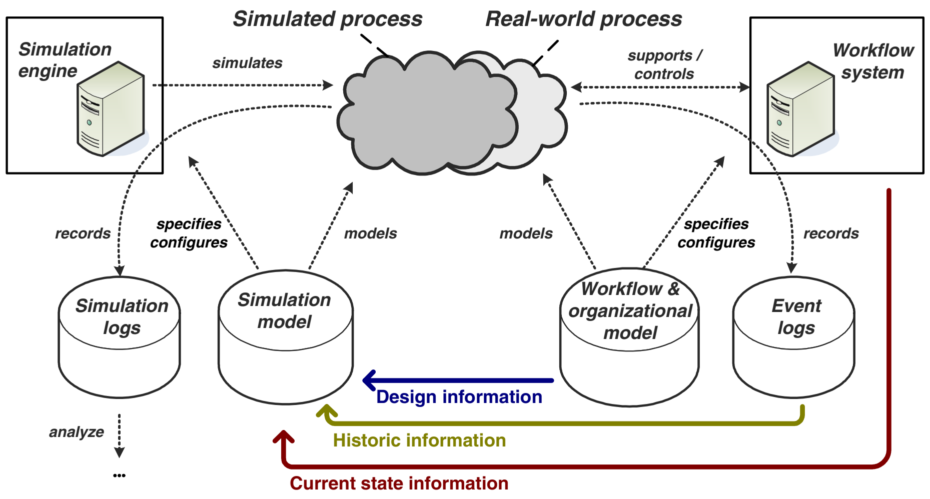


Fig. 1. Overview of our integrated workflow management (right) and simulation (left) system

designed before enactment and are used for the configuration of the workflow system. During the enactment of the process, the performed activities are recorded in *event logs*. An event log records events related to the offering, start, and completion of work items, e.g., an event may be 'Mary completes the approval activity for insurance claim XY160598 at 16.05 on Monday 21-1-2008'.

The right-hand side of Figure 1 is concerned with enactment using a workflow system while the left-hand side focuses on analysis using simulation. In order to link enactment and simulation we propose to use three types of information readily available in workflow systems to create and initialize the simulation model.

- *Design information.* The workflow system has been configured based on an explicit process model describing control and data flows. Moreover, the workflow system uses organizational data, e.g., information about users, roles, groups, etc.
- *Historic information.* The workflow system records all events that take place in 'event logs' from which the complete history of the process can be reconstructed. By analyzing historic data, probability distributions for workflow events and their timing can be extracted.
- *State information.* At any point in time, the workflow process is in a particular state. The current state of each process instance is known and can be used to initialize the simulation model. Note that this current state information includes the control-flow state (i.e., 'tokens' in the process model), case data, and resource data (e.g., resource availability).

By merging the above information into a simulation model, it is possible to construct an *accurate model based on observed behavior* rather than a manually-constructed model which approximates the workflow's anticipated behavior. Moreover, the state information supports a 'fast forward' capability, in which simulation can be used to explore different scenarios with respect to their *effect in the near future*. In this way, simulation can be used for *operational decision making*.

Based on this approach, the system design in Figure 1 allows different simulation experiments to be conducted. For the 'as-is' situation, the simulated and real-world processes should overlap as much as possible, i.e., the two process 'clouds' in Figure 1 coincide. For the 'to-be' situation, the observed differences between the simulated and real-world processes can be explored and quantified. In our implementation we ensure that the simulation logs have the same format as the event logs recorded by the workflow system. In this way we can use the *same tools* to analyze both simulated and real-world processes.

To do this, we need state-of-the art *process mining* techniques to analyze the simulation and event logs and to generate the simulation model. To demonstrate the applicability of our approach, we have implemented the system shown in Figure 1 using ProM [1] and YAWL [2]. YAWL is used as the workflow management system and has been extended to provide high-quality design, historic, and state information. The process mining framework ProM has been extended to merge the three types of information into a single simulation model. Moreover, ProM is also used to analyze and compare the logs in various ways.

The paper is organized as follows. Related work is reviewed in Section 2. Section 3 describes the approach proposed. Section 4 presents a running example, which is then used in Section 5 to explain the implementation realized using YAWL and ProM. Section 6 concludes the paper by discussing the three main innovations presented in this paper.

2 Related Work

Our work combines aspects of workflow management, simulation, and process mining. Some of the most relevant contributions from these broad areas are reviewed below.

Prominent literature on workflow management [6,13,19] focuses on enactment, and research on workflow analysis usually focuses on verification, rather than simulation. Conversely, publications on simulation typically concentrate on statistical aspects [11,16,12] or on a specific simulation language [10]. Several authors have used simulation or queuing techniques to address business process redesign questions [4,5,14], and most mature workflow management systems provide a simulation component [7,8]. However, none of these systems uses historic and state information to learn from the past and to enable operational decision making. We are not aware of any toolset that is able to extract the current state from an operational workflow management system and use this as the starting point for transient analysis.

In earlier work we first introduced the notion of using historic and state information to construct and calibrate simulation models [15,20], and used Protos, ExSpect, and COSA to realize the concept of short-term simulation [15]. However, this research did not produce a practical publicly available implementation and did not use process mining techniques.

Process mining aims at the analysis of event logs [3]. It is typically used to construct a static model that is presented to the user to reflect on the process. Previously we showed that process mining can be used to generate simulation models [17], but design and state information were not used in that work.

3 Approach

A crucial element of the approach in Figure 1 is that the *design*, *historic* and *state* information provided by the workflow system are used as the basis for simulation. Table 1 describes this information in more detail.

Table 1. Process characteristics and the data sources from which they are obtained

Design information *(obtained from the workflow and organization model used to configure the workflow system)*	**Historic information** *(extracted from event logs containing information on the actual execution of cases)*	**State information** *(based on information about cases currently being enacted using the workflow system)*
• control and data flow (activities and causalities)	• data value range distributions	• progress state of cases (state markers)
• organizational model (roles, resources, etc.)	• execution time distributions	• data values for running cases
• initial data values	• case arrival rate	• busy resources
• roles per task	• availability patterns of resources	• run times for cases

The design information is static, i.e., this is the specification of the process and supporting organization that is provided at design time. This information is used to create the structure of the simulation model. The historic and state information are dynamic, i.e., each event adds to the history of the process and changes the current state. Historic information is aggregated and is used to set parameters in the simulation model. For instance, the arrival rate and processing times are derived by aggregating historic data, e.g., the (weighted) average over the last 100 cases is used to fit a probability distribution. Typically, these simulation parameters are not very sensitive to individual changes. For example, the average processing time typically changes only gradually over a long period. The current state, however, is highly sensitive to change. Individual events directly influence the current state and must be directly incorporated into the initial state of the simulation. Therefore, design information can be treated as static, while historic information evolves gradually, and state information is highly dynamic.

To realize the approach illustrated in Figure 1 we need to merge design, historic and state information into a single simulation model. The design information is used to construct the structure of the simulation model. The historic information is used to set parameters of the model (e.g., fit distributions). The state information is used to initialize the simulation model. Following this, traditional simulation techniques can be used. For example, using a random generator and replication, an arbitrary number of independent simulation experiments can be conducted. Then statistical methods can be employed to estimate different performance indicators and compute confidence intervals for these estimates.

By modifying the simulation model, various 'what-if' scenarios can be investigated. For example, one can add or remove resources, skip activities, etc. and see what the effect is. Because the simulation experiments for these scenarios start from the current state of the actual system, they provide a kind of 'fast-forward button' showing what will happen in the near future, to support operational decision making. For instance, based on the predicted system behavior, a manager may decide to hire more personnel or stop accepting new cases.

Importantly, the simulations yield simulation logs in the same format as the event logs. This allows process mining techniques to be used to view the real-world processes and the simulated processes in a unified way. Moreover, both can be compared to highlight deviations, etc.

4 Running Example

Consider the credit card application process expressed as a YAWL workflow model in Figure 2. The process starts when an applicant submits an application. Upon receiving an application, a credit clerk checks whether it is complete. If not, the clerk requests additional information and waits until this information is received before proceeding. For a complete application, the clerk performs further checks to validate the applicant's income and credit history. Different checks are performed depending on whether the requested loan is large (e.g. greater than $500) or small. The validated application is then passed on to a manager to decide whether to accept or reject the application. In the case of acceptance, the applicant is notified of the decision and a credit card is produced and delivered to the applicant. For a rejected application, the applicant is notified of the decision and the process ends.

Here we assume that this example workflow has been running for a while. In YAWL but also any other workflow system the following runtime statistics can be gathered about the long-term behavior of this process.

– Case arrival rate: 100 applications per week
– Throughput time: 4 working days on average

With respect to resources, there are eight members of staff available, which include three capable of acting as 'managers' and seven capable of acting as 'clerks'. (One person can have more than one role.)

Further assume that due to a successful Christmas promotion advertised in November, the number of credit card applications per week has temporarily

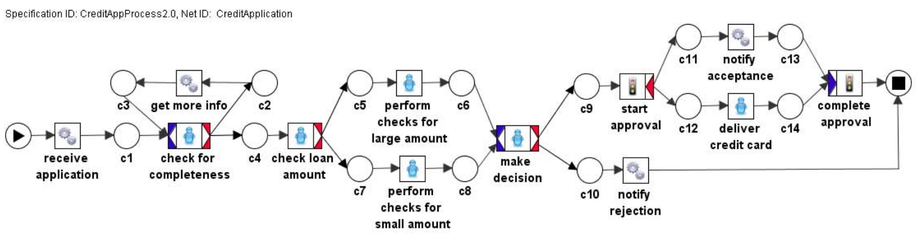


Fig. 2. A credit application process modeled in YAWL

doubled to 200. The promotion period is now over and we expect the rate to decrease to 100 applications per week again. However, as a result of the increased interest, the system now has a backlog of 150 applications in various stages of processing, some of which have been in the system for more than a week. Since it is essential that most applications are processed before the holiday season, which begins in a fortnight from now (the 'time horizon' of interest), management would like to perform simulation experiments from the current state ('fast forward') to determine whether or not the backlog can be cleared in time.

5 Realization through YAWL and ProM

We now use the example introduced in Section 4 to describe our proof-of-concept implementation supporting the approach depicted in Figure 1. The realization is based on the YAWL workflow environment [2] and the process mining framework ProM [1]. We focus on the new capabilities that have been added to these systems, and briefly explain the main steps that need to be performed[1].

5.1 Extracting Simulation-Relevant Information

The information contained in the workflow specification is supplemented with historical data obtained from the event logs and data from the organizational model database. This was achieved by implementing two new functions in the workflow engine to export historical data from the logs for a particular specification and to export the organizational model (i.e., information about roles and resources).

In the YAWL workflow system, event logs are created whenever an activity is enabled, started, completed or cancelled, together with the time when this event occurred and with the actor who was involved. Logs are also kept for data values that have been entered and used throughout the system. Therefore, we can retrieve historical data about process instances that have finished execution. In this work we assume that the simulation experiments are being carried out on 'as-is' process models for which historical data is available. A function has been

[1] A detailed description of how to generate a simulation model including operational decision support is provided in our technical report [18]. The example files and the ProM framework can be downloaded from `http://www.processmining.org`.

```
<Process>
    <ProcessInstance id="5">
        <AuditTrailEntry>
            <Data>
                <Attribute name="loanAmt">550</Attribute>
            </Data>
            <WorkflowModelElement>
                    receive_application_3
            </WorkflowModelElement>
            <EventType>complete</EventType>
            <Timestamp>
                    2008-02-29T15:20:01.050+01:00
            </Timestamp>
            <Originator>MoeW</Originator>
        </AuditTrailEntry>
            ...
    </ProcessInstance>
            ...
</Process>
```

(a) A log entry for the completion of activity 'receive application' carried out by resource MoeW with loan amount $550

```
<OrgModel>
    <OrgEntity>
            <EntityID>1</EntityID>
            <EntityName>manager</EntityName>
            <EntityType>Role</EntityType>
    </OrgEntity>
    <OrgEntity>
            <EntityID>2</EntityID>
            <EntityName>clerk</EntityName>
            <EntityType>Role</EntityType>
    </OrgEntity>
            ...
    <Resource>
            <ResourceID>PA-529f00b8-0339</ResourceID>
            <ResourceName>JonesA</ResourceName>
            <HasEntity>2</HasEntity>
    </Resource>
            ...
</OrgModel>
```

(b) An excerpt from an organizational model with roles and resources, where resource JonesA has role 'clerk'

Fig. 3. Part of an organizational model and historical data extracted from the workflow engine

created which extracts the historical data for a specification from the workflow engine and exports audit trail entries in the *M*ining *XML* (MXML) log format. Some sample data for the credit application example is shown in Figure 3(a). This historical data is used for mining information about case arrival rates and distribution functions for the data values used in future simulation experiments.

Similarly, the YAWL workflow system gives access to the organizational model through a function which extracts all available role and resource data in an organization and exports this information in the XML format required by ProM. Some sample data with the roles of clerk and manager are shown in Figure 3(b). This information is used to identify available roles and resources that are relevant for a given specification.

5.2 Generating the Simulation Model

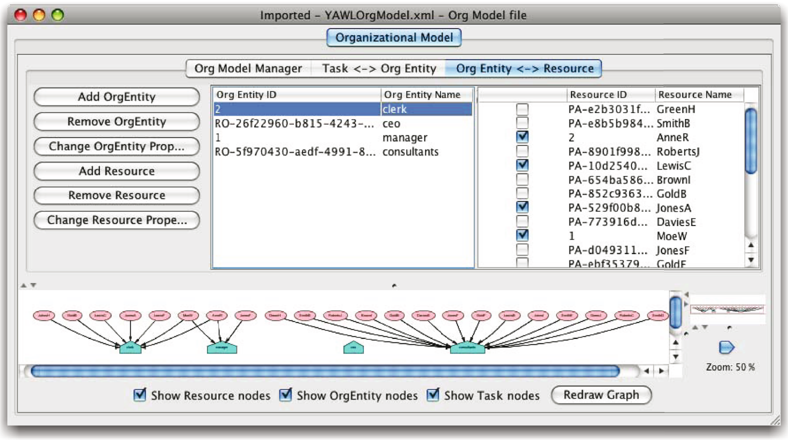
From (1) the extracted workflow specification, (2) the newly extracted organizational model, and (3) the event log file, we can now generate a simulation model that reflects the process as it is currently enacted. The direct usage of design information avoids mistakes that are likely to be introduced when models are constructed manually, and the automated extraction of data from event logs allows the calibration of the model based on actually observed parameters.

To generate the model, four basic steps need to be performed within ProM (a sample screenshot is shown for each phase in Figures 4 and 5):

1. The YAWL model, the organizational model, and the event log need to be imported from YAWL and analyzed.
2. Simulation-relevant information from the organizational model and log analysis needs to be integrated into the YAWL model.

3. The integrated YAWL model must be converted into a Petri net model (because our simulation tool is based on Coloured Petri Nets).
4. Finally, the integrated and converted model can be exported as a Coloured Petri Net (CPN) model for simulation.

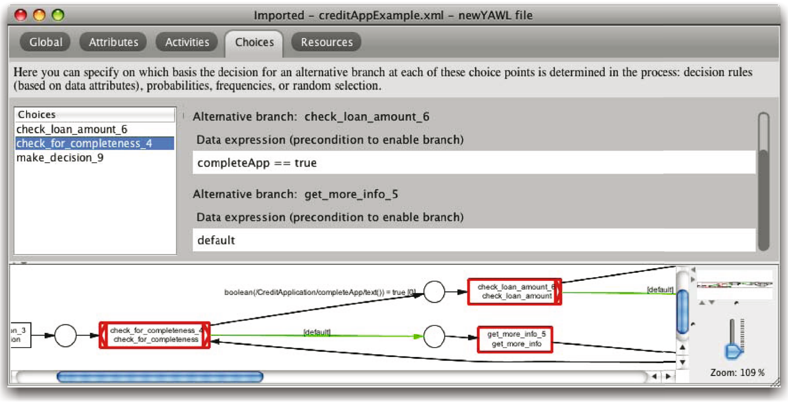
We can then use the CPN Tools system [9] to simulate the generated model. However, to produce useful results we do not want to start from an empty initial state. Instead we load the current state of the actual YAWL system into the CPN Tools for simulation.



Org Entity ID	Org Entity Name
2	clerk
RO-26f22960-b815-4243-...	ceo
1	manager
RO-5f970430-aedf-4991-8...	consultants

	Resource ID	Resource Name
☐	PA-e2b3031f...	GreenH
☐	PA-e8b5b984...	SmithB
☑	2	AnneR
☐	PA-8901f998...	RobertsJ
☑	PA-10d2540...	LewisC
☐	PA-654ba586...	BrownI
☐	PA-852c9363...	GoldB
☑	PA-529f00b8...	JonesA
☐	PA-773916d...	DaviesE
☑	1	MoeW
☐	PA-d049311...	JonesF
☐	PA-ebf35379...	GoldF

(a) Data is imported from different sources. Here the organizational model import is shown



Here you can specify on which basis the decision for an alternative branch at each of these choice points is determined in the process: decision rules (based on data attributes), probabilities, frequencies, or random selection.

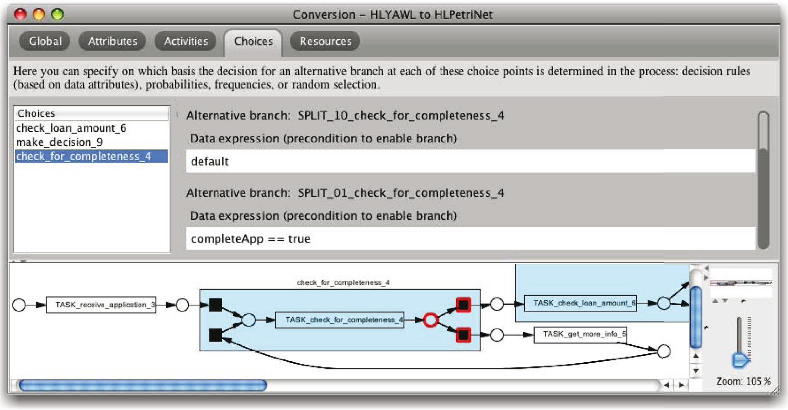
Choices
check_loan_amount_6
check_for_completeness_4
make_decision_9

Alternative branch: check_loan_amount_6
Data expression (precondition to enable branch)
completeApp == true

Alternative branch: get_more_info_5
Data expression (precondition to enable branch)
default

(b) The organizational model and the information obtained from the log analysis are integrated into the imported YAWL model

Fig. 4. *Phase 1*: The workflow and organizational model are imported and integrated with the information obtained from event log analysis



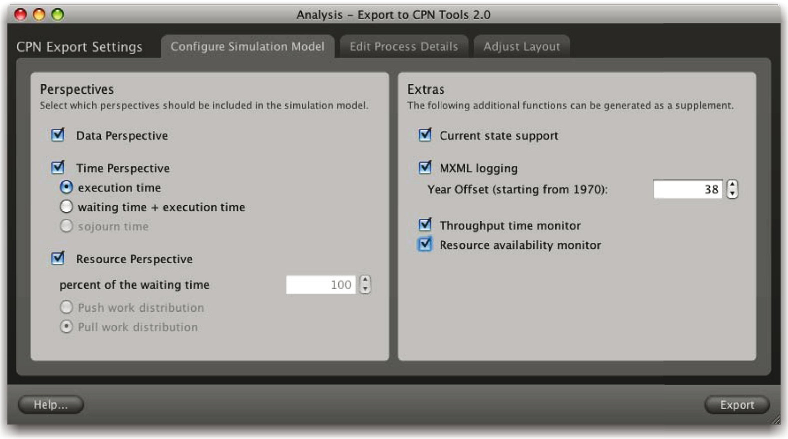
Here you can specify on which basis the decision for an alternative branch at each of these choice points is determined in the process: decision rules (based on data attributes), probabilities, frequencies, or random selection.

Choices
check_loan_amount_6
make_decision_9
check_for_completeness_4

Alternative branch: SPLIT_10_check_for_completeness_4
Data expression (precondition to enable branch)
default

Alternative branch: SPLIT_01_check_for_completeness_4
Data expression (precondition to enable branch)
completeApp == true

(a) The integrated YAWL model is translated into a Petri net while preserving all the simulation-relevant information



Perspectives
Select which perspectives should be included in the simulation model.

[✓] Data Perspective

[✓] Time Perspective
[✓ execution time]
waiting time + execution time
sojourn time

[✓] Resource Perspective
percent of the waiting time: 100
Push work distribution
[✓ Pull work distribution]

Extras
The following additional functions can be generated as a supplement.

[✓] Current state support

[✓] MXML logging
Year Offset (starting from 1970): 38

[✓] Throughput time monitor
[✓] Resource availability monitor

(b) After importing, merging, and converting the data, a simulation model including current state support can be generated

Fig. 5. *Phase 2*: To enable the export to CPN Tools, the YAWL model is first converted into a Petri net. Then, a CPN model of the process is generated.

5.3 Loading the Current State

To carry out simulation experiments for operational decision making purposes (the 'fast forward' approach), it is essential to include the current state of the workflow system. This allows us to make use of the data values for the current cases as well as the status of the work items for current cases within the simulation experiments. A new function has been created to extract current state information of a running workflow from the YAWL system and to export this information as a CPN Tools input file (see Figure 6).

```
fun getInitialCaseData() = [(41, {loanAmt = 1500,completeApp = false,decideApp = false}),
            (40, {loanAmt = 0,completeApp = false,decideApp = false}),
            (39, {loanAmt = 500,completeApp = false,decideApp = false})];
fun getNextCaseID() = 42;
fun getInitialTokensExePlace(pname:STRING) = case pname of
            "TASK_check_for_completeness_4`E"=>[(41,"-154","JonesA")] | _ => empty;
fun getInitialTokens(pname:STRING) = case pname of
            "Process`COND_c2_15"=>[(39,"-43200")] | "Overview`Start"=>[(40,"-155")] | _ => empty;
fun getBusyResources() = ["JonesA"];
fun getCurrentTimeStamp() = "1205203218";
fun getTimeUnit() = "Sec";
```

Fig. 6. CPN Tools input file with initial state information. Several cases are in different states in the system. For example, application No. 41 is currently being checked by JonesA for completeness, and has a run time of 154 secs, i.e., ca. 2.57 mins.

The following information is obtained about the current state and is introduced as the initial state of a simulation run.

- All the running cases of a given workflow and their marking.
- All the data values associated with each case.
- Information about enabled work items.
- Information about executing work items and the resources used.
- The date and time at which the current state file is generated.

When the empty initial state file of the generated simulation model is replaced with the file depicted in Figure 6, tokens are created in the CPN model that reflect the current system status (see Figure 7). For example, among the three *Case data* tokens is the data associated with application No. 41. The resource JonesA is currently performing a check activity on this case and hence, it does not appear in the list of free resources.

We now follow the scenario described in Section 4 for simulation experiments, i.e., due to a promotion 150 cases are in the system. We load the state file containing these 150 cases into the model and perform simulation experiments for the coming two weeks. We also add more resources to the model and observe how this influences the backlog and the throughput times for processing credit card applications within this time horizon.

5.4 Analyzing the Simulation Logs

We simulate the process from the generated CPN model for four different scenarios:

1. An empty initial state. ('empty' in Figure 8)
2. After loading the current state file with the 150 applications that are currently in the system and no modifications to the model, i.e., the 'as-is' situation. ('as is' in Figure 8)
3. After loading the current state file but adding four extra resources (two having the role 'manager' and three having the role 'clerk'), i.e., a possible 'to-be' situation to help clear the backlog more quickly. ('to be A' in Figure 8)

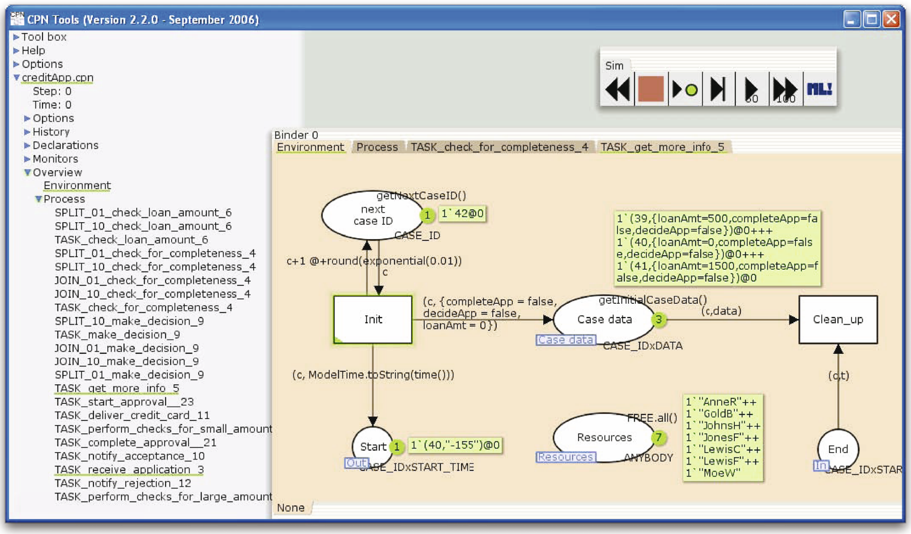

CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
creditApp.cpn
Step: 0
Time: 0
Options
History
Declarations
Monitors
Overview
Environment
Process
SPLIT_01_check_loan_amount_6
SPLIT_10_check_loan_amount_6
TASK_check_loan_amount_6
SPLIT_01_check_for_completeness_4
SPLIT_10_check_for_completeness_4
JOIN_01_check_for_completeness_4
JOIN_10_check_for_completeness_4
TASK_check_for_completeness_4
SPLIT_10_make_decision_9
TASK_make_decision_9
JOIN_01_make_decision_9
JOIN_10_make_decision_9
SPLIT_01_make_decision_9
TASK_get_more_info_5
TASK_start_approval__23
TASK_deliver_credit_card_11
TASK_perform_checks_for_small_amount
TASK_complete_approval__21
TASK_notify_acceptance_10
TASK_receive_application_3
TASK_notify_rejection_12
TASK_perform_checks_for_large_amount
Sim
Binder 0
Environment
Process
TASK_check_for_completeness_4
TASK_get_more_info_5
getNextCaseID()
next case ID
1
1`42@0
CASE_ID
c+1 @+round(exponential(0.01))
c
1`(39,{loanAmt=500,completeApp=false,decideApp=false})@0+++
1`(40,{loanAmt=0,completeApp=false,decideApp=false})@0+++
1`(41,{loanAmt=1500,completeApp=false,decideApp=false})@0
Init
(c, {completeApp = false, decideApp = false, loanAmt = 0})
getInitialCaseData()
Case data
3
(c,data)
Clean_up
Case data
CASE_IDxDATA
(c, ModelTime.toString(time()))
(c,t)
Start
1
1`(40,"-155")@0
Out
CASE_IDxSTART_TIME
FREE.all()
Resources
7
Resources
ANYBODY
1`"AnneR"++
1`"GoldB"++
1`"JohnsH"++
1`"JonesF"++
1`"LewisC"++
1`"LewisF"++
1`"MoeW"
End
In
CASE_IDxSTAR
None

Fig. 7. The generated CPN model after loading the current state file

4. After loading the current state file and adding eight extra resources (four having the role 'manager' and six having the role 'clerk'). ('to be B' in Figure 8)

We can see the difference among these four scenarios in Figure 8, which depicts the development of the number of cases (i.e., applications) in the workflow system over the coming two weeks for an example simulation run per scenario. In the case of Scenario 1 the simulation starts with having 0 credit card applications in the system. This does neither reflect the normal situation nor does it capture our current backlog of cases. Only after a while, does this simulation represent the normal behavior of the credit card application process (i.e., with ca. 100 applications arriving per week). The other three scenarios load a defined initial state, which contains the 150 applications that we assume to be currently in the system. Furthermore, one can observe that in the scenarios where we add extra resources to the process, the case load decreases more quickly to a normal level than without further intervention. However, the scenario 'to be B' does not seem to perform much better than the scenario 'to be A' although twice as many resources have been added. This way, we can assess the effect of possible measures to address the problem at hand, i.e., we can compare different 'what-if' scenarios in terms of their estimated real effects.

CPN Tools has powerful simulation capabilities, which we can leverage. For example, it is possible to automatically replicate simulation experiments to enable statistical analyses, such as calculating confidence intervals for specific process characteristics. For instance, Figure 9 depicts the 95% confidence intervals of the average case throughput times based on 50 replicated simulations for each of the four simulation scenarios. One can observe that the estimated throughput time

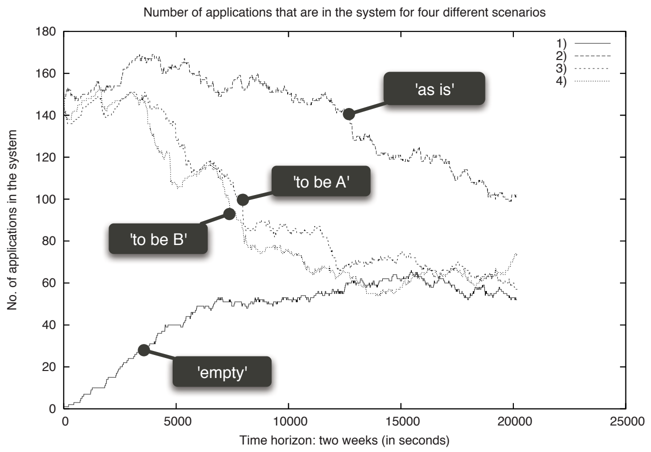


Fig. 8. Number of applications in the simulated process for the different scenarios. While the scenario with the empty state has initially 0 applications, the other scenarios are initialized by loading 150 applications from the current state file.

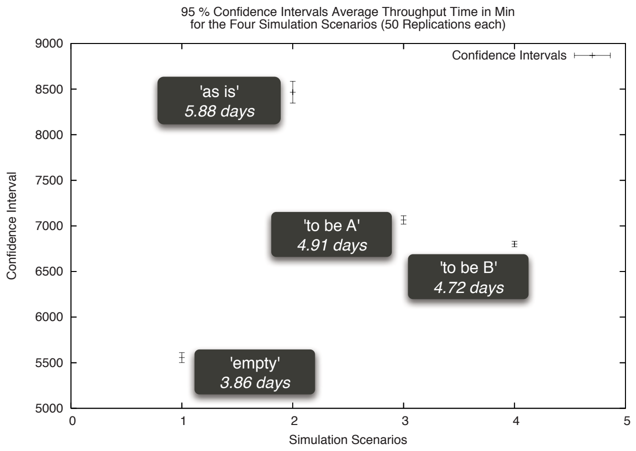
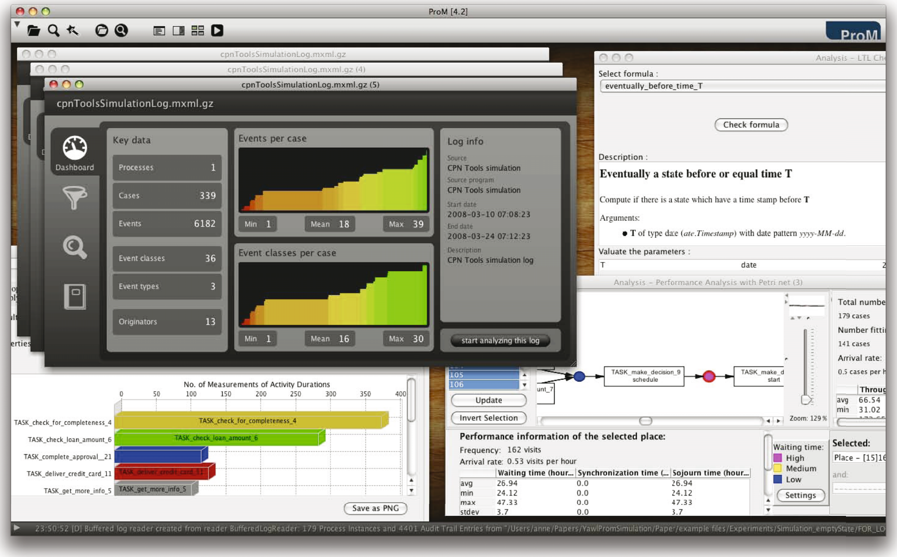


Fig. 9. Simulation run showing the 95% confidence intervals of the throughput times for the different simulation scenarios. The length of the confidence interval indicates the degree of variation.

for the 'empty' scenario (i.e., based on the usual situation) is ca. 4 days, while the expected throughput time for the 'as is' scenario (i.e., actually expected based on the current backlog situation) is almost 6 days.

While CPN Tools already provides powerful logging facilities and even generates gnuplot scripts that can be used to plot certain properties of the simulated process, we also generate MXML event log fragments during simulation, similar to the one shown in Figure 3(a) for the workflow log. These fragments can then


ProM [4.2]
cpnToolsSimulationLog.mxml.gz
cpnToolsSimulationLog.mxml.gz (4)
cpnToolsSimulationLog.mxml.gz (5)
Dashboard
Key data
Events per case
Event classes per case
Log info
start analyzing this log
No. of Measurements of Activity Durations
Save as PNG
Analysis – LTL Checker
Check formula
Analysis – Performance Analysis with Petri net (3)
TASK_make_decision_9 schedule
TASK_make_decision_9 start
Update
Invert Selection
Zoom: 129 %
Settings

cpnToolsSimulationLog.mxml.gz

Key data

Processes	1
Cases	339
Events	6182
Event classes	36
Event types	3
Originators	13

Events per case: Min 1, Mean 18, Max 39

Event classes per case: Min 1, Mean 16, Max 30

Log info
Source: CPN Tools simulation
Source program: CPN Tools simulation
Start date: 2008-03-10 07:08:23
End date: 2008-03-24 07:12:23
Description: CPN Tools simulation log

Select formula : eventually_before_time_T

Description :
Eventually a state before or equal time T

Compute if there is a state which have a time stamp before **T**

Arguments:

- **T** of type date (*ate.Timestamp*) with date pattern *yyyy-MM-dd*.

Valuate the parameters :
T date

Total number: 179 cases
Number fitting: 141 cases
Arrival rate: 0.5 cases per h

	Throug
avg	66.54
min	31.02

Performance information of the selected place:
Frequency: 162 visits
Arrival rate: 0.53 visits per hour

	Waiting time (hour...	Synchronization time (...	Sojourn time (hour...
avg	26.94	0.0	26.94
min	24.12	0.0	24.12
max	47.33	0.0	47.33
stdev	3.7	0.0	3.7

Waiting time: 🟪 High 🟨 Medium 🟦 Low

Selected: Place – [15]16

```
23:50:52 [D] Buffered log reader created from reader BufferedLogReader: 179 Process Instances and 4401 Audit Trail Entries from "/Users/anne/Papers/YawlPromSimulation/Paper/example files/Experiments/Simulation_emptyState/FOR_LO
```

Fig. 10. The generated simulation logs can be analyzed with the same tool set as the initial workflow logs

be combined using the CPN Tools filter of the ProM*import* framework, which facilitates the conversion of event logs from various systems into the MXML format that is read by ProM.

The ability to use the same toolset for analyzing the simulation logs and analyzing the actual workflow logs constitutes a big advantage because the simulation analysis results can be more easily related to the initial properties of the process. In particular, since we support the loading of current cases into the initial state at the beginning of the simulation, *we can easily combine the real process execution log ('up to now') and the simulation log (which simulates the future 'from now on') and look at the process in a unified manner* (with the possibility of tracking both the history and the future of particular cases that are in the system at this point in time).

Figure 10 shows a screenshot of ProM while analyzing the simulation logs generated by CPN Tools. Various plug-ins can be used to gain more insight into the simulated process. For example, in Figure 10 the Log Dashboard (top left), the Basic Statistics plug-in (bottom left), the Performance Analysis plug-in (bottom right), and the LTL Checker (top right) are shown. The former two provide a general overview about the cases and activities in the process, whereas the Performance Analysis plug-in finds bottlenecks (e.g., in Figure 10 a bottleneck for starting the activity 'Make decision' is highlighted), and the LTL Checker can be used to verify specific properties of interest (e.g., "How many cases could be processed until they are in the stage where a decision can be made in under 3 days?").

6 Discussion

In this paper we presented an innovative way to link workflow systems, simulation, and process mining. By combining these ingredients it becomes possible to analyze and improve business processes in a consistent way. The approach is feasible, as demonstrated by our implementation using YAWL and ProM. To conclude, we would like to discuss the three main challenges that have been addressed in this research.

6.1 Faithful Simulation Models

Although the principle of simulation is easy to grasp, it takes time and expertise to build a good simulation model. In practice, simulation models are often flawed because of incorrect input data and a naïve representation of reality. In most simulation models it is assumed that resources are completely dedicated to the simulated processes and are eager to start working on newly arriving cases. In reality this is not the case and as a result the simulation model fails to capture the behavior of resources accurately. Moreover, in manually constructed models steps in the processes are often forgotten. Hence simulation models are usually too optimistic and describe a behavior quite different from reality. To compensate for this, artificial delays are added to the model to calibrate it and as a result its predictive value and trustworthiness are limited. In the context of workflow systems, this can be partly circumvented by using the workflow design (the process as it is enforced by the system) and historic data. *The approach presented in this paper allows for a direct coupling of the real process and the simulation model.* However, the generated CPN models in this paper can be improved by a better modeling of resource behavior. Moreover, the process mining techniques that extract characteristic properties of resources need to be improved to create truly faithful simulation models.

6.2 Short-Term Simulation

Although most workflow management systems offer a simulation component, simulation is rarely used for operational decision making and process improvement. One of the reasons is the inability of traditional tools to capture the real process (see above). However, another, perhaps more important, reason is that existing simulation tools aim at strategic decisions. Existing simulation models start in an arbitrary initial state (without any cases in the pipeline) and then simulate the process for a long period to make statements about the steady-state behavior. However, this steady-state behavior does not exist (the environment of the process changes continuously) and is thus considered irrelevant by the manager. Moreover, the really interesting questions are related to the near future. Therefore, *the 'fast-forward button' provided by short-term simulation is a more useful option.* Because of the use of the current state and historic data, the predictions are more valuable, i.e., of higher quality and easier to interpret and apply. The approach and toolset presented in this paper allow for short-term simulation. In the current implementation the coupling between YAWL and ProM is not well-integrated, e.g., the translation of insights from simulation to concrete actions in the workflow system can be improved. Further research is needed to provide a seamless, but generic, integration.

6.3 Viewing Real and Simulated Processes in a Unified Manner

Both simulation tools and management information systems (e.g., BI tools) present information about processes. It is remarkable that, although both are typically used to analyze the same process, the results are presented in completely different ways using completely different tools. This may be explained by the fact that for a simulated process different data is available than for the real-world process. However, *the emergence of process mining techniques allows for a unification of both views.* Process mining can be used to extract much more detailed and dynamic data from processes than traditional data warehousing and business intelligence tools. Moreover, it is easy to extend simulation tools with the ability to record event data similar to the real-life process. Hence, process mining can be used to view both simulated and real processes. As a result, it is easier to both compare and to interpret 'what-if' scenarios.

Acknowledgements. This research was supported by the IOP program of the Dutch Ministry of Economic Affairs and by Australian Research Council grant DP0773012. The authors would like to especially thank Michael Adams, Eric Verbeek, Ronny Mans, and also Christian Günther, Minseok Song, Lindsay Bradford, and Chun Ouyang for their valuable support in implementing the approach for YAWL and ProM. We also would like to thank Marlon Dumas for sharing his valuable insights during the many discussions we had about this topic.

References

1. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., Alves de Medeiros, A.K., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM 4.0: Comprehensive Support for Real Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
3. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business Process Mining: An Industrial Application. Information Systems 32(5), 713–732 (2007)
4. Ardhaldjian, R., Fahner, M.: Using simulation in the business process reengineering effort. Industrial engineering, pp. 60–61 (July 1994)
5. Buzacott, J.A.: Commonalities in Reengineered Business Processes: Models and Issues. Management Science 42(5), 768–782 (1996)
6. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons, Chichester (2005)

7. Hall, C., Harmon, P.: A Detailed Analysis of Enterprise Architecture, Process Modeling, and Simulation Tools. Technical report 2.0, BPTrends (September 2006)
8. Jansen-Vullers, M., Netjes, M.: Business process simulation – a tool survey. In: Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark (October 2006)
9. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer 9(3-4), 213–254 (2007)
10. Kelton, D.W., Sadowski, R., Sturrock, D.: Simulation with Arena. McGraw-Hill, New York (2003)
11. Kleijnen, J., van Groenendaal, W.: Simulation: a statistical perspective. John Wiley and Sons, New York (1992)
12. Laugna, M., Marklund, J.: Business Process Modeling, Simulation, and Design. Prentice Hall, Upper Saddle River, New Jersey (2005)
13. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River (1999)
14. Reijers, H.: Design and Control of Workflow Processes. LNCS, vol. 2617. Springer, Berlin (2003)
15. Reijers, H.A., van der Aalst, W.M.P.: Short-Term Simulation: Bridging the Gap between Operational Control and Strategic Decision Making. In: Hamza, M.H. (ed.) Proceedings of the IASTED International Conference on Modelling and Simulation, pp. 417–421. IASTED/Acta Press, Anaheim (1999)
16. Ross, S.M.: A course in simulation. Macmillan, New York (1990)
17. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering Colored Petri Nets From Event Logs. International Journal on Software Tools for Technology Transfer 10(1), 57–74 (2008)
18. Rozinat, A., Wynn, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.: Workflow Simulation for Operational Decision Support using YAWL and ProM. BPM Center Report BPM-08-04, BPMcenter.org (2008)
19. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
20. Wynn, M.T., Dumas, M., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Business Process Simulation for Operational Decision Support. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM 2007. LNCS, vol. 4928, pp. 66–77. Springer, Heidelberg (2008)

Analyzing Business Continuity through a Multi-layers Model

Yudistira Asnar and Paolo Giorgini

Department of Information and Communication Technology,
University of Trento, Italy
{yudis.asnar,paolo.giorgini}@disi.unitn.it

Abstract. Business Continuity Management (BCM) is a process to manage risks, emergencies, and recovery plans of an organization during a crisis. It results in a document called Business Continuity Plans (BCP) that specifies the methodology and procedures required to backup and recover the functional unit of a disrupted business. Traditionally, the BCP assessment is based only on the continuity of IS infrastructures and does not consider possible relations with the business objectives and business processes. This traditional approach assumes that the risk of business continuity is resulted from the disruption of the IS infrastructures. However, we believe there are situations where the risk emerges even the infrastructures up and running. Moreover, the lack of modeling framework and the aided-tool make the process even harder.

In this paper, we propose a framework to support modeling and analysis of BCP from the organization perspective, where risks and treatments are modeled and analyzed along strategic objectives and their realizations. An automated reasoner based on cost-benefit analysis techniques is proposed to elicit and then adopt the most cost-efficient plan. The approach is developed using the Tropos Goal-Risk Framework and the Time Dependency and Recovery Model as underlain frameworks. A Loan Originating Process case study is used as a running example to illustrate the proposal.

1 Introduction

Information Systems (IS) are currently evolving in so called socio-technical systems, where human and organization factors along technical aspects assume a more and more critical role in the correct operation of the system. A socio-technical system is represented as a complex network of interrelationships between human and technical systems that includes hardware, software, users, stakeholders, data, and regulations [1]. As reported in [2], economic and social factors results being crucial in such systems and introduce challenges that lay beyond the mere technical aspects.

In sectors such as e-Banking, e-Commerce, etc., where the business strongly depends on the availability of IS's services, an organization should be able to ensure the continuity of its business objectives accordingly to the evolution of regulations (e.g., Basel II [3] or Sarbanes-Oxley Act [4]) as well as customers' needs. Business Continuity Management (BCM) is a process aiming at managing risks, emergencies, and recovery plans of an organization during a crisis and ensuring the returning to the normal business operations [5]. A Business Continuity Plan (BCP) [6] specifies the methodologies and procedures required to backup and recover every functional units of the business.

Traditionally, BCP focuses mainly on the analysis of IT infrastructures and does not consider other aspects of the business such as business-process and business-objective [7,8]. For instance, in a e-Shopping scenario, where the main business-objective is *selling items to customers*, the continuity of business-objective might depend not only from the IT infrastructures (e.g., inventory servers, firewall, payment servers, and authentication servers), but also from the operational-level of the organization, such as *delayed payment services* or even more higher level, such as *existence of new competitors*.

In this paper, we propose a framework to support the analysis of business continuity from a socio-technical perspective. Essentially, we extend our previous work on risk analysis [9] with the light of the Time Dependency and Recovery (TDR) model [8]. Our previous framework is extended in order to analyze the business-objectives, to realize them at more operational level (business process [10] or tasks) and, finally, to identify the required artifacts to execute the processes. To model dependencies among assets (objectives, processes, artifacts), we adopt the time-dependency relation from the TDR model. This proposed framework intends to assists analysts in: 1) analyzing assets, 2) defining additional measures to fulfill the stakeholders' target, and 3) defining the most cost-effective mitigation plans.

The remaining paper is organized as follows. Next we present a running example, the *Loan Originating Process (LOP)* of a bank (§2). We then introduce the modeling framework (§3) that extends our previous Goal-Risk framework with the TDR model, and the analysis processes supported by the framework itself (§4). Then, we apply the framework to the LOP case study to evaluate our proposal (§5), and, finally, we discuss related works (§6) and conclude the paper (§7).

2 Running Example

The case study that we use in this paper is originated within the European project SERENITY.[1] It focuses on a typical *Loan Origination Process* (LOP) that starts by receiving a loan application and ends, possibly, with the loan approval. Essentially, a Loan Department within a bank is responsible to accept loan applications, handle the applications, and ensure the loan repayment. These objectives are operationalized through a set of business processes. For instance, once the bank receives a loan application, it starts the handling process verifying the data and calculating the credit score. The score is assessed either internally (in-house assessment) or by an external party (Credit Bureau). Afterward, the bank defines the loan schema, namely defining the loan cap and its interest. In this example, we assume it is always the case that the customer agrees with the loan schema proposed by the bank. Surely, the bank is also interested in ensuring the repayment of the loan.

Uncertain events (i.e., threats, un/intentional events, incidents, risks) may affect the availability of assets. For instance, events like computer virus outbreak, database failure, the outage of national identity service are considered as disruptions for the

[1] http://www.serenity-project.org/

loan department. Essentially, these disruptions are hard, or impossible, to avoid, but they might be still acceptable if their effects vanish after an acceptable period (called Maximum Tolerable Period of Disruption-MTPD). For an example, the goal of receiving loan is still satisfied though it is disrupted for 2 hours. To maintain the MTPD, all responsible stakeholders establish a contingency plan in case their assets are disrupted. The plan, typically, consists of the Recovery Time Objectives (RTOs) that represent the recovery time of assets. For instance, the IT department ensures that the database of loan application system will be recovered within 1 hour after the disruption. For any set of uncertain events, analysts should assess the sufficiency of RTOs to meet the MTPD. In the case of insufficiency, additional measures need to be introduced. Moreover, these additions should be analyzed carefully before their adoption because they introduce additional costs, and very often they introduce other kind of problems to the system.

3 Modeling Framework

To assess BCP, we need to identify and analyze any related assets that are involved in the business. To this extend, we use the Tropos Goal-Risk (GR) framework [9] to analyze risk and Time Dependency and Recovery model [8] to capture interdependencies among assets. A Business Continuity Plan (BCP) is defined in terms of a set of RTOs for all assets. Ideally, it must satisfy the MTPD of business objectives required by stakeholders.

In the following subsections, we explain the underlain framework (TDR model), which captures time dependencies among assets. Afterward, we present the extension of the GR framework for analyzing the Business Continuity in an organization, and also the process to develop a GR model.

3.1 Time Dependency and Recovery Model

The TDR model allows us to model the interdependencies between assets in realizing business objectives.

Definition 1. *A TDR model is a pair $\langle N, \rightarrow \rangle$ where N is a set of nodes (assets) and $\rightarrow \subseteq N \times N$ represents inter-dependency relations between nodes that is tolerable for a given time t.*

For example, in Fig. 1, the task entry loan application by Bank Employee *(T_{02})* requires the resource secure desktop client *(R_{02})*. We depict this as $T_{02} \xmapsto{15'} R_{02}$ that refers to T_{02} will be not available if R_{02} is unavailable for more than 15 *time unit* (in this paper, we use *minute* as a default time unit). Dash-lines refer to the concept of OR dependency, for instance G_{02} can depend either on T_{01} or T_{02} .

Using the reasoning framework proposed in [8], we can assess the sufficiency of RTOs for all assets against the MTPD of business objectives. Moreover, the proposed tool is able to calculate the Maximum Recovery Time (MRT) of each asset. If all RTOs of assets are less-or-equal of the MRTs, then the continuity of business objectives is guaranteed. Contrarily, the continuity of business might be disrupted. In the case of RTOs have not been defined, we may always use the MRT as threshold for RTO in order to guarantee the business continuity.

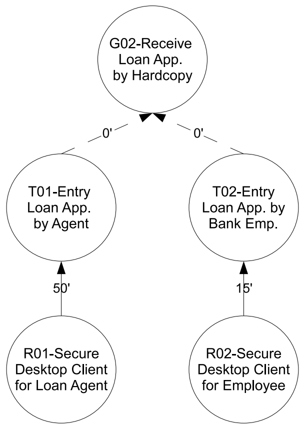

G02-Receive Loan App. by Hardcopy
0'
0'
T01-Entry Loan App. by Agent
T02-Entry Loan App. by Bank Emp.
50'
15'
R01-Secure Desktop Client for Loan Agent
R02-Secure Desktop Client for Employee

Fig. 1. The TDR Model

3.2 The Goal-Risk Framework

To model and assess BCPs, we need to analyze 1) business objectives and their realizations (process and artifacts), 2) interdependencies among assets, and 3) the level of risk that threats business objectives, directly or indirectly. However, the "original" GR framework [11] is able to deal with 1 and 3, while the TDR model focuses more on 2. The idea here is to adapt the notion of inter-dependency relation from the TDR model. Thus, the GR framework is able to capture the assets in an organization and is able to model and analyze the BCP.

The Tropos Goal Risk (GR) framework introduced in [11] (more details in [9]) adopts the idea of three layers analysis from Defect Detection Prevention (DDP) [12]. It consists of three conceptual layers – asset, event, and treatment (as depicted in Fig. 2) – to analyze the risk of uncertain events over organizations' strategies. The **asset layer** analyzes business objectives of the stakeholders and their realizations (i.e., processes and artifacts), whereas the **event layer** captures uncertain events along their impacts to the asset layer and the **treatment layer** models treatments to be adopted in order to mitigate risks.

Definition 2. *A GR model is a set of tuple $\langle \mathcal{N}, \mathcal{R}, \mathcal{I} \rangle$, where:*

- *$\mathcal{N}$ is a set of nodes of three types:* goals, tasks, resources, *and* events*;*
- *$\mathcal{R}$ is represented as $(N_1, \ldots, N_n) \overset{r}{\longmapsto} M$, where $N_i \in \mathcal{N}$, $M \in (\mathcal{N} \cup \mathcal{I})$, and r is the type of the relation. $N_1, \ldots, N_n$ are called* source nodes *and M is the* target node. *r consists of* AND/OR-decomposition, contribution, *and* alleviation, means-end, *and* needed-by[2]*;*
- *$\mathcal{I} \subseteq \mathcal{E} \times (\mathcal{N} \setminus \mathcal{E})$ is a special type of relation, called impact relation. It relates events ($\mathcal{E} \subseteq \mathcal{N}$) with other constructs ($\mathcal{N} \setminus \mathcal{E}$) representing the severity of events toward the asset layer.*

[2] This is a new kind of relation that was not used in the original GR framework.

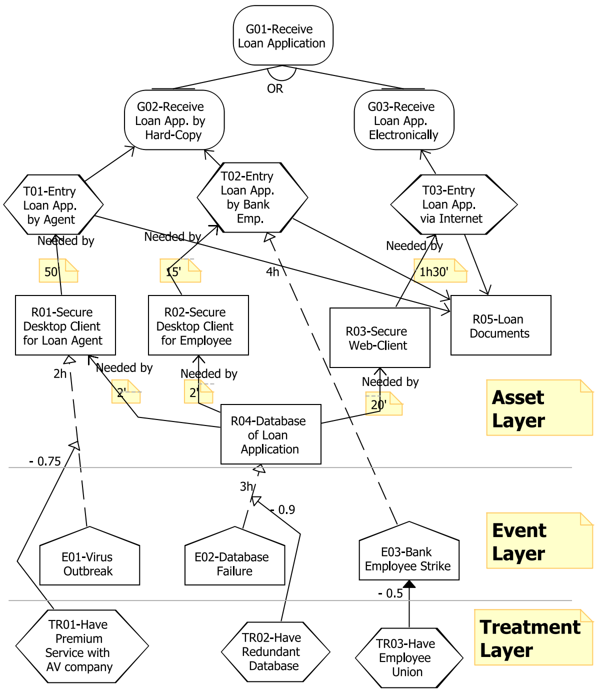

G01-Receive Loan Application
OR
G02-Receive Loan App. by Hard-Copy
G03-Receive Loan App. Electronically
T01-Entry Loan App. by Agent
T02-Entry Loan App. by Bank Emp.
T03-Entry Loan App. via Internet
Needed by
50
5'
4h
1h30'
R01-Secure Desktop Client for Loan Agent
R02-Secure Desktop Client for Employee
R03-Secure Web-Client
R05-Loan Documents
2h
2'
20'
R04-Database of Loan Application
Asset Layer
- 0.75
3h
- 0.9
E01-Virus Outbreak
E02-Database Failure
E03-Bank Employee Strike
Event Layer
- 0.5
TR01-Have Premium Service with AV company
TR02-Have Redundant Database
TR03-Have Employee Union
Treatment Layer

Fig. 2. The Extended GR Model

Goals (depicted as ovals in Fig. 2) represent the objectives that actors intend to achieve. *Tasks* (hexagons) are course of actions used to achieve goals or treat events. Tasks might need *resources* (rectangles) during their execution or even produce resources. To avoid confusion between tasks for achieving goals and tasks for mitigating risk, from now on we name the former as tasks and the latter as *treatments*, respectively. To model a situation where a task is a means to achieve the end-a goal, we adopt the Tropos [13] *means-end* relation (line arrow), and similarly for the task that produces a resource. So for example, the tasks entry loan application by agent *(T_{01})* and entry loan application by bank employment *(T_{02})* are means to achieve the goal receive application by hard-copy *(G_{02})*. Moreover, either T_{01} or T_{02} produce the resource of loan documents *(R_{05})*, which later might be used in other processes.

To analyze BCP, a GR model needs also to capture assets dependencies. We introduce the *needed-by* relation, adapted form the TDR model, to model a task that needs a particular resource, a resource that needs another resource or a task that needs another task. This type of relation is annotated with time, which represents the maximum disruption period that is tolerable by dependent assets (we use minutes as default time unit). For example, Secure desktop client for Loan Agent *(R_{01})* is needed by the task T_{01} (the time-dependency is 20 minutes) and R_{01} requires to access database of loan applications *(R_{04})* (the time-dependency is 2 minutes). The disruption of R_{01} will not result in the failure of T_{01} for more than 20 minutes. For computing the MRT, a

GR model (2) uses the proposed reasoner in [8] since we can develop the corresponding TDR model (1) following the rules described in the Section 4.

The fulfillment of goals (also the execution of tasks and the provision of resources) might be disrupted by the occurrence of uncertain events (pentagons). Here, an event is characterized into two attributes: likelihood (λ) and its consequences [14].[3] To simplify the calculation the *likelihood* is represented in terms of the number of occurrences within a specific period (in this framework, the time period is a year) judged by experts.[4] For instance, if an expert judges that the likelihood of an event is 0.1, then it implies that the event took place once in 10 years. To model consequences, we use the *impact* relations (dash-line with hallow arrow).

Possible treatments are introduced in the treatment layers. They are aiming at mitigating risks (events with negative impact). Moreover, with the help of a CASE tool [5], analysts can define, which treatments should be adopted to achieve the acceptable risk level.

3.3 Modeling Process

The modeling process of a GR model starts from the **asset layer**, which consists of objectives, processes, and artifacts. We initially identify all business objectives (goals) of stakeholders and then we refine them by iterative decompositions. For example, we identify that stakeholders have two top-goals: receive loan application *(G_{01})* and handle loan application *(G_{04})*. Then, goal G_{01} is OR-decomposed into receive loan application by hard-copy *(G_{02})* or receive loan application electronically *(G_{03})*. The refinement process continues until each leaf-goal is tangible, that is there exists at least a task to fulfill it. As soon as analyst identifies the processes/tasks (the operation level of the asset layer) that realize the business objectives, the modeling process continues with the refinement of tasks using AND/OR decomposition. The process stops when each leaf-task is an atomic activity that cannot be anymore broken down in sub activities [10]. Finally, we analyze whether there are necessary artifacts/resources (e.g., T_{01} requires R_{01}) to execute tasks (the artifact level of the asset layer). Some of resources may require other resources (e.g., R_{01} requires R_{04}) or produced by the execution of tasks (e.g., T_{01} produces R_{05}).

The fulfillment of business objectives might be disrupted by the occurrence of uncertain external events. Essentially, in **the event layer** we identify negative events (i.e., threats, un/intentional events, incidents) that disrupt business objectives direct or indirectly (by disrupting the supporting assets). For instance, the resource secure desktop client for loan agent *(R_{01})* might be disrupted by the occurrence of virus outbreak *(E_{01})*. This event will cause 2 hours of unavailability for R_{01} . Taxonomy-based approaches, such as Computer Program Flaws [15], Faults [16], can be used to identify this class of events related to the software systems. For identifying events in other domains (e.g., management, financial), analysts should conduct the interviews to the

[3] In this paper, we consider only events with negative consequences (i.e., risks, threats, incidents).

[4] The model allows us to represent the likelihood in terms of *Probability Distribution Function* for a better result (i.e., precision), but it requires more complex mathematical computation.

[5] `http://sesa.dit.unitn.it/sistar_tool`

related stakeholders or the domain experts. However, the availability of resources is not sufficient to guarantee the continuity of business objectives. There could be circumstances where the disruption is introduced from the process level. For example, the task entry loan application by bank employee *(T_{02})* can be unavailable for 4 hours because of the occurrence of event bank employee strike *(E_{03})*. To identify risks at this level, we can use organizational-driven [17,18] and again taxonomy-based [19] approaches.

Suppose the bank intends, also, to satisfy the goal ensure loan repayment. This objective can be realized in two different ways (processes): 1) assessing the credit score and 2) underwrite the loan according to the credit score. Though, the bank is able to carry on both processes to ensure the repayment of the loan, the risk of a economic crisis may still disrupt the business objective. For this type of events, obstacle approach [20] can be used.

We recommend analysts to start the event identification process from the artifact level and then move up to the process and objective level. In this manner, we prevent the spurious identification of an event's impact. For example, the event virus outbreak *(E_{01})* might be modeled to impact the goal receive loan application *(G_{01})*. However, this is not correct because actually E_{01} obstructs R_{01} that is used to fulfill G_{01} . In other words, if an event disrupts a resource, then certainly it will also produce a similar effect to tasks that use such a resource and consequently this will affect goals that the tasks are supposed to satisfy. Conversely, in the case of the event economic crisis and the goal repayment of the loan, the event does not obstruct any task or any resource that are realized the goal. Identified events are refined using again decomposition relations until all leaf-event are assessable.

Once the strategic and event layers have been analyzed, we identify and analyze the countermeasures that might be adopted to mitigate risk in **the treatment layer**. To mitigate risks, treatments can operate in two ways: reducing likelihood and/or reducing Time-Period of Disruption (TPD). To reduce the likelihood, we use the *contribution* (depicted as line with filled-arrow) with the annotation ($[-1, 0)$) indicating the extent of likelihood reduction. For instance, the treatment have employee union *(TR_{03})* mitigates to 50% the likelihood of the event bank employee strike *(E_{03})*. It is presumably because the union may intermediate the conflict between employees and employers. However, we use the *alleviation* relation (depicted as line with hallow-arrow) to capture the mitigation of risk impact (in this context is the reduction of TPD). For instance, the treatment have redundant database *(TR_{02})* reduces 0.9 of the TPD caused by the event database failure *(E_{02})*.

Summing up, we have revisited the semantics of relations in the GR framework to reason about business continuity. For instance, in [11] the GR model cannot model the time-dependency among the constructs. Moreover, a impact relation, initially, represents how much evidence (satisfaction and denial) is propagated to the asset layer once an event occurs. To model "disruption", we need to revisited the semantic of this relation. In this case, an impact relation depicts how long is the disruption once an event occurs. By means of this model, ones can reason about the sufficiency of existing BCP, in terms of RTO, to meet the MTPD. The following section, we present the analysis supported by the model.

4 Analysis Process

Once we have the extended GR model we can analyze the continuity of the business objectives performing two different kinds of analysis.

– *Treatments Analysis*, intended to elicit all possible sets of treatments that are able to mitigate the risk until the acceptable level. Analysts will choose the most adequate mitigation to introduce following some criteria (e.g., additional costs, possible side-effects).
– *Cost-Benefit Analysis*, aiming at identifying the most cost-effective treatments to reduce the loss introduced by business discontinuity. This analysis is useful when there is no possible set of treatments that is able to reduce the level of risk until the acceptable level. In this case, analysts typically choose the most cost-effective set of treatments.

Inputs for both analyses are:

1. A multi-layers model (e.g., Fig. 2 and Fig. 4);
2. Acceptable risk, represented in terms of pairs Maximum Time Period of Disruption (MTPD) and Maximum Likelihood (Max.λ) of disruption for each top goal (e.g., $MTPD(G_{01}) = 60$ minutes - Max.$\lambda(G_{01}) = 2$, $MTPD(G_{04}) = 120$ minutes-Max.$\lambda(G_{04}) = 2$);
3. "Significant" business objectives, which are defined as top-level goals and other subgoals that the stakeholders believe to be important for the organization. For each of these goals, we specifies its utility for the organization[6] (e.g., $Utility(G_{01}) = 80$, $Utility(G_{02}) = 50$);
4. Likelihood of events (e.g., $\lambda(E_{01}) = 12$, $\lambda(E_{03}) = 3$);
5. Treatments costs (e.g., $Cost(TR_{01})$=200, $Cost(TR_{02})$=70).

Definition 3. *For any given Multi-layers model $\langle \mathcal{N}, \mathcal{R}, \mathcal{I} \rangle$, we build a TDR model $\langle N, \rightarrow \rangle$, where:*

– *N is $\mathcal{N}$ in the asset layer;*
– *$\rightarrow$ is constructed from $\mathcal{R}((N_1, \ldots, N_n) \xmapsto{r} M)$ where $N_1, \ldots, N_i, M \in \mathcal{N}$ in the asset layer*

$$\rightarrow = \bigcup_{\mathcal{R}} \begin{cases} N \xmapsto{t} M, & \textit{if } r = \textit{needed-by}^{7}; \\ N \xmapsto{0} M, & \textit{if } r = \textit{means-end} \wedge M \textit{ is a goal} \wedge N \textit{ is a task}; \\ M \xmapsto{0} N, & \textit{if } r = \textit{means-end} \wedge M \textit{ is a resource} \wedge N \textit{ is a task}; \\ \bigcup_{i=1\ldots n}(N_i \xmapsto{0} M), & \textit{if } r = \textit{decomposition}. \end{cases}$$

[6] We quantify the utility in the range $[0, 100]$. Conceptually, the notion of utility and value are different as indicated in literature about *expected utility* and *expected value* [21]. To assess the utility of an asset, one can assess it by summing up all the values generated by the assets. For instance, a server may have a value not more than 10000, but it may have utility much more beyond its value.

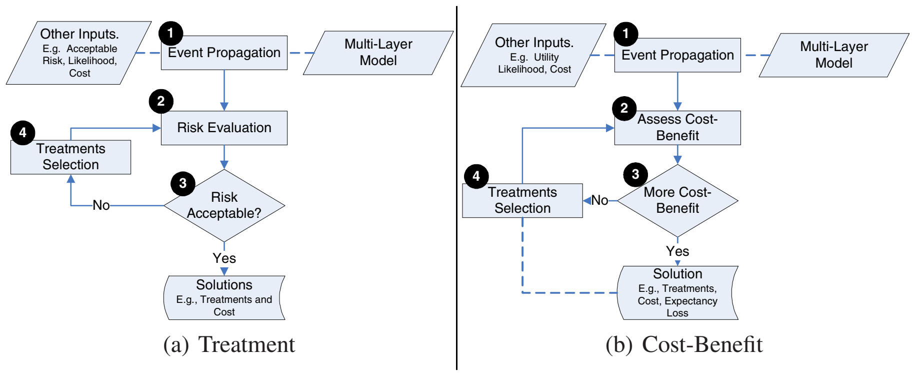


Fig. 3. Analysis Process

Compare Fig. 1 and Fig. 2 to have an idea of the correspondence between a TDR model and an Extended GR model. Given a TDR model $\langle N, \rightarrow \rangle$, for each $n \in N$, the MRT $(mrt(n))$ is calculated as follow [8]:

$$mrt(n) = \begin{cases} MTPD_n, & \text{if } N \text{ is a top-goal;} \\ min\{mrt(m) + t)|n \overset{t}{\longmapsto} m\}, & \text{otherwise.} \end{cases}$$

4.1 Treatment Analysis

Treatments analysis is represented step-by-step in Fig. 3(a). (Step 1) Risks – likelihood and consequences – of events are propagated throughout the model. (Step 2) We evaluate whether it is possible to satisfy all top goals with a risk under given values. This is done looking at how much the likelihood of top-goals and how long for they will be disrupted. If the risk is unacceptable (Step 3), then we refine the model introducing treatments. In this framework, we adopt the algorithm *Find Treatments* proposed in [9] to identify the necessary treatments. Essentially, the algorithm is an adaptation of the *greedy search algorithm* [22] that aims at suppressing the increase of costs because of new treatments. If the TPD of top-goals is not acceptable (TPD greater than MDTP), then the algorithm will propose treatments connected by alleviation relations. If the TPD is equal to MTPD, then it is acceptable if it occurs less-or-equal than Max.λ, otherwise the algorithm will propose the treatments connected by contribution relations to the event layer (Step 4). Notice in the worst case, this process will explore all possible subsets of treatments (i.e., $2^{N(treatments)} - 1$), which hardly will happen in practice. Finally, we possibly obtain different solutions (a solution consists of several treatments) that satisfy the acceptable risk and cost, and then we decide on the bases of criteria such as cost, stakeholders' preference, company culture, etc. which solution to implement.

4.2 Cost-Benefit Analysis

Cost-benefit analysis is useful when analysts cannot find any possible composition of treatments to mitigate the risk until the acceptable level. This analysis is aiming at

[7] t is the time-dependency in a *needed-by* relation.

finding the most advantageous (i.e., cost effective) solution. The notion of advantageous (ADV) is represented in terms of the ratio between benefit and cost (1) [8], while benefit is modeled as an inverse function of the loss expectancy - LE -(2)[9].

$$ADV(S) = \frac{1}{\sum\limits_{G \in significant\text{-}goals} LE(G) \times Cost(S)} \quad (1)$$

$$LE(G) = [\![\lambda(G) - Max.\lambda(G)]\!] \times Utility(G) \times [\![TPD(G) - MTPD(G)]\!] \quad (2)$$

Essentially, the loss is introduced when the TPD is greater than MTPD and it happens more often than Max.λ. In this framework, the loss expectancy is calculated as multiplication of the likelihood distance, the utility of the goal, and the overhead of disruption period.

The overall process of cost-benefit analysis is depicted in Fig. 3(b). (Step 1) a set of treatments is selected, and the loss expectancy of every significant goals and the total cost are calculated to obtain the ADV (Step 2). This process continues exploring every possible combination of treatments (Step 3). Moreover, the notion of cost-benefit might be enriched by considering other factors (e.g., time of implementation, intangible values) besides only loss-expectancy and cost. Notice this process is an exhaustive process that requires to explore all possible subset of treatments. However, some optimization can be taken to reduce the possible search space. For instance, the algorithm records the most cost-effective solution ignoring the branch of search space, which is less beneficial than the recorded solution. Finally, (Step 4) the result of this process is only a solution that theoretically, based on the equation (1), is the most cost-effective solution. Typically, this type of solution would be easy to get an approval by the stakeholders because it proposes the set of treatments, which is the most cost-effective. Moreover, this analysis can be used, in conjunction with the treatment analysis, to evaluate among proposed solutions.

5 Validation through an Example in Large

To evaluate our approach and its implementation, we ran a number of experiments with the *Loan Origination Process* case study that is a simplification of SERENITY e-Business scenario [23]. As illustrated in Fig. 4, let consider two top goals for the bank: receive loan application *(G_{01})* and handle loan application *(G_{04})*. Suppose stakeholders expressed their acceptable risks (i.e., MTPD, Max.λ) for the two goals as indicate in Table 1. For a given MTPD, we compute the MRT of every asset (indicated as the number at upper-left every constructs in Fig. 4) required to satisfy the MTPD. Suppose also, stakeholders argue about the importance of subgoal G_{02}, that can endanger the image of the organization in case it will not be satisfied (even if G_{01} is satisfied). We quantified the G_{02} utility as 50, which is slightly lesser than the utility for G_{01} ($Utility(G_{01}) = 80$). Differently, goal G_{04} $Utility(G_{04}) = 40$) results being less important than G_{01} and G_{02} since its failure will not be visible outside of the organization.

[8] Analysts must adopt at least a treatment to mitigate risk and therefore the $Cost$ cannot be 0.

[9] The function "$[\![x]\!]$" never results a value lower than 0. E.g., $[\![5]\!] = 5$, $[\![-2]\!] = 0$, $[\![-0.002]\!] = 0$.

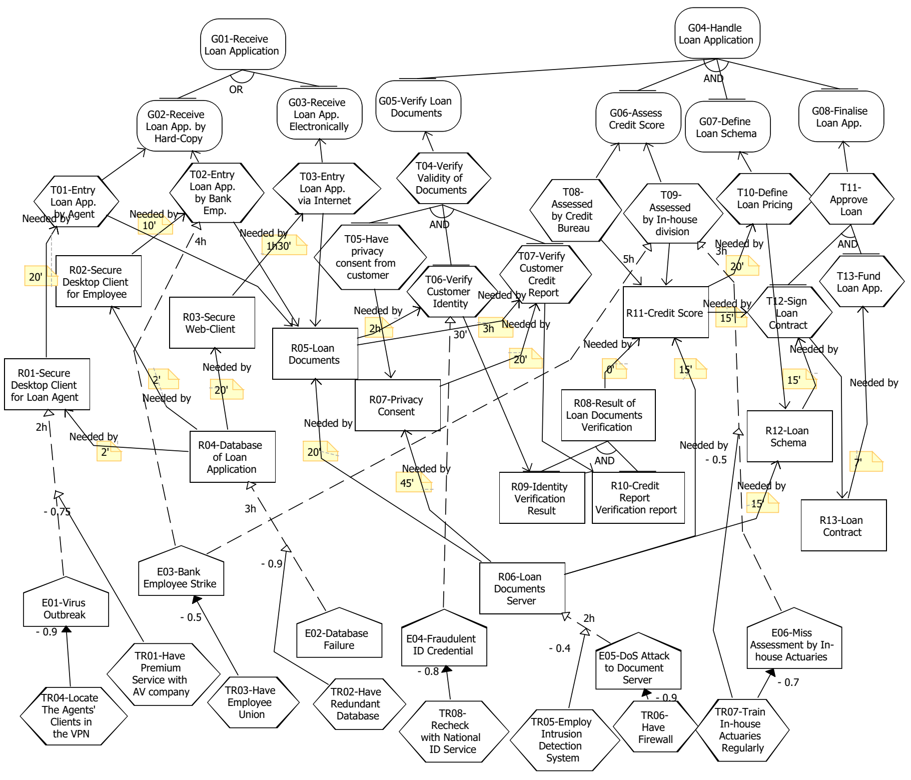

G01-Receive Loan Application
G04-Handle Loan Application
OR
AND
G02-Receive Loan App. by Hard-Copy
G03-Receive Loan App. Electronically
G05-Verify Loan Documents
G06-Assess Credit Score
G07-Define Loan Schema
G08-Finalise Loan App.
T01-Entry Loan App. by Agent
T02-Entry Loan App. by Bank Emp.
T03-Entry Loan App. via Internet
T04-Verify Validity of Documents
T05-Have privacy consent from customer
T06-Verify Customer Identity
T07-Verify Customer Credit Report
T08-Assessed by Credit Bureau
T09-Assessed by In-house division
T10-Define Loan Pricing
T11-Approve Loan
T12-Sign Loan Contract
T13-Fund Loan App.
R01-Secure Desktop Client for Loan Agent
R02-Secure Desktop Client for Employee
R03-Secure Web-Client
R04-Database of Loan Application
R05-Loan Documents
R06-Loan Documents Server
R07-Privacy Consent
R08-Result of Loan Documents Verification
R09-Identity Verification Result
R10-Credit Report Verification report
R11-Credit Score
R12-Loan Schema
R13-Loan Contract
E01-Virus Outbreak
E02-Database Failure
E03-Bank Employee Strike
E04-Fraudulent ID Credential
E05-DoS Attack to Document Server
E06-Miss Assessment by In-house Actuaries
TR01-Have Premium Service with AV company
TR02-Have Redundant Database
TR03-Have Employee Union
TR04-Locate The Agents' Clients in the VPN
TR05-Employ Intrusion Detection System
TR06-Have Firewall
TR07-Train In-house Actuaries Regularly
TR08-Recheck with National ID Service
Needed by
20'
10'
1h30'
2h
3h
30'
45'
15'
7'
2'
0'
4h
5h
- 0.75
- 0.9
- 0.5
- 0.8
- 0.4
- 0.7
- 0.5

Fig. 4. The Model for Assessing the BCP of Loan Originating Process

Table 1. The Inputs of Top Goals and "Significant" Goals

Goals	MTPD(G)	Max. $\lambda(G)$	$Utility(G)$
G01 Receive Loan Application	60	2	80
G04 Handle Loan Application	120	2	40
G02 Receive Loan App. by Hard-Copy			50

Given these inputs, in Table 3 we see how risks disrupt the business continuity. For instance, R_{01} should have at most 2 times of 80 minutes of disruption (MRT) in one year. Unfortunately, the impact of E_{01} results in 12 times of 2 hours disruption, which is unacceptable. However, the assets of T_{02} , T_{09} , T_{06} are not at risk because either they occurs less than 2 times a year or their disruption is less than their MRT. To mitigate such risk, treatment analysis enumerates 81 possible solutions (i.e., sets of treatments) that can satisfy the stakeholders' inputs. For the sake of simplicity, we concentrate only on five of them, namely $S_1 - S_5$ as indicated in Table 2.

From Table 3, we can observe that the MRT of R_{06} is 80 minutes for 2 times/year. However, with S_2 the event E_{05} is mitigated into 2 hours for 2 times/year to R_{06} , which is acceptable by the stakeholders because the likelihood of the disruption is not

Table 2. Total Cost of Possible Treatments

Treatment	Cost	S_1	S_2	S_3	S_4	S_5
TR01 Have Premium Service with AV company	200	X			X	X
TR02 Have Redundant Database	50	X	X	X	X	X
TR03 Have Employee Union	100					
TR04 Locate The Agents' Clients in the VPN	90		X	X		
TR05 Employ Intrusion Detection System	30	X		X		X
TR06 Have Firewall	10		X		X	X
TR07 Train In-house Actuaries Regularly	70	X	X	X	X	X
TR08 Recheck with National ID Service	40					
Total Cost		350	220	240	330	360

Table 3. Risks in The LOP scenario Initial and After Treatments Adoption

Event-Src	Target	MRT	Init	S_1	S_2	S_3	S_4	S_5
E01 Virus Outbreak	R01	2-80'	12-2h	12-30'	1.2-2h	1.2-2h	12-30'	1.2-30'
E02 Database Failure	R04	2-72'	10-3h	10-18'	10-18'	10-18'	10-18'	10-18'
E03 Bank Employee Strike	T02	2-1h	2-4h	2-4h	2-4h	2-4h	2-4h	2-4h
E03 Bank Employee Strike	T09	2-2h	2-5h	2-5h	2-5h	2-5h	2-5h	2-5h
E04 Fraudulent ID Credential	T06	2-2h	24-30'	24-30'	24-30'	24-30'	24-30'	24-30'
E05 DoS Attack to Doc. Server	R06	2-80'	20-2h	20-72'	2-2h	20-72'	2-2h	2-72'
E06 Miss Ass. In-house Actuaries	T09	2-2h	4-3h	1.2-1.5h	1.2-1.5h	1.2-1.5h	1.2-1.5h	1.2-1.5h
Total Cost			350	220	240	330	360	

exceeded. However, S_5, which includes treatment TR_{05}, results in 72 minutes for 2 times/year. It implies the business is never discontinued because the TR_{05} can be recovered before the disruption appears in the business level. Each solution has a different cost and also a different impact on the reduction of risk, as presented in Table 3. Notice that all solutions ($S_1 - S_5$) produce an acceptable level of risk, but S_2 results being the cheapest solution. However, S_3 can be also a good candidate since it can reduce, further, the outage-period of R_{06} from 2 hours to 72 minutes with only a bit higher cost. Decision about S_2 or S_3 is now responsibility of analysts, they have to evaluate what is better for the organization.

To show the cost-benefit analysis, we suppose now that stakeholders are more risk averse than in the previous case. MTPD for goals G_{01} and G_{04} are reduced to 2 and 50 minutes, respectively. Consequently, the new MTPD will results in shorter MRT for each asset. Unfortunately, in this case there is no possible combination of treatments that can reduce the risk until the acceptable level. In this situation, the analyst might simply ignore this fact and accept the risk per se, or consider to adopt the the most beneficial solution.

Notice in Table 3, the asset of T_{02} and T_{09} results in an acceptable disruption because it happens only twice a year. Though in this setting the MRT of T_{06} is much smaller (e.g., $MRT(T_{06}) = 50'$), the recovery time of T_{06} is much smaller (i.e., $30'$) therefore T_{06} cannot caused unacceptable disruption. Conversely, with S_1 the system still suffers 12 times/year an outage of 30 minutes for R_{01} where the MRT of R_{01} is 22 minutes. In other words, the system is discontinued for 8 minutes, 12 times/year (see Table 4 for the complete ones). Consequently, these outages will introduce a loss

Table 4. ADV of Possible Treatments in the LOP scenario

	S_1	S_2	S_3	S_4	S_5
Disruption after Treatments					
R01	12-8'			12-8'	
R04	10-4'	10-4'	10-4'	10-4'	10-4'
R06	20-7'		20-7'		
Results					
Cost	350	220	240	330	360
LE	21920	4800	10400	16320	4800
ADV (in 10^{-7})	1.30344	9.4697	4.00641	1.8568	5.78704

(expectancy) for G_{01} , G_{02} , and G_{04} , as define in equation (2). For instance, in Table 4 the resulting loss expectancy for S_1 is 21920 with a cost of 350. Looking at the table, S_2 results the most cost-effective solution, the lowest level of LE and the cheapest cost.

To summing up, this section has presented how this approach works in two settings: 1) resulting a set of countermeasures that need to be introduced to ensure the business continuity of an organization and 2) to find the most cost-effective set of treatments to maintain the business continuity. This approach does not require very precise inputs (e.g., likelihood, time-dependency, etc.). However, we recommend to analysts to use the worst possible scenarios while assessing the inputs, though it means "overshooting risks".

6 Related Work

KAOS [20,24], a goal-oriented requirements engineering methodology, has been proposed aiming at identifying not only *what* and *how* aspect of goals but also *why*, *who*, and *when*. Moreover, KAOS introduces also the concept of *obstacles* [20] and *anti-goal* [24], which can be seen as boundaries in goal analysis. Those two concepts can be used to identify the top-events that may threaten the asset layer of a GR model. Moreover, the refinement of obstacles and anti-goals are compatible of the decomposition of an event.

Liu et al. [25] propose a methodological framework for security requirements analysis based on *i**. They use the NFR framework [26] to support the formal analysis of threats, vulnerabilities, and countermeasures. This framework captures more details of a malicious events occurs by identifying who is the attacker, what are the vulnerabilities, and what countermeasures should be taken. In our work, we do not distinguish between a disruption due to malicious or non-malicious intents.

Moreover, the works, namely Fault Tree Analysis (FTA) [27] or *attack tree* [28], have similar representation with events in the multi-layer model. Those works capture and analyze the events that may harm the system. Therefore, ones may replace the event layer with those works because of familiarity reason. Notice, those works require objective-quantitative data that can be obtained by recording past experiences.

Approaches like Multi-Attribute Risk Assessment (MARA) [29] can improve the risk assessment process by considering multi-attributes. Many factors like reliable, available, safety and confidentiality can result critical for a system and each of them has its own risk value. This introduces the need for the analyst to find the right trade-off among these factors. In this work, we only assess the recoverability property that is part of the availability. Our results in assessing the recoverability of the system can be useful as one of the input to perform MARA.

Electronic Data Processor (EDP) Audit shares many commonalities with the work in Business Continuity Management. Essentially, the EDP Audit is mirror the activity of business audit [30]. It is a process collecting evidence to determine whether IS systems protect assets, maintain the data integrity, achieve the goals of organization effectively, and consume resources efficiently [31]. To achieve this end, auditors should ensure that the EDP contingency plan is sufficient and has been in place. In this domain, our framework may assist the auditors to analyze the sufficiency of the plan (i.e., RTO).

Finally, approaches on business process modeling, such as Business Process Modeling Notation [32], declarative business process [33], might be useful to structure the process level of the asset layer. It is useful to improve the precision of inter-dependency analysis among assets.

7 Concluding Remarks

In this paper, we have presented a comprehensive framework to analyze the business continuity of an organization. The framework models all levels of assets (e.g., objective, process, and artifact) that may be involved in the continuity of the business. In order to guarantee the continuity of business under uncertainty (e.g., incidents, attacks, human-errors, hardware-failures), we need to introduce a set of treatments to mitigate risks. The proposed framework, allows the analysts to explore and analyze all possible sets of treatments that can be introduced to mitigate the risk (severity or likelihood) of these events. Moreover, the framework also proposes cost-benefit analysis that allows the analyst to select the most cost-effective treatments.

As future work, we intend to introduce more precise description of processes and artifacts in the asset layer by means of more expressive languages (e.g., BPMN, ADL). Moreover, we plan to do more works in order to increase the accuracy of the BCP assessment and its usability. We also intend extending the analysis to a multi-actor environment, where an actor may depend on other actors and they may dis/trust each other. It is also interesting to explore BCP in organization where business objectives and activities are outsourced to other parties.

However, we are aware that the continuity/recoverability problem is only one issue of a critical system (i.e., security and dependability properties). Therefore, the continuity of a business is necessary for a secure and dependable system but it is not sufficient. There are other issues, such as confidentiality, that may compromise the system though the continuity of business is still guaranteed.

Acknowledgment

This work has been partly supported by the projects EU-SERENITY and PRIN-MENSA. Thanks to Emmanuele Zambon for the discussion and inputs on this work.

References

1. Mate, J.L., Silva, A. (eds.): Requirements Engineering for Sociotechnical Systems. Information Science Pub., Hershey (2005)
2. Neumann, P.G.: RISKS-LIST: RISKS-FORUM Digest (accessed May 27, 2008), http://catless.ncl.ac.uk/Risks/
3. Basel Committee on Banking Supervision: Basel II: International Convergence of Capital Measurement and Capital Standards: a Revised Framework (June 2004), http://www.bis.org/
4. Sarbanes, P., Oxley, M.G.: Public company accounting reform and investor protection act. Government Printing Office, Washington (2002)
5. BSI: Business Continuity Management. BSI 25999-1 (2006)
6. Doughty, K. (ed.): Business Continuity Planning Protecting Your Organization's Life. Best practice series. Auerbach, Boca Raton (2001)
7. Lam, W.: Ensuring Business Continuity. IT Professional 4, 19–25 (2002)
8. Zambon, E., Bolzoni, D., Etalle, S., Salvato, M.: A Model Supporting Business Continuity Auditing and Planning in Information Systems. In: Proc. of ICIMP 2007, p. 33 (2007)
9. Asnar, Y., Giorgini, P.: Risk Analysis as part of the Requirements Engineering Process. Technical Report DIT-07-014, DIT - University of Trento (March 2007)
10. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, New York (2007)
11. Asnar, Y., Giorgini, P.: Modelling Risk and Identifying Countermeasures in Organizations. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 55–66. Springer, Heidelberg (2006)
12. Feather, M.S., Cornford, S.L., Hicks, K.A., Johnson, K.R.: Applications of Tool Support for Risk-Informed Requirements Reasoning. Computer Systems Science & Engineering 20(1) (2005)
13. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. JAAMAS 8(3), 203–236 (2004)
14. ISO/IEC: Risk Management-Vocabulary-Guidelines for Use in Standards. ISO/IEC Guide 73 (2002)
15. Landwehr, C.E., Bull, A.R., McDermott, J.P., Choi, W.S.: A Taxonomy of Computer Program Security Flaws. ACM Comp.Surveys 26(3), 211–254 (1994)
16. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic Concepts and Taxonomy of Dependable and Secure Computing. TDSC 1(1), 11–33 (2004)
17. Bhuiyan, M., Islam, M., Koliadis, G., Krishna, A., Ghose, A.: Managing business process risk using rich organizational models. In: Computer Software and Applications Conference, 2007. 31st Annual International. COMPSAC 2007, vol. 2, pp. 509–520 (2007)
18. COSO: Enterprise Risk Management - Integrated Framework. Committee of Sponsoring Organizations of the Treadway Commission (September 2004)
19. Carr, M.J., Konda, S.L., Monarch, I., Ulrich, F.C., Walker, C.F.: Taxonomy-Based Risk Identification. Technical Report CMU/SEI-93-TR-6, SEI-CMU (June 1993)
20. van Lamsweerde, A., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. TSE 26(10), 978–1005 (2000)
21. Bernoulli, D.: Exposition of a New Theory on the Measurement of Risk. Econometrica 22, 23–36 (1954) (original 1738)
22. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)

23. Asnar, Y., Bonato, R., Bryl, V., Campagna, L., Dolinar, K., Giorgini, P., Holtmanns, S., Klobucar, T., Lanzi, P., Latanicki, J., Massacci, F., Meduri, V., Porekar, J., Riccucci, C., Saidane, A., Seguran, M., Yautsiukhin, A., Zannone, N.: Security and Privacy Requirements at Organizational Level. Research report A1.D2.1, SERENITY consortium. EU-IST-IP 6th Framework Programme - SERENITY 27587 (November 2006)
24. van Lamsweerde, A., Brohez, S., Landtsheer, R.D., Janssens, D.: From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. In: Proc. of RHAS 2003 (2003)
25. Liu, L., Yu, E.S.K., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: Proc. of RE 2003, pp. 151–161 (2003)
26. Chung, L.K., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
27. Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. NASA (2002)
28. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobbs Journal 12(24), 21–29 (1999)
29. Butler, S.A.: Security Attribute Evaluation Method: a Cost-Benefit Approach. In: Proc. of ICSE 2002, pp. 232–240. ACM Press, New York (2002)
30. Bace, R.G.: Intrusion Detection. Sams Publishing (2000)
31. Weber, R.: EDP Auditing. McGraw-Hill, New York (1982)
32. López, H.A., Massacci, F., Zannone, N.: Goal-Equivalent Secure Business Process Re-engineering for E-Health. In: Proc. of MOTHIS 2007 (2007)
33. Bryl, V., Mello, P., Montali, M., Torroni, P., Zannone, N.: B-Tropos: Agent-Oriented Requirements Engineering Meets Computational Logic for Declarative Business Brocess Modeling and Verification. In: Proc. of CLIMA VIII (2007)

Resource Allocation vs. Business Process Improvement: How They Impact on Each Other

Jiajie Xu, Chengfei Liu, and Xiaohui Zhao

Centre for Information Technology Research
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia
{jxu,cliu,xzhao}@groupwise.swin.edu.au

Abstract. Resource management has been recognised as an important topic for the execution of business processes since long time ago. Yet, most exiting works on resource allocation have not paid enough attentions to process characteristics, such as structural and task dependencies. Furthermore, no effort has been made on optimising resource allocation by improving business processes. To address this issue, we propose an approach that optimises the use of resources in an enterprise by exploring the structural features of a business process and adapting the structures of the business process to better fit the resources available in the enterprise. After a motivating example, we describe a role-based business process model for resource allocation. Then we present strategies for resource allocation optimisation and discuss the relationship between resource allocation and business process improvement. A set of heuristic rules are discussed and algorithms based on these rules are designed for optimising resource allocation with a particular optimisation goal.

1 Introduction

Business Process Management (BPM) is aimed to investigate how to help enterprises improve their business processes, and thereby enable enterprises to achieve their business goals with lower cost, shorter time and better quality. Nowadays business process management systems [11] have been widely used in many business scenarios. Because the execution of business processes depends on the available resources, the performance of a business process is subject to the degree of match between the given resources and the structure of the business process. When the structure of a business process is fixed, the business process performance, in terms of cost and time, may vary greatly with different resource allocation plans. To this end, several works have addressed the impact of business process structures on resource management [2, 4, 8]. Other works [7, 16, 17] have discussed the process evolvement according to the changing user requirements, yet few have taken resources factors into consideration.

We reckon that resource allocation and business process impact on each other. Structures of business process set a constraint on how resources are allocated to tasks due to the dependency. However, it is possible that a business process is not well-defined, and as a result the resources may not be utilised optimally to reach certain business goal. It is desired that the structure of a business process is improved so that resources can be utilised in a more optimal way. However, as far as we know, no work has discussed this kind of improvement. In this paper, we collectively discuss the problems of resource allocation optimisation for business processes, and resource oriented business process improvement.

To incorporate the resource allocation into business process improvement, this paper proposes a role-based business process model to specify the relationship between business processes, tasks, roles and resources. Based on this model, a comprehensive framework is established to pre-analyse and optimise the resource allocation and business process improvement, and thereby adapt the two to the best match. The contribution of this paper to current business process improvement and resource allocation lies in the following aspects:

- Enable the pre-analysis on resource allocation and utilisation before the execution of business processes, and therefore be able to check if some resource allocation requirements can be satisfied;
- Enable the business process structure change to better optimising resource allocation;
- Develop algorithms for allocating resources to a business process with a particular optimisation criterion for achieving minimal cost with a certain time constraint.

The remainder of this paper is organised as follows: Section 2 discusses our problem for collectively optimising resource allocation and improving business processes with a motivating example; Section 3 introduces a role based business process model, which defines the related notions for resource allocation, and the relationship among these notions; Two algorithms for resource allocation optimisation and resource oriented business process improvement are presented in Section 4; Section 5 reviews the work related to our approach, and discusses the advantages of our approach; Lastly, Concluding remarks are given in Section 6.

2 Motivating Example

We use an example to illustrate the problem that we are tackling in this paper. Figure 1 shows a business process with eight tasks and four gateways. Assume that the set of resources used for this business process are given in Table 1 and are classified according to roles. The cost for each role is also shown in the table. A role describes the capability of its resources for performing certain tasks. For each task, the roles that

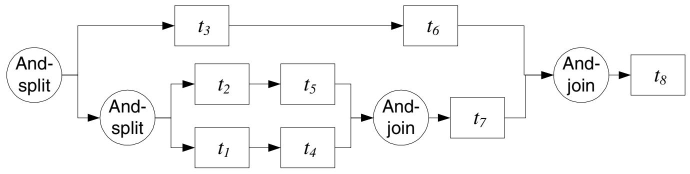


Fig. 1. Business process structure

Table 1. Resource classification

o	Cost	Resource
r_1	\$50/hr	s_{11}, s_{12}
r_2	\$25/hr	s_2
r_3	\$40/hr	s_{31}, s_{32}
r_4	\$20/hr	s_4
r_5	\$25/hr	s_5

Table 2. Capabilities of roles

Task / Role	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
r_1	2hr			2hr			1hr	
r_2	3hr	1.5hr						
r_3		1hr			2hr			2hr
r_4			2hr	2.5hr				
r_5					3hr	2hr		3hr

are capable of performing it are shown is Table 2. The time for a role to perform a task is also indicated in the table. For example, Resources s31 and s32 can perform role r3 at the cost of \$40 per hour. Task t_4 can be performed by resources of role *r1* and role *r4* in 2 hours and 2.5 hours, respectively

Time and cost are two criteria to evaluate the performance of business process. Assume resources are allocated as Figure 2(a). The time required is 7 hours, and meanwhile the expense is \$537.5. In the situation of allocation as Figure 2(b), the cost is reduced to \$455, while the execution time is increased to 9.5 hours. In reality, an enterprise always has a time constraint on a production business process such that the processing time is no more than a deadline. Therefore, the resource allocation is considered to be optimised when the expense is low but the time constraint can be satisfied. In this example, we assume the deadline is 7.5 hours. An optimised resource allocation for this scenario is shown in Figure 2(c) where the expense is 487.5\$ and time is 7.5 hours which just satisfies the time constraint. Compared with Figure 2(c), the allocation in Figure 2(a) is worse because it is more expensive, even though both of them can satisfy time constraint; the allocation in Figure 2(b) is less expensive, however, it violates the time constraint and hence not usable. Therefore, in order to improve the performance of this business process, resources are expected to be allocated as Figure 2(c) under the time constraint.

However, sometimes the structure of business process may prevent resources from being allocated in the optimised way. For instance, if the time constraint is 11.5 hours, Figure 2(b) is the optimised allocation pattern under the business process structure. However, because the limit of time is rather long, t_1 and t_2 can be done in sequential order rather than parallel order. In other words, the business process can be changed to a new business process as shown in Figure 3. If we choose to allocate resources as shown in Figure 2(d), we can achieve an expense of \$457.5, which is less than that in Figure 2(b), and the time is 11.5 hours thereby satisfy the time constraint. Therefore

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_{12}
t_5	s_{31}
t_6	s_5
t_7	s_{11}
t_8	s_{31}

(a)

Task	Resource
t_1	s_2
t_2	s_{31}
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_5

(b)

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_{31}

(c)

Task	Resource
t_1	s_2
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_5

(d)

Fig. 2. Resource allocation

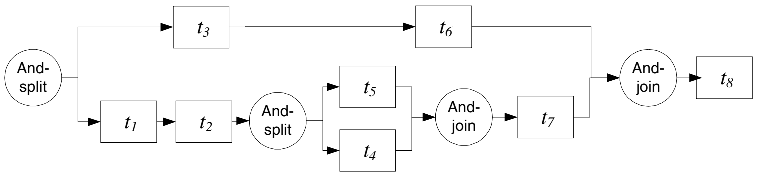


Fig. 3. Changed business process structure

based on the time requirement and available resources, business process redesign may contribute to improve the performance of business process through enabling resource to be allocated in a more optimised way.

From this example, we know that given a set of available resources, optimised resource allocation is based on the structure of business process and the requirements on the business process. Furthermore, a business process can be improved for the purpose of optimising resource allocation. In summary, we expect that the following requirements will be met in our resource allocation scheme:

- It should take into account the structural characteristics of a business process. The structural constraints and dependencies defined in a business process must be followed in resource allocation.
- It should guarantee the resource allocated with minimal expense within a given period.
- When necessary, a business process may be improved for better optimising resource allocation.

3 Role-Based Business Process Model

In this section, a model comprising the definitions for resources, roles, tasks and business processes is introduced to describe the relationships among these notions that will be used in resource allocation and business process improvement.

Definition 1 (Resource). A resource *s* denotes an available unit for executing a task. In real cases, a resource can be a human, a machine or a computer. In this model, a resource has an attribute of role:

– *Role* indicates which group this resource is belonged to according to its position. In this model a resource can have only one role to perform, yet a role may include multiple resources.

Definition 2 (Role). A role *r* denotes a class of resources that own the same capability. In order to simplify resource allocation, we further assume that resources with the same role have the same cost. Therefore role has an attribute of *cost*.

– *Cost* denotes the monetary cost a resource of role *r* is chosen to perform a task. Cost in this model is valued by the hourly pay of the role.

A role is 'an abstraction to define the relationship between a set of resources and the capabilities of resources' [2]. The resources belonging to role *r* may be capable to perform several tasks, where function *capable*(*r*, *t*) depicts such mapping relationships. When resource *s* is capable of performing *t*, *capable*(*s*, *t*) is true. Therefore we have

$$capable(r, t)\ \&\&\ r = role(s) \rightarrow capable\ (s, t)$$

Definition 3 (Task). A task *t* is a logical unit of work that is carried out by a resource. A task has an attribute of *role*:

– *role* defines what kind of resources can perform this task. In other words, a task can be performed by resources that can match the role attribute of task. In this model, one task can have many roles, so each task has a none-empty role set R:$\{r\}$.

Execution time, associated with a role *r* and a task *t*, specifies the duration required for *r* to execute *t*. We denote this by a function time(*r*, *t*). When resource *s* of role *r* is used to execute task *t*, time can be returned by function time(*s*, *t*).

Definition 4 (Business process). A business process represents a series of linked tasks, which collectively describe the procedure how a business goal is achieved. The structure of a business process *p* can be modelled as an directed acyclic graph in the form of $P(T, E, G, type, v_s, v_t)$, where

(1) $T = \{t_1, t_2, \ldots, t_n\}$, $t_i \in$ T ($1 \leq i \leq n$) represents a task in the business process fragment;
(2) $G = \{g_1, g_2, \ldots, g_m\}$, $g_i \in$ G ($1 \leq i \leq m$) represents a gateway in the business process fragment;
(3) E is a set of directed edges. Each edge $e = (v_1, v_2) \in E$ corresponds to the control dependency between *vertex* v_1 and v_2, where $v_1, v_2 \in T \cup G$;
(4) For each $v \in T \cup G$, *ind*(*v*) and *outd*(*v*) define the number of edges which take *v* as terminating and starting nodes respectively.
(5) *type*: $G \rightarrow Type$ is a mapping function, where *Type* = { *And-Join*, *And-Split*, *Or-Join*, *Or-Split* }. Therefore,
    If *type*(*g*) = "*And-Split*" or "*Or-Split*" then *ind*(*g*) = 1, *outd*(*g*) > 1;
    If *type*(*g*) = "*And-Join*" or "*Or-Join*" then *ind*(*g*) > 1, *outd*(*g*) = 1.

(6) v_s is the starting node of the business process fragment, which satisfies that $v_s \in T \cup G$ and $ind(v_s) = 0$;

(7) v_t is the terminating node of the business process p, which satisfies that $v_t \in T \cup G$ and $outd(v_t) = 0$;

(8) $\forall v \in T \setminus \{ v_s, v_t \}$, $ind(v) = outd(v) = 1$.

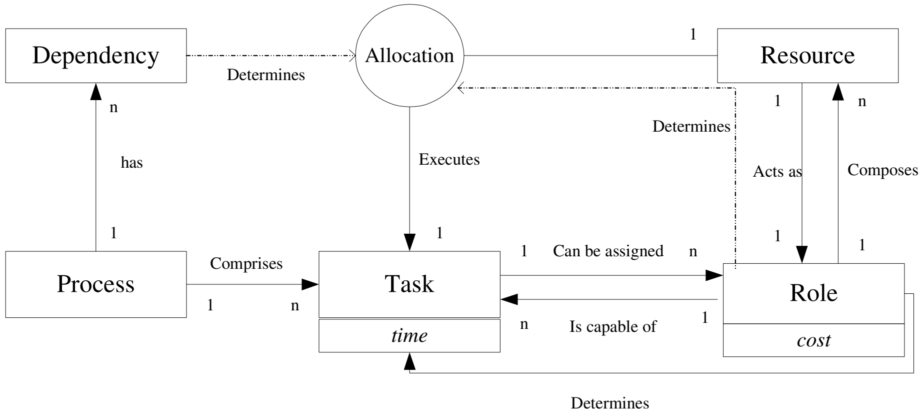


Fig. 4. Business process model

Figure 4 illustrates the relationships among these definitions for resource allocation purposes. A business process consists of a set of tasks and gateways. Each resource performs as one role, and one role may represent multiple resources. Any task can be executed by a set of roles, and a role may be capable of executing many tasks. When allocating resources to tasks, the allocation is subject to the dependency due to the structure of the process and the roles of resources. Cost is an attribute of a role, and it denotes the money that the enterprise has to pay for resources of that role when they are allocated to execute tasks. Time for a task to be executed is determined by which role is assigned to perform this task.

4 Resource Allocation and Business Process Improvement

As explained in the motivating example, we set the cost and time as the resource allocation optimisation criteria for the discussion in this paper. In this section, we first discuss the set of basic rules that follow the optimisation criteria. Then we describe the main data structures used for resource allocation. The main steps of our optimisation algorithms are highlighted afterward. Finally, we present two strategies and corresponding algorithms for resource allocation and business process improvement.

4.1 Basic Rules

As the resource allocation problem is to search for a resource allocation scheme that meets the requirements on cost and time. This searching process has to comply with the following rules:

***Rule* 1.** One resource can only serve for one task at one time. When this rule is violated, we call it resource allocation *conflict* at the task.
***Rule* 2.** The overall execution time of a business process is not allowed to exceed the time limit. If this rule is violated, resources will be required to be reallocated for shortening the process time.
***Rule* 3.** Whenever possible, the expense for executing a business process should be minimal.

In fact, *Rule* 3 is our optimisation objective while *Rule* 1 and *Rule* 2 are the constraints for achieving the objective. In other words, *Rule* 3 is applicable under the condition that *Rule* 1 and *Rule* 2 cannot be violated.

4.2 Data Structure

In Section 2, we introduced two tables for a business process *p*, the *role table* shown in Table 1 and the *capability table* shown in Table 2. For describing our resource allocation algorithms, we also require other two data structures: an *allocation table* and a path table.

An *allocation table* is used to record the allocation information for each task in a business process in the format of the following table.

Task(t)	*Role(r)*	*Resource(s)*	*Start Time(st)*	*End Time(et)*

When a task is allocated with a resource, the allocation table will be appended with a new record, where (1) "Task" *t* denotes the name of task; (2) "Role" *r* denotes the role that is selected for performing; (3) "Resource" *s* of role *r* is the specific resource that is assigned to execute *t*; (4) "Start Time" *st* is the starting time of *t* to be executed; (5) "End Time" *et* is the time *t* finishes. It is computed as $et = st + time(t, s)$, where $time(t, s)$ denotes the time that resource *s* needs to execute task *t*.

A *path table* records the information of all paths from the start node v_s to end node v_t on business process *p*.

Path(i)	*TaskSet(ts)*	*Time(tm)*

In the path table, "*Path*" *i* denotes the path number of this path. "*TaskSet*" *ts* records the set of tasks belonging to path *i*. "*Time*" *t* denotes the total time required to execute all the tasks in *ts*. Note, a task in business process *p* may appear in more than one path.

4.3 Resource Allocation Steps

As the cost and time for executing a task are unknown until it is allocated with actual resource, the analysis on the business process performance is inevitably involved with resource allocation. Resource allocation to tasks will be done in such an optimised way that cost is minimal while satisfying time constraint.

According to the rules introduced in Section 4.1, optimised resource allocation for business process is carried out by the following three steps:

(1) A basic allocation strategy will be applied to searching for a resource allocation satisfying *Rule* 3 and *Rule* 1, which aims the minimal expense for executing a business process with balanced allocation for all paths. Surely, no resource is allocated to more than one task at any time.

(2) In case that the allocation scheme in Step (1) violates *Rule* 2, an adjustment strategy will be applied to shorten the execution time by re-allocating resources until time constraint is satisfied;

(3) In case that the time is less than the limit from Step (1) or Step (2), according to *Rule3*, the adjustment strategy will be applied to do the resource oriented business process improvement in order to achieve a lower expense while maintaining the time constraint to be satisfied.

Step (1) will be discussed in Section 4.4, and Section 4.5 introduces how Step (2) and Step (3) are carried out.

4.4 Basic Allocation Strategy

In the first step, we introduce the basic resource allocation strategy. The goal of this basic strategy is first to minimise the overall expense without considering the time limit. This strategy is achieved by two steps: Firstly, each task is allocated with a role which makes the expense to be minimal, and allocation table is updated according to the resource allocations. However, due to the characteristic of business process structure, it is possible that a role is over-allocated in such a way that at a time, a resource is allocated to perform more than one task, and hence allocation conflict is made and *Rule 1* is not satisfied. Therefore the second step is used to handle allocation conflicts through reallocation. In this procedure, overall expense is aimed to be minimal. Also, in order to improve efficiency, for the routings in parallel or selective blocks, balanced time is preferred for the allocation of different paths.

The basic resource allocation strategy is shown in Algorithm 1. Lines 1-3 initialise several variables *ntbp* for nodes to be processed, *pd* for processed nodes and *pathT* for storing all paths of the business process *p*. The function *genPathTable*(*p*) generates the path table for *p* with time for each path set to 0. Lines 4-17 are the loop for processing one node of the graph for *p*, starting from v_s to v_t. Function *getNextNode*(*ntbp*) (Line 5) finds the next node *v* in *ntbp* such that *v* cannot be processed before its predecessor nodes. For a task node *v* (Line 6), function *bestRole* (*v*) (Line 7) returns the role of minimal expense to execute *v*. A heuristic rule is used here: if two roles are capable to execute task *v* at same expense and one can only be assigned to *v*, then this role is selected. Function *allocRes*(*r*) (Line 8) assigns a resource *s* for *r*. When a resource of role *r* is allocated *v*, the one that is available to perform *v* is selected. Lines 9-10 calculate the maximum ending time *tm* for all paths involving *v* as a node. When all the required information is ready, function *alloc*(*t*, *r*, *s*, *st*, *et*) adds a new record into the allocation table *allocT*. Line 11 resets the ending time for all paths for *v*. After that, for either a task node or a gateway node, the successor nodes of *v* will be added to *ntbp*, and *v* is removed from *ntbp* and added to *pd* (Lines 14-16).

To this point, each task has been allocated with a resource that is least expensive for executing the task. However, we need to check and see if *Rule* 1 is violated. If so, we have to change resources for the task at which the allocation conflict occurs. This is achieved by calling the function *conflictProc*(*allocT, 0, p*) to check the allocation starting from the beginning of the business process (Line 18).

```
Input:     p      –  a business process
           roleT  –  the associated role table for p;
           capaT  –  the capability table for p.
Output:    allocT –  the result allocation table.

1   ntbp = { $v_s$ }; // initial value of nodes to be processed
2   pd = $\emptyset$; // set for processed nodes
3   pathT = genPathTable(p);
4   while (ntbp $\neq \emptyset$)
5     v = getNextNode(ntbp); // get v such that pred(v)$\in$pd or pred(v)=$\emptyset$
6     if (v$\in$P.T) then
7       r = bestRole(v);
8       s = allocRes(r);
9       pts = paths(v); // find all paths that involve v
10      tm = max{pts[i].time}; // maximum time for all paths in pts
11      alloc(v, r, s, tm, tm+time(r, v))$\rightarrow$allocT;
12      for each pt in pts do pt.time += time(r, v) end for;
13    end if
14    ntbp = ntbp $\cup$ succ(v);
15    ntbp = ntbp \ {v};
16    pd = pd $\cup$ {v};
17  end while
18  call conflictProc(allocT, 0, p);
19  return allocT;
```

Algorithm 1. Basic allocation

Algorithm 2 is a function for resolving resource allocation conflicts. *Conflict-Proc*(*allocT, t, p*) is to check conflict for tasks started after time *t* in the order they appear in *p*. When a task v_0 is checked, if conflicts exist, all conflicts involving this task are handled. When there is only one task v_1 conflict with v_0 and they are in the same nearest *And* block, three approaches can be made: reallocation on v_0 or on v_1, or change the structure. The longest path (the path with the most overall executing time from the starting node to the terminating node) processing time in three cases are computed respectively and compared. Process is changed when its processing time is no less than other cases. Otherwise, resource is reallocated at the task which leads to minimal overall processing time. The resource that increase minimal expense but does not increase time is preferred. If there are multiple conflicts, each of them will be handled until there is no task conflict with v_0. Each task *v* conflicting with v_0 is selected, and reallocation is done on the task in the longest path among those paths including v_0 or *v*. Function *replaceRow*(*allocT*, (*v*, *r*, *s*, *v.st*, *v.st.et*)) returns an allocation table that replaces the row for *v* in *allocT* with the specified new row. Function *adjustTime*(*allocT*, *tm, p*) update the start time and end time for those tasks that start after the time *tm* in *allocT* for process *p*, and returns the end time of v_t.

```
function $conflictProc$(allocationTable $allocT$, Time $startTime$, Process $p$)
$timeline$=$startTime$;
$V=\{v \mid v.st \geq startTime\}$
while($timeline < vt.st$)
   select $v_0 \in V : v_0.st$=min($\{v'.st \mid v' \in V\}$)
   $V_c=\{v' \mid v'.st < v_0.et \;\&\; v' \in V\}$
   $timeline = v_0.st$;
   if ($|V_c|$=0) then
      $V=V \setminus \{v_0\}$;
   else if(($|V_c|$=1& $andStruc(v_0, V_c[0])$)) then
      $v_1= V_c[0]$;
      $r_1=nextBestRole(v_0)$; $s_1=allocRes(r_1)$;        //reallocate $v_0$
      $allocT_1=replaceRow(allocT, (v_0, r_1, s_1, v_0.st, v_0.st+ time(r_1, v_0)))$;
      $t_1= adjustTime(allocT_1, v_0.st, p)$;
      $r_2=nextBestRole(v_1)$; $s_1=allocRes(r_2)$;        //reallocate $v_1$
      $allocT_2=replaceRow(allocT, (v_1, r_2, s_2, v_1.st, v_1.st+ time(r_2, v_1)))$;
      $t_2= adjustTime(allocT_2, v_1.st, p)$;
      $p'=changeP(p, v_0, v_1)$;                         //change structure
      $r_3= allocT[v_0].role$; $s_3= allocT[v_0].resource$;
      $allocT_3=replaceRow(allocT, (v_0, r_3, s_3, v_2.et, v_2.et+ time(r_3, v_0)))$;
      $t_3= adjustTime(allocT_3, v_0.st, p')$;
      if $t_3 \leq min(t_1, t_2)$ then   $p=p'$; $allocT=allocT_3$;
      else if($t_2<t_1$) then   $V=V \setminus \{v_0\}$; $allocT= allocT_2$;
      Else   $V=V \setminus \{v_0\}$; $allocT= allocT_1$;
      end if
   else
      $allocT'=allocT$;
      for each $v \in V_c$
         $pt = longestPath(paths(v_0) \cup paths(v)$;
         $v'= pt.Tasks \cap \{ v_0, v \}$;
         $r'=nextBestRole(v')$; $s'=allocRes(r')$;
         $allocT=replaceRow(allocT, (v', r', s', v'.st, v'.st+ time(r', v')))$;
         $adjustTime(allocT, v'.st, p)$;
         if ($v' = v_0$) then $V=V \setminus \{v_0\}$;   break;   end if;
      end for
   end if
end while
return $allocT$;
```

Algorithm 2. Resource Allocation Conflict Resolution

Consider the example introduced in Section 2. The basic allocation algorithm first comes up with an initial allocation as shown in Figure 5(a). After that, allocation conflicts are checked. Starting from t_1, each task will be checked. When t_1 is examined, we can easily detect that task t_1 conflicts with t_2 because they use same resource s_2 in an intervening duration. At this stage, reallocation will be applied on t_1 because it is on the longest path, and hence the overall time can be reduced and times of different paths

Task	Resource
t_1	s_2
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_4
t_6	s_5
t_7	s_{11}
t_8	s_5

(a)

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_4
t_6	s_5
t_7	s_{11}
t_8	s_5

(b)

Task	Resource
t_1	s_{12}
t_2	s_2
t_3	s_4
t_4	s_4
t_5	s_{32}
t_6	s_5
t_7	s_{11}
t_8	s_5

(c)

Fig. 5. Allocation conflict and handling

become more balanced. This results in the change shown in Figure 5(b). Similarly, the conflict between t_4 and t_5 can be resolved by reallocating s_5 with s_{32} as shown in Figure 5(c).

4.5 Adjustment Strategy

A time constraint is not considered in the basic strategy. As Rule 2 stated, the overall execution time of a business process is not allowed to exceed the time limit. In case this rule is violated, i.e., the ending time of v_t in the allocation table is greater than the time limit t_{max}, we have to follow Rule 2 and shorten the time until it is within t_{max}. However, if the overall time is less than t_{max}, we may relax the time and to reduce the expense based on possible business process improvement.

First we discuss the adjustment strategies for the case that time constraint is violated. We have several heuristic rules. As the overall time is dependent on the longest path in the business process, it is more effective to adjust those tasks belonging to the longest path. When a task t currently assigned with resource s is reallocated with s', both the time and cost may change accordingly. Assume $\Delta time$ denotes the time reduced and $\Delta expense$ is expense increased. The value of $\Delta time/\Delta expense$, called as *compensation ratio*, can be used to measure the effectiveness of an adjustment. Obviously a higher *compensation ratio* is preferable because more time can be reduced with less expense. If this adjustment is on a task that belongs to the longest path, the overall time will be reduced accordingly.

Algorithm 3 is designed for handling time constraint violation. The reallocation process is done until time constraint is satisfied. Firstly, the longest path *pt* is selected. In Lines 3-17, we select a task on *pt* to be reallocated. *mcr*, with 0 as initial value, is used to record the maximal compensation ratio for each reallocation, and *mallocT* is the allocation table after such a reallocation has been made. For each task *v* in *pt*, the role r_v of maximal compensation ratio (calculated by *maxRatioRole*(*v*)) for *v* selected, and resource s_v of r_v is reallocated, and *allocT'* is the allocation table to record this reallocation in Lines 5-7. If *v* is in And block, this reallocation may cause allocation conflict, hence function *conflictProc*(*allocT', v.st, p*) is called to handle potential resource allocation conflicts from the starting time of *v*. Lines 9-11 computes the *compensation ratio cr* of this reallocation from overall perspective. If *cr* is lager than *mcr*, *mcr* is updated to *cr* and *mallocT* is changed to *allocT'* in Lines 13-14. After compensation ratio on all the tasks in *pt* has been computed, *allocT* is updated to *mallocT* and returned if time constraint is satisfied.

```
Input:        p        -  allocation table(based on business process p)
              allocT   -  the old allocation table
              pathT    -  a path table for p
              $t_{max}$    -  time limit
Output:       allocT   -  new allocation table

 1   while($v_t$.et> $t_{max}$)
 2     pt = longestPath(pathT);
 3     mcr=0;
 4     for each $v \in$ pt.Tasks
 5       $r_v$= maxRatioRole(v);
 6       $s_v$= allocRes(r);
 7       allocT'=replaceRow(allocT, v, $r_v$, $s_v$, v.st, v.st+time($r_v$, v));
 8       if(v is in And block) then call conflictProc(allocT') end if;
 9       et= allocT[$v_t$].et; exp=expense(allocT[v].role, v);          //previous
10       et'= adjustTime(allocT', v.st, p); exp'=expense($r_v$, v);   //new
11       cr=(et-et') / (exp'-exp);
12       if(cr>mcr) then
13         mcr=cr;
14         mallocT=allocT';
15       end if
16     end for
17     allocT=mallocT;
18   end while
19   return allocT;
```

Algorithm 3. Time constraint violation handling approach

Come back to the example introduced in Section 2, the outcome from Algorithm 1 is shown in Figure 5(c), where the time constraint is violated. At this stage, adjustment strategy must be applied in order to guarantee time constraint be satisfied. The longest path is computed as path 1 ($t_1 \rightarrow t_4 \rightarrow t_7 \rightarrow t_8$). Therefore, reallocation will be done on tasks on path 1. It is easy to calculate and compare the compensation ratio for replacing each task in this path and find that reallocation for t_8, will achieve the maximal compensation ratio. Therefore reallocation on t_8 is applied and the new allocation is shown as Figure 2(c).

Now we discuss the adjustment strategies for the case that the overall time is less than the time limit t_{max}. We also have some heuristic rules to reduce expense while relaxing time. In Algorithm 1, the resource allocation conflict caused by two parallel executing tasks that required same role with minimal expense but there was no sufficient resource for allocating them was resolved by reallocating one of them with a higher expense resource. In this scenario, actually, we could change the structure of the process to support both of them to be assigned with the original cheapest resources. The reduced expense through process change is usually compensated with increased time. Therefore, it is wise to make change to those tasks that do not belong to a long path, because a task in a long path has less room for increasing time.

Algorithm 4 is designed for relaxing time for maximum reduction of expense. *allocT'* and *p'* records the allocation table and the process structure after change, and they are initialised with *allocT* and *p* respectively (Lines 1-2). While the end time of *allocT'* is less than t_{max} (Line 3), the process change is accepted (Line 22) and further process improvement can be made based on *allocT'* and *p'*. Process change is realized in the following way. In the process *p* the shortest path *pt* is first selected (Line 4), and the change is focused on task in *pt*. For each task *v* in *pt* (Line 5), we select a role *r* by function *minRatioRole*(*v*) that looks for the maximum expense deduction with minimum time increase (Line 6). If such a role exists and it is not used by *v* (Line 7),

Input:	*p*	-	allocation table(based on business process *p*)
	allocT	-	the old allocation table
	pathT	-	a bath table for *p*
	t_{max}	-	time limit
Output:	*allocT*	-	new allocation table
	p	-	new process structure

```
1   allocT'=allocT;  //new allocation table after reallocation, initially allocT
2     p'=p;          //new process structure after reallocation, initially p
3     while(allocT'[$v_t$.et]< $t_{max}$)
4       pt=shortestpath(PathT);
5       for each $v \in pt$
6         r = minRatioRole(v);
7         if($r \neq$null and allocT[v].role$\neq$r)
8           ts=getTasks(r, v.st, v.et);
9           mtm= $v_t$.et;
10          for each $v' \in ts$
11            ts'=longestPath(paths(v'));
12            if (ts'.time<mtm)
13              mv=v'; mtm=ts'.time;
14            end if
15          end for
16      end for
17      p'=changeP(p, v, mv);
18      s=allocRes(r);
19      allocT'= replaceRow(allocT, (v, r, s, v.st, v.st+time(r, v)));
20      call conflictProc(allocT', v.st, p');
21      adjustTime(allocT', v.st, p');
22      if(allocT'[$v_t$.et]< $t_{max}$) then allocT=allocT'; p=p'; end if;
23    end while
24    return allocT;
```

Algorithm 4. Relaxation approach

function *getTasks*(*r*, *v.st*, *v.et*) returns the set of tasks that are within the same nearest *And* block, are overlapped with *v*, and are assigned resources with the same role (Line 8), then Lines 9-16 finds the task *mv* with the minimal time in its involved longest path. Lines 17-21 change the structure and replace the resource.

For the example in Section 2, when the time constraint is 11.5 hours rather than 8, allocation after *Algorithm* 3 is shown as Figure 5(c). However, t_2 is not using resource of best role due to conflict with t_1, therefore, we examine if process change contributes to expense reduction. In the new process shown as Figure 3, resource allocation can be made as shown in Figure 2(d) and the expense is reduced. Therefore, in this case, we adopt the changed business process structure in Figure 3 and resource allocation in Figure 2(d).

5 Related Work and Discussion

Resource allocation is a topic related to task/workflow scheduling, which seeks the proper execution sequence for a set of tasks to achieve specific goals. This procedure is dependent on the resource allocated to execute the tasks. Scheduling problem has been discussed at task level and workflow level in previous work. Task level scheduling is based on independent tasks. Many algorithms have been proposed to schedule tasks within the homogenous systems in previous literatures such as [13, 18]. Our paper investigates the resource allocation in a heterogeneous environment, and for heterogeneous scheduling many work has been done. In order to reduce the processing time, Topcuoglu, Hariri and Wu have proposed an Earliest-Finished-Time (HEFT) algorithm in [10]. The HEFT algorithm selects the task with the highest upward rank value at each step and assigns the selected task to the processor. This algorithm can minimise the earliest finished time, but the cost is not considered. The work in [6] deals with problem of scheduling tasks to minimise transition cost within a rigid deadline for completion. A mathematical formulation and a two-phased algorithm are introduced to solve this problem in [6]. All the works on task level scheduling have their limit in effective resource management because they do not follow the structure of a business process.

Compared with task scheduling, workflow scheduling is based on tasks of compulsory execution order constraint. In [5], Johann, Euthimios and Michael have proposed some modelling primitives to express the lower- and upper-bound of time constraints for workflow scheduling. In addition, they have also developed the technique for checking if time constraints are satisfied at process build and instantiation time, and enforcing these constraints at run-time. Their work has solved the problem of deadline constraints and provided the solution about how to avoid the deadline violation, but this work does not include any resource allocation strategy. Work [9, 3] has modelled the workflow with resource constraints. In the framework of [9], workflow scheduling is under both temporal constraints and resource constraints (composed of control constraints and cost constraints). However, this work does not touch how to manage and allocate resource in workflow. A number of Grid workflow management systems such as [1, 12] have proposed scheduling algorithms to facilitate the workflow execution and minimise the execution time. Yu and Buyya [14, 15] have considered not only execution time, but also execution cost. In [15], Yu has proposed a genetic algorithm for scheduling scientific workflows for utility Grid applications by minimising the execution time while meeting user's budget constraint. In [14], the problem of minimising the overall cost while meeting user's deadline constraint for the scientific workflow scheduling is also investigated.

In contrast to the previous work, the work discussed this paper is intended to improve performance of business processes by integrating resource allocation with business process structural improvement. Compared to existing approaches, our approach has the following features:

- Business process improvement for providing better resource allocation. In our approach, the structure of process can be modified to better adapt to the resources allocation when necessary. Based on the analysis on relationship between the types of available resource and the business process structure, such modifications can make the process structure more effective for available resources. In this way, resource allocation on the new process has better performance than allocation on the old process structure.
- Resource optimisation based on business process characteristics. In our approach, business process characteristics, which include the constraints and dependencies pre-defined by the process structure, are preserved in the resource allocation.
- Requirement oriented resource allocation. Our approach is able to guarantee the requirements set for resource allocation been satisfied. In this paper, time constraints and cost requirements for a business process have been considered.

6 Conclusion

This paper discussed the problem of resource allocation for business processes. An approach was proposed to allocate resources to tasks in such a way that the total expense is minimal while the requirement of executing time on a business process is satisfied. In this approach, a basic strategy is applied first to minimise total expense. Then, an adjustment strategy is applied to modify allocation such that the time constraint is met in a smart way. The advantage of this approach over previous approaches lies in the relationship between the effective resource allocation and the business process improvement. To cater for the resource allocation requirements and available resources of an enterprise, the structure of a business process can be changed. After the structure of the business process is changed, the performance of the business process in terms of better utilising resources is improved.

In the future, we plan to take more task dependencies into account in the resource allocation and explore more structure changes of a business process.

References

1. Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K.: Task scheduling strategies for workflow-based applications in grids. In: Proceedings of the 5th International Symposium on Cluster Computing and the Grid, Cardiff, UK, pp. 759–767 (2005)

2. Du, W., Eddy, G., Shan, M.-C.: Distributed resource management in workflow environments. In: Proceedings of the 5th Database Systems for Advanced Applications, Melbourne, Australia, pp. 521–530 (1997)
3. Etoundi, R.A., Ndjodo, M.F.: Feature-oriented workflow modelling based on enterprise human resource planning. Business Process Management Journal 12, 608–621 (2006)
4. Huang, Y.-N., Shan, M.-C.: Policies in a resource manager of workflow systems: modeling, enforcement and management. In: Proceedings of the 15th International Conference on Data Engineering, p. 104. IEEE Computer Society, Sydney (1999)
5. Eder, J., Panagos, E., Rabinovich, M.: Time constraints in workflow systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 286–300. Springer, Heidelberg (1999)
6. Lee, Y.-J., Lee, D.-W., Chang, D.-J.: Optimal task scheduling algorithm for non-preemptive processing system. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 905–910. Springer, Heidelberg (2006)
7. Liu, C., Orlowska, M.E., Li, H.: Automating handover in dynamic workflow environments. In: Proceedings of the 10th International Conference on Advanced Information Systems Engineering, pp. 159–171 (1998)
8. R-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating planning and scheduling in workflow domains. Expert Systems with Applications 33, 389–406 (2006)
9. Senkul, P., Toroslu, I.H.: An architecture for workflow scheduling under resource allocation constraints. Information System 30, 399–422 (2004)
10. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13, 260–274 (2002)
11. van der Aalst, W., van Hee, K.: Workflow management: models, methods, and systems. MIT Press, Cambridge (2004)
12. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the ASKALON grid environment. SIGMOD Record 34, 56–62 (2005)
13. Wu, A.S., Yu, H., Jin, S., Lin, K.-C., Schiavone, G.A.: An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 15, 824–834 (2004)
14. Yu, J., Buyya, R., Tham, C.-K.: Cost-based scheduling of scientific workflow application on utility grids. In: International Conference on e-Science and Grid Technologies, Melbourne, Australia, pp. 140–147 (2005)
15. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14, 217–230 (2006)
16. Zhao, X., Liu, C.: Version management in the business process change context. In: Proceedings of the 5th International Conference on Business Process Management, pp. 198–213 (2007)
17. Zhao, X., Liu, C., Yang, Y., Sadiq, W.: Handling instance correspondence in inter-organisational workflows. In: Proceedings of the 19th International Conference on Advanced Information Systems Engineering, pp. 51–65 (2007)
18. Zomaya, A.Y., Teh, Y.-H.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel and Distributed Systems 12, 899–911 (2001)

Detecting and Resolving Process Model Differences in the Absence of a Change Log

Jochen M. Küster[1], Christian Gerth[1,2], Alexander Förster[2], and Gregor Engels[2]

[1] IBM Zurich Research Laboratory, Säumerstr. 4
8803 Rüschlikon, Switzerland
{jku,cge}@zurich.ibm.com

[2] Department of Computer Science, University of Paderborn, Germany
{gerth,alfo,engels}@upb.de

Abstract. Business-driven development favors the construction of process models at different abstraction levels and by different people. As a consequence, there is a demand for consolidating different versions of process models by detecting and resolving differences. Existing approaches rely on the existence of a change log which logs the changes when changing a process model. However, in several scenarios such a change log does not exist and differences must be identified by comparing process models before and after changes have been made. In this paper, we present our approach to detecting and resolving differences between process models, in the absence of a change log. It is based on computing differences and deriving change operations for resolving differences, thereby providing a foundation for variant and version management in these cases.

Keywords: process change management, process model differences.

1 Introduction

The field of business process modeling has a long standing tradition. Recently, new requirements and opportunities have been identified which allow the tighter coupling of business process models to its underlying IT implementation: In Business-Driven Development (BDD) [11], business process models are iteratively refined, from high-level business process models into models that can be directly executed. In such scenarios, a given business process model can be manipulated by several people and different versions of the original model can be created. At some point in time, these different versions need to be consolidated in order to integrate selected information found in different versions into a common process model. Technically, this consolidation involves inspecting differences and resolving differences by performing change operations on the original model in order to integrate information found in one of the versions.

Detection of differences introduced into a process model is straightforward in a process-aware information system [5,15] that provides change logs (see e.g. [21]). However, there exist scenarios where such a change log is not available: Either the tool does not provide one or process models are exchanged across tool boundaries. In such situations, detection of differences has to be performed by comparing process models before and after changes have been made. For each difference detected, appropriate change operations have to be derived which together can be considered as a reconstructed change log.

In addition, process modeling tools have to fulfill specific requirements concerning user-friendliness: The business user (usually not a computer scientist) should be able to inspect and resolve differences. As a consequence, differences must be displayed in a form that is understandable by the business user, by grouping related differences or those differences that can be resolved together. Similarly, resolution of differences should involve compound change operations (rather than primitive change operations) which enable the business user to deal with differences such as insertion or deletion of a task and automate the reconnection of control flow in the process model.

In this paper, we present our solution to the problem of computing differences, displaying differences and resolving differences between two process models in the situation that *no change log* is available. Detection of differences makes use of the concept of correspondences [13], well-known from model merging and model composition, but enriched with the technique of Single-Entry-Single Exit fragments (SESE fragments) [20]. Using SESE fragments we are able to associate each difference with a compound change operation that resolves the difference. Overall, our approach provides a foundation for variant and version management in cases where no change log is available which is a common situation in process modeling tools and in scenarios where process models are exchanged across tool boundaries.

The paper is structured as follows: Section 2 introduces our scenario for detection and resolution of differences and describes key requirements. Then, in Section 3, we discuss the foundations for our approach, correspondences and SESE fragments. In Section 4 and Section 5 we present our approach for difference detection and visualization. In Section 6, our approach to difference resolution is described and a prototype for initial validation is presented. We conclude with a discussion of related and future work.

2 A Scenario for the Detection and Resolution of Differences

In business-driven development, business process models are manipulated by several persons and multiple versions of a shared process model need to be consolidated at some point in time. A basic scenario is obtained when a process model V_1 is copied and then changed into a process model V_2, possibly by another person. After completion, only some of the changes shall be applied to the original model V_1 to create a consolidated process model. Figure 1 shows an example process model V_1 that has been changed into a process model V_2. In the following, we use process models in a notation similar to activity diagrams in UML [12].

Both models describe the handling of a claim request by an insurance company. V_1 starts with an *InitialNode* followed by the *actions "Check Claim"* and *"Record Claim"*. Then, in the *Decision*, it is checked whether the claim is covered by the insurance contract or not. In the case of a positive result the claim is settled. In the other case the claim is rejected and closed, represented by the *actions "Reject Claim"* and *"Close Claim"*. We now assume that V_2 is derived from V_1 by another business user who introduces and deletes elements, with the final result that is shown in Figure 1. A manual inspection of the process models V_1 and V_2 leads to the identification of the following differences:

- The *action "Check Claim"* is moved into a newly inserted cyclic structure. In addition a new *action "Retrieve additional Data"* is inserted into the cyclic structure.

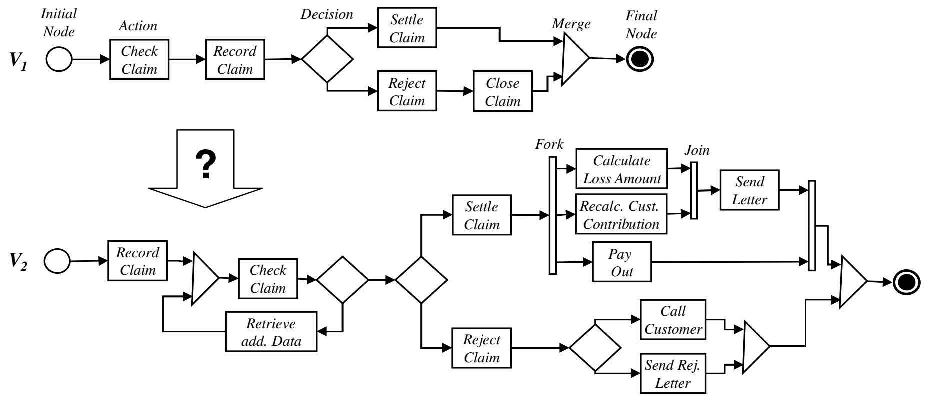


Fig. 1. Versions V_1 and V_2 of a business process model

- A parallel structure (*Fork* and two *Joins*) is introduced in V_2 containing four *actions "Calculate Loss Amount", "Recalculate Customer Contribution", "Pay Out", and "Send Letter"*.
- *Action "Close Claim"* has been deleted in V_2.
- A new alternative structure (*Decision* and *Merge*) is inserted in V_2 together with two *actions "Call Customer"* and *"Send Rejection Letter"*.

```
- deleteEdge(InitialNode, "Check Claim")
- deleteEdge("Check Claim", "Record Claim")
- addEdge(InitialNode, "Record Claim")
- deleteEdge("Record Claim", Decision)
- addControlNode(Merge)
- addEdge("Record Claim", Merge)
- addEdge(Merge, "Check Claim")
- addControlNode(Decision)
- ...
```

Fig. 2. Primitive change operations applied to V_1 in order to obtain V_2

For larger process models, manual identification of differences represents a large overhead. As such, techniques for detecting and resolving differences between process models are required which typically depend on the modeling language as well as on constraints of the modeling environment. In our scenario, we assume that no change log is available. A simple approach would then compute all changed elements and express them using primitive change operations as displayed in Figure 2. For the business user, such a change log is difficult to handle because the relationship between change operations and the process model elements is difficult to determine. Furthermore, changes are not grouped into compound change operations [21] which package several related primitive change operations. For example, for inserting the new alternative structure with the *Decision* and *Merge*, the business user has to insert these two nodes by appropriate *addControlNode* operations, delete edges by *deleteEdge* operations and insert new edges by *insertEdge* operations. This is in contrast to a compound operation that comprises all these change primitives, being close to the conceptual understanding of the change. As a consequence, we define the following requirements a solution for detection and resolution of differences should fulfill:

- (Detection) The solution must provide a technique to re-construct one possible change log which represents the transformation steps for transforming one process model into the other process model.

- (Visualization) Differences should be grouped and associated to areas where they occur in order to improve usability by the business user.
- (Resolution) The solution should enable the business user to resolve differences using compound change operations rather than change primitives manipulating individual process model elements.
- (Resolution) The business user should have the opportunity to select only some of the changes and apply them in any order when possible.

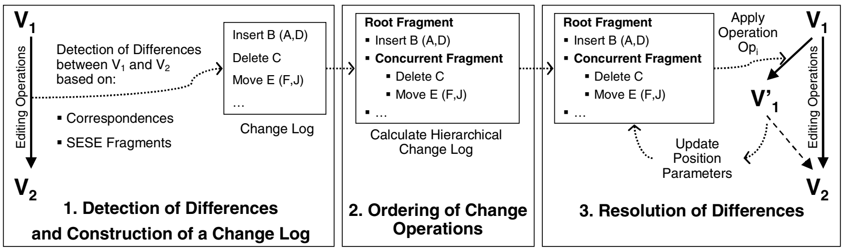

V1
Editing Operations
V2
Detection of Differences between V1 and V2 based on:
Correspondences
SESE Fragments
Insert B (A,D)
Delete C
Move E (F,J)
...
Change Log
1. Detection of Differences and Construction of a Change Log
Root Fragment
Insert B (A,D)
Concurrent Fragment
Delete C
Move E (F,J)
...
Calculate Hierarchical Change Log
2. Ordering of Change Operations
Apply Operation Opi
V'1
Update Position Parameters
3. Resolution of Differences

Fig. 3. Overview of our process merging approach

Figure 3 provides an overview of the approach that we have developed based on these requirements: The first step is to detect differences between the two process models. This detection makes use of correspondences and SESE fragments. For each difference, a change operation is generated which resolves the difference between the two models. In the second step, change operations are ordered according to the structure of the process models. The third step is then to resolve differences between the process models in an iterative way, based on the business user's preferences.

3 Correspondences and SESE Fragments

In this section, we first define business process models and provide a summary of the concepts of Single-Entry-Single-Exit fragments [20]. Then we introduce correspondences. Fragments and correspondences will be later used for detecting differences and also provide a basis for deriving change operations.

3.1 Process Models and SESE Fragments

For the following discussions, we assume a business process model $V = (N, E)$ consisting of a finite set N of nodes and a relation E representing control flow. N is partitioned into sets of *Actions* and *ControlNodes*. *ControlNodes* contain Decision and Merge, Fork and Join, InitialNodes and FinalNodes. In addition, we assume that the following constraints hold:

1. *Actions* have exactly one incoming and one outgoing edge.
2. Nodes are connected in such a way that each node is on a path from the InitialNode to the FinalNode.

3. Control flow splits and joins are modeled explicitly with the appropriate *ControlNode*, e.g. *Fork*, *Join*, *Decision*, or *Merge*. *ControlNodes* have either exactly one incoming and at least two outgoing edges (*Fork*, *Decision*) or at least two incoming and exactly one outgoing edge (*Join*, *Merge*).
4. An *InitialNode* has no incoming edge and exactly one outgoing edge and a *FinalNode* has exactly one incoming edge and no outgoing edge.
5. A process model contains exactly one *InitialNode* and exactly one *FinalNode*.

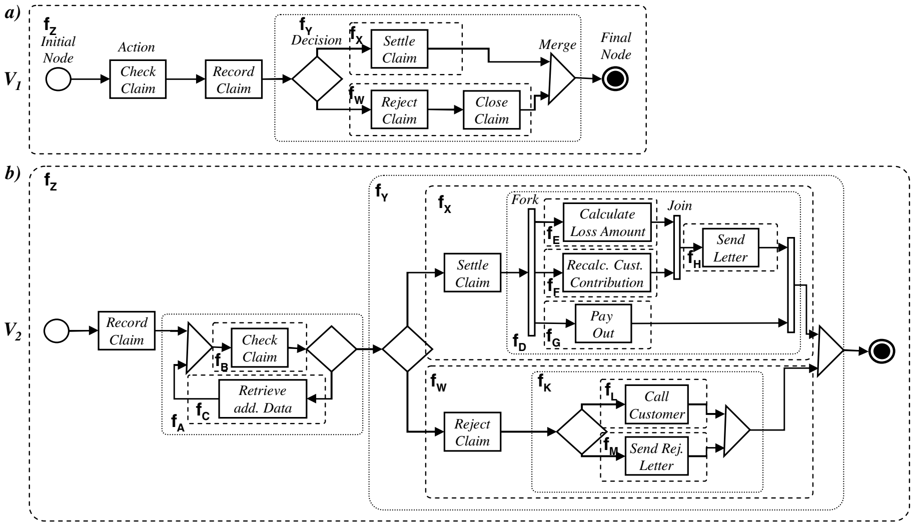


Fig. 4. Versions V_1 and V_2 decomposed into canonical SESE fragments

In general, process models can be decomposed into SESE fragments [20]. A SESE fragment is a non-empty subgraph in the process model with a single entry and a single exit edge. The fragment which surrounds the entire process model is also considered as a SESE fragment which we refer to as *root fragment*. Among all possible SESE fragments, we select so-called canonical fragments which are not overlapping on the same hierarchical level and denote them with $\mathcal{F}(V)$ for a given process model V. Figure 4 shows an example of a SESE decomposition into canonical fragments, with fragments visualized by a surrounding of dotted lines.

The canonical fragments of a process model V can be organized into a process structure tree (PST) [20], denoted by $PST(V)$, according to the composition hierarchy of the fragments (see Figure 5 for the tree obtained for V_1). If a fragment f_1 contains another fragment f_2 (respectively node n), then f_1 will be the parent of fragment f_2 (node n) in this tree and fragment f_2 (node n) will be one of its children. Further, the root of the tree is the root fragment. We distinguish between different types of fragments as follows [20]:

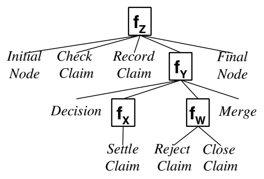


Fig. 5. Process structure tree of V_1

- a well-structured fragment f is either a sequential, a sequential branching, a cyclic, or a concurrent branching fragment. For example, in Figure 4, f_Y is a well-structured sequential branching fragment and f_W is a well-structured sequential fragment.
- an unstructured concurrent fragment f is not well-structured and contains no cycles and has no decisions and no merges as children. In Figure 4, f_D is an unstructured concurrent fragment.
- an unstructured sequential fragment f is not well-structured and has no forks and no joins as children.
- a complex fragment f is any other fragment that is none of the above.

Given a fragment $f \in \mathcal{F}(V)$, we denote by $type(f)$ the type of the fragment and by $frag(f)$ the parent fragment. Similarly, given a node $x \in N$, we denote by $type(x)$ the type of the node and $frag(x)$ the parent fragment of x. For example, $type(x) = Action$ means that x is an *Action* node.

SESE fragments have been used successfully for checking soundness [18,14] of process models but they are also beneficial for detection of differences between process models, discussed later in this paper.

3.2 Correspondences

Correspondences are useful for the detection of differences because they provide the link between elements in different process models. We assume process models $V_1 = (N_1, E_1)$ and $V_2 = (N_2, E_2)$ and $x \in N_1$ and $y \in N_2$ as given. A correspondence is used to express that a model element x has a counterpart y with the same functionality in the other version. In such a case, we introduce a 1-to-1 correspondence between them. In the case that a model element x does not have a counterpart with the same functionality, we speak of a 1-to-0 correspondence. In case that y does not have a counterpart, we speak of a 0-to-1 correspondence. In addition, refinement of an element into a set of elements would give rise to a 1-to-many correspondence and abstraction of a set of elements into one element would give rise to a many-to-1 correspondence. These last two types are not needed in our scenario.

We express a 1-to-1 correspondence by inserting the tuple (x, y) into the set of correspondences $\mathcal{C}(V_1, V_2) \subseteq N_1 \times N_2$. We further introduce the set of elements in V_1 which do not have a counterpart and denote this set by $\mathcal{C}_{1-0}(V_1, V_2)$. Similarly, we denote the set of elements in V_2 without counterparts as $\mathcal{C}_{0-1}(V_1, V_2)$. Similarly, we introduce correspondences for SESE fragments, expressed in the sets $\mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$, $\mathcal{C}^F_{1-0}(\mathcal{F}(V_1), \mathcal{F}(V_2))$, and $\mathcal{C}^F_{0-1}(\mathcal{F}(V_1), \mathcal{F}(V_2))$.

In our scenario, we assume that the functionality of a node remains the same if it is copied. Correspondences can then be computed in a straightforward way by first establishing 1-to-1 correspondences between all nodes [1] (respectively fragments) of a process model when copying process model V_1 to create an initial V_2. After obtaining the final V_2 by editing operations, all 1-to-1 correspondences have to be inspected and 1-to-0 or 0-to-1 correspondences are created if nodes (respectively fragments) have

[1] Correspondences are internally based on the unique identifiers of elements.

been deleted or added. In addition, for new nodes (respectively fragments) in V_2, additional 0-to-1 correspondences have to be created. In other scenarios across tool boundaries, other means of correspondence computation are required which might involve semantic matching techniques.

Fig. 6 shows correspondences between versions V_1 and V_2 of the process model introduced earlier in this paper. A dotted line represents 1-to-1 correspondences and connects model elements with the same functionality between V_1 and V_2. 1-to-0 correspondences are visualized by dotted elements in V_1 and 0-to-1 correspondences are visualized by dotted elements in V_2.

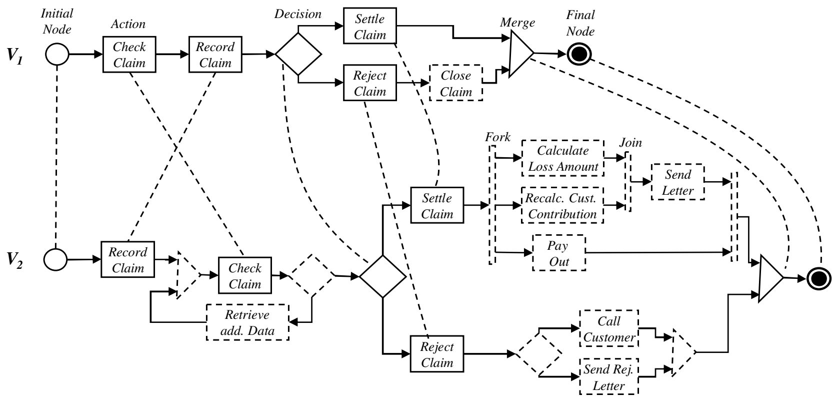


Fig. 6. Correspondences between nodes of V_1 and V_2

We further assume a partial ordering relation on actions and fragments of a process model, restricted to a partial ordering within each fragment. The partial orders will be later used for detecting moved elements. Given an element x (fragment or action), we denote by $order_f$ the partial order of elements in fragment f derived by the control flow order of elements within f and we write $x <_f y$ for x smaller than y according to this order[2]. Let a tuple of actions or fragments $(x, y) \in \mathcal{C}(V_1, V_2) \cup \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ and $frag(x) = f_1$ and $frag(y) = f_2$ be given. Then we write $order_{f_1}(x) \neq order_{f_2}(y)$ if and only if there exists an element $(z_1, z_2) \in \mathcal{C}(V_1, V_2) \cup \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ with $frag(z_1) = f_1$ and $frag(z_2) = f_2$ such that $z_1 <_{f_1} x$ and $z_2 >_{f_2} y$, or $z_1 >_{f_1} x$ and $z_2 <_{f_2} y$, or z_1 and x are unordered and z_2 and y are ordered, or z_1 and x are ordered and z_2 and y are unordered.

4 Detection of Differences

In this section, we describe an approach to detect differences between process models, based on the existence of correspondences and SESE fragments.

[2] For cyclic fragments, we assume an order obtained by a depth-first search of the fragment along the control flow edges.

4.1 Action and Fragment Differences and Change Operations

The correspondences between two process models can be used to identify differences. One form of differences that can occur are those that result from adding, deleting or moving actions, as defined in the following:

Definition 1 (Action Differences). *Given two business process models V_1, V_2 and sets of correspondences $\mathcal{C}_{1-0}(V_1,V_2)$, $\mathcal{C}_{0-1}(V_1,V_2)$ and $\mathcal{C}(V_1,V_2)$, we define the following action differences:*

- *an InsertAction difference is defined as an element $y \in \mathcal{C}_{0-1}(V_1,V_2)$ and $type(y) =$ Action,*
- *a DeleteAction difference is defined as an element $x \in \mathcal{C}_{1-0}(V_1,V_2)$ and $type(x) =$ Action,*
- *a MoveAction difference is defined as a tuple of actions $(x,y) \in \mathcal{C}(V_1,V_2)$ and either $(frag(x), frag(y)) \notin \mathcal{C}^F(\mathcal{F}(V_1),\mathcal{F}(V_2))$ or $((frag(x), frag(y)) \in \mathcal{C}^F(\mathcal{F}(V_1),\mathcal{F}(V_2))$ and $order_{frag(x)}(x) \neq order_{frag(y)}(y))$.*

The identification of *InsertAction* and *DeleteAction* differences is straightforward. With regards to *MoveAction* differences, we distinguish between *intra-fragment* differences where the action has been moved within corresponding SESE fragments and *inter-fragment* differences where actions have been moved between SESE fragments. The detection of inter-fragment differences can be done by iterating over all 1-to-1 correspondences and checking whether the surrounding SESE fragments are also in a 1-to-1 correspondence. If this is not the case, then the element has been moved and is considered as an inter-fragment difference. The detection of intra-fragment differences has to compare all elements within a fragment with the elements in the corresponding fragment and identify changes in the order of elements. Each action difference can be directly converted into a suitable *InsertAction*, *DeleteAction* or *MoveAction* operation which resolves the difference, shown in Figure 7. The position parameters a and b specify the position where action x is inserted or moved to in process model V_1.

Compound Change Operation applied on V	Effects on Process Model V
InsertAction(V,x,a,b)	Insertion of a new action x (by copying action y) between two succeeding elements a and b in process model V and reconnection of control flow.
DeleteAction(V,x)	Deletion of action x and reconnection of control flow.
MoveAction(V,x,a,b)	Movement of action x between two succeeding elements a and b in process model V and reconnection of control flow.

Fig. 7. Overview of compound change operations for actions

In addition to action differences, different versions of process models can also be constructed by introducing or removing control nodes, as well as deleting or changing edge connections involving such control nodes. These changes give rise to differences concerning the fragment structure of the process models and are defined as follows:

Definition 2 (Fragment Differences). *Given two business process models V_1 and V_2 and sets of correspondences between SESE fragments $\mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$, $\mathcal{C}^F_{1-0}(\mathcal{F}(V_1), \mathcal{F}(V_2))$, and $\mathcal{C}^F_{0-1}(\mathcal{F}(V_1), \mathcal{F}(V_2))$, we define the following fragment differences:*

- *an InsertFragment difference is defined as a SESE fragment $f_2 \in \mathcal{C}^F_{0-1}(\mathcal{F}(V_1), \mathcal{F}(V_2))$.*
- *a DeleteFragment difference is defined as a SESE fragment $f_1 \in \mathcal{C}^F_{1-0}(\mathcal{F}(V_1), \mathcal{F}(V_2))$.*
- *a MoveFragment difference is defined as a tuple of fragments $(f_1, f_2) \in \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ and either $(frag(f_1), frag(f_2)) \notin \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ or $((frag(f_1), frag(f_2)) \in \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ and $order_{frag(f_1)}(f_1) \neq order_{frag(f_2)}(f_2))$.*
- *a ConvertFragment difference occurs if the type of the fragment has changed or f_1 has a control node as child that has no counterpart in f_2 or f_2 has a control node as child that has no counterpart in f_1.*

The identification of *InsertFragment*, *DeleteFragment* and *MoveFragment* differences is analogous to action differences. Identification of *ConvertFragment* differences involves iteration over all tuples $(f_1, f_2) \in \mathcal{C}^F(\mathcal{F}(V_1), \mathcal{F}(V_2))$ and examining whether the type of f_1 or f_2 has changed or whether one of the fragments has a control node as child that has no counterpart in the other fragment.

Each fragment difference described can be resolved by an appropriate change operation. An overview of the operations and their effect on a process model is given in Figure 8. Note that here for the *InsertFragment* difference a number of different change operations is given, inserting the fragment of suitable type into the process model. The type can be determined by inspecting the fragment in V_2.

Figure 9 shows one possible set of compound change operations obtained for the example earlier in this paper. Here, *InsertCyclicFragment*$(V_1, \_, \_, f_A)$ will insert a new cycle into V_1, *MoveAction*$(V_1,$ *"Check Claim"*$, ...)$ moves *"Check Claim"* to its

Compound Change Operation applied on V	Effects on Process Model V
InsertFragment(V,a,b,f_2) *The generic operation InsertFragment is realized by:* • *InsertParallelFragment(V,a,b,f_2)* • *InsertAlternativeFragment(V,a,b,f_2)* • *InsertSequentialFragment(V,a,b,f_2)* • *InsertCyclicFragment(V,a,b,f_2)* • *InsertUnstructuredConcurrentFragment(V,a,b,f_2)* • *InsertUnstructuredSequentialFragment(V,a,b,f_2)* • *InsertComplexFragment(V,a,b,f_2)*	Insertion of a new fragment f_1 between two succeeding elements a and b in process model V, copying the structure of f_2, and reconnection of control flow.
DeleteFragment(V,f_1)	Deletion of fragment f_1 from process model V and reconnection of control flow.
MoveFragment(V,f_1,a,b)	Move of fragment f_1 between two succeeding elements a and b in process model V and reconnection of control flow.
ConvertFragment(V,f_1,f_2)	Conversion of a fragment f_1 into the fragment type of f_2, replacing the structure from f_1 with the structure of f_2, and reconnection of control flow.

Fig. 8. Overview of compound change operations for fragments

- InsertCyclicFragment(V_1, _, _, f_A)
- MoveAction(V_1,"Check Claim", _, _)
- InsertAction(V_1,"Retrieve add. Data", _, _)
- InsertUnstr.Conc.Fragment(V_1, _, _, f_D)
- InsertAction(V_1,"Calc. Loss Amount", _, _)
- InsertAction(V_1,"Recalc. Cust. Contr.", _, _)
- InsertAction(V_1,"Pay Out", _, _)
- InsertAction(V_1,"Send Letter", _, _)
- InsertAlternativeFragment(V_1, _, _, f_K)
- InsertAction(V_1,"Call Customer", _, _)
- InsertAction(V_1,"Send Rej. Letter", _, _)
- DeleteAction(V_1,"Close Claim")

Fig. 9. Compound change operations that transfer V_1 into V_2

- **InsertCyclicFragment**(V_1,"Record Claim", Decision, f_A)
- MoveAction(V_1,"Check Claim", _, _)
- InsertAction(V_1,"Retrieve add. Data", _, _)
- **InsertUnstr.Conc.Fragment**(V_1,"Settle Claim", Merge, f_D)
- InsertAction(V_1,"Calc. Loss Amount", _, _)
- InsertAction(V_1,"Recalc. Cust. Contr.", _, _)
- InsertAction(V_1,"Pay Out", _, _)
- InsertAction(V_1,"Send Letter", _, _)
- **InsertAlternativeFragment**(V_1,"Reject Claim", Merge, f_K)
- InsertAction(V_1,"Call Customer", _, _)
- InsertAction(V_1,"Send Rej. Letter", _, _)
- **DeleteAction**(V_1,"Close Claim")

Fig. 10. Compound change operations with position parameters

new position, and *InsertAction*(V_1, *"Retrieve add. Data"*, ...) will insert another action. Note that *Insert* and *Move* operations are still incomplete because the position parameters have not been specified. In general, if the position parameters of an operation are determined, we call this operation applicable. The computation of position parameters will be discussed in the following subsection.

4.2 Computation of Position Parameters

According to our requirements, differences between two versions of a process model should be resolvable in an arbitrary way which depends on the position parameters.

Figure 11 shows a simple example where two actions *"A3"* and *"A4"* have been inserted, leading to *InsertAction(V_1,"A3", _, _)* and *InsertAction(V_1,"A4", _, _)*. In order to ensure that the business user can choose both operations, we compute position parameters to be *InsertAction(V_1,"A3","A1","A2")* and *InsertAction(V_1,"A4","A1","A2")*. If either *"A3"* or *"A4"* were position parameters, this would induce a dependency between them, requiring that one of them is applied before the other one.

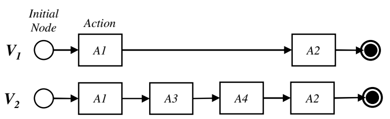


Fig. 11. Simple example

In order to avoid such situations, we express position parameters in terms of *fixpoints* for given process models V_1 and V_2 and a set of correspondences $\mathcal{C}(V_1, V_2)$. A *fixpoint pair* is a pair of nodes $(n_1, n_2) \in \mathcal{C}(V_1, V_2)$ such that n_1 and n_2 are not moved in the process models by any change operation that has been derived from the differences between V_1 and V_2. Given a fixpoint pair (n_1, n_2), both n_1 and n_2 are called fixpoints. For example, in Figure 11, both *("A1","A1")* and *("A2","A2")* are fixpoint pairs. Using fixpoints as position parameters also ensures that the insert and move operations can always produce a model that is connected because the newly inserted or moved element or fragment can be connected to the fixpoints automatically.

Figure 10 shows the operations with position parameters computed (bold printed operations are applicable, others are not yet applicable and need position parameters). After applying an operation, the set of fixpoints increases and the position parameters are recomputed, making more operations applicable.

In the following, we first reason about the completeness of change operations derived using our approach.

4.3 Completeness of Change Operations

The set of change operations containing all compound change operations for two business process models V_1 and V_2 derived according to our approach is denoted by $Changes_{Compound}(V_1, V_2)$. After defining action and fragment differences, one question to ask is whether these are all differences that can occur when changing a process model V_1 into a process model V_2. For this, we assume an ideal minimal change log consisting of change primitives [17] inserting or deleting nodes (Action or ControlNodes) and edges, namely *addActionNode*, *addControlNode*, *addEdge*, *deleteActionNode*, *deleteControlNode*, *deleteEdge*.

Given process models V_1 and V_2, a minimal sequence of primitive change operations op_i converting V_1 into V_2, denoted as $ChangeLog_{min}(V_1, V_2)$, is called a minimal change log. Given such a minimal change log, we have to show that each entry in this change log gives rise to a compound change operation involving actions or fragments, so no entry will be ignored. The following theorem establishes a relationship between the minimal change log and our compound change operations:

Theorem 1 (Completeness of Differences). *Given two business process models V_1, V_2, $ChangeLog_{min}(V_1, V_2)$ and $Changes_{Compound}(V_1, V_2)$, for each $op \in ChangeLog_{min}(V_1, V_2)$ there exists $c \in Changes_{Compound}(V_1, V_2)$ such that c comprises op.*

Proof sketch: Given $op \in ChangeLog_{min}(V_1, V_2)$:

– If $op = addActionNode$, then there exists $y \in \mathcal{C}_{0-1}(V_1, V_2)$ which gives rise to an InsertAction difference which means that we derive an InsertAction operation c comprising op.
– If $op = addControlNode$, then the control node either creates a new fragment or is inserted into an existing fragment. The first case induces an InsertFragment difference, the second case a ConvertFragment difference. In both cases, c (InsertFragment or ConvertFragment) comprises op.
– If $op = addEdge$, then this involves integrating new nodes (ControlNodes or Actions) into the process model, reordering of existing fragments or nodes, or reconnection of existing nodes in case of deletions. In the first case, there must be a suitable addActionNode or addControlNode operation, leading to appropriate *Insert* operations comprising op. In the second case, the addEdge operation (possibly together with other addEdge operations) gives rise to MoveAction or MoveFragment differences comprising op. In the third case, there must be a suitable DeleteAction or DeleteFragment operation comprising op.
– The cases $op = deleteActionNode$, $op = deleteControlNode$ and $deleteEdge$ can be treated analogously.

This result shows that it is possible to detect differences based on actions and fragments. Note that each fragment difference usually involves more than one control node or edge difference and gives rise to the possibility to abstract from several individual differences relating to edge reconnection or control node changes.

5 Computation of Hierarchical Change Log

In order to enable user-friendly resolution of changes, change operations can be visualized according to the structure of the two process models which is obtained by their SESE decomposition: Given such a fragment decomposition, each operation can be associated to the fragment in which it occurs. In the following, we first introduce a joint process structure tree as a basis of such a hierarchical change log. Given two process structure trees $PST(V_1)$, $PST(V_2)$ and correspondences between their nodes, then the joint PST is denoted as $PST(V_1,V_2)$. The joint PST can be constructed as follows:

– for a pair $(v_1,v_2) \in \mathcal{C}^F(V_1,V_2)$, a new node v_3 is inserted into $PST(V_1,V_2)$ with $fragment(v_3) = fragment(v_1)$.
– for a node $v_1 \in \mathcal{C}^F_{1-0}(V_1,V_2)$, a new node v_3 is inserted into $PST(V_1,V_2)$ with $fragment(v_3) = fragment(v_1)$,
– for a node $v_2 \in \mathcal{C}^F_{0-1}(V_1,V_2)$, a new node v_3 is inserted into $PST(V_1,V_2)$ with $fragment(v_3) = fragment(v_2)$.

Based on the joint PST, we can define a hierarchical change log. The idea of the hierarchical change log is to arrange change operations according to the structure of the process model by associating to each SESE fragment the change operations that affect it (for a formal definition see [10]).

Figure 12 shows a hierarchical change log for the two versions V_1 and V_2 of a process model introduced earlier in this paper. For example, the

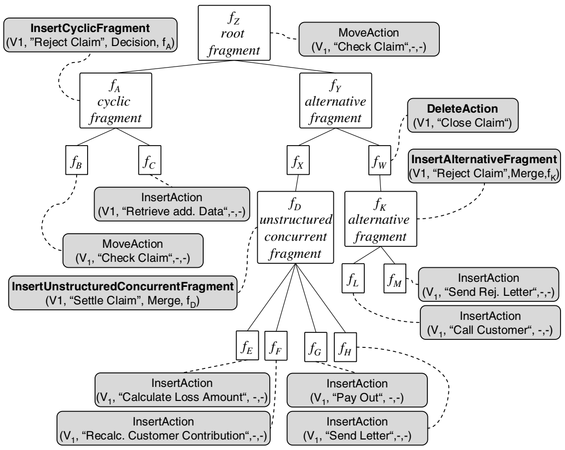


Fig. 12. Hierarchical change log of example

InsertCyclicFragment operation takes place within the root fragment f_Z. The *InsertUnstructuredConcurrentFragment* occurs in the branch f_X of the alternative fragment f_Y. Within the newly inserted unstructured parallel fragment f_D, there are several *InsertAction* operations. Further operations such as the *InsertAlternativeFragment* or *DeleteAction* operations are also associated to their fragments.

Using the hierarchical change log, one can easily identify the areas of the process model that have been manipulated. This enables the business user to concentrate on those changes that are relevant to a certain area in the process model and increases thereby usability. The hierarchical change log can also be beneficial for identifying dependencies between change operations and for identifying groups of change operations that can be applied as a change transaction. In the next section, we elaborate on the application of change operations and tool support.

6 Application of Operations and Tool Support

The operations in the change log with position parameters are ready for application. Figure 13 shows V_1 and the application of the *InsertUnstructuredConcurrentFragment* operation, leading to the insertion of the *Fork* and two *Joins* and the automated reconnection of control flow. Alternatively, the user could have chosen any other operation of the change log that is applicable.

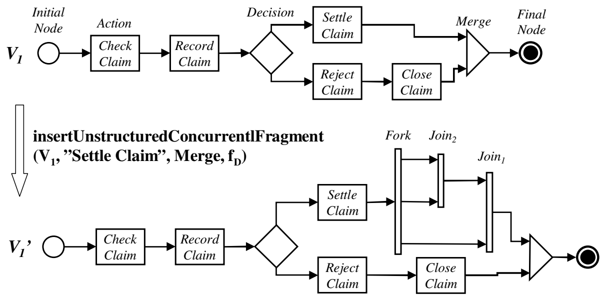


Fig. 13. Applying a compound change operation

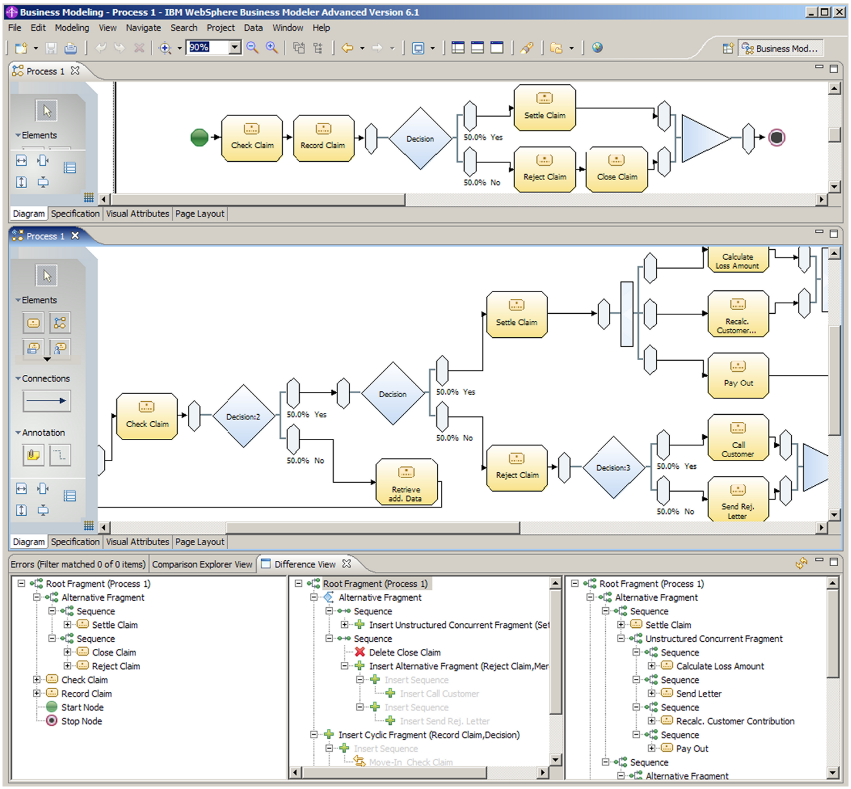
By recomputing the position parameters of the remaining operations we can increase the number of applicable operations and refine existing position parameters. In the example, the position parameters of the *InsertAction(V_1,"Calculate Loss Amount", _, _)*, *InsertAction(V_1,"Recalc. Customer Contribution", _, _)*, *InsertAction(V_1,"Pay Out", _, _)*, and *InsertAction(V_1,"Send Letter", _, _)* operations can be computed after inserting the unstructured concurrent fragment. This leads to a new change log which is shown in Figure 14.

As proof of concept, we have implemented a prototype as an extension to the IBM WebSphere Business Modeler [1] (see Fig. 15), including functionality for creation of correspondences when copying a process model, decomposition of process models into SESE fragments and detection and resolution of differences.

Root Fragment
- **InsertCyclicFragment**(V_1,"Record Claim", Decision, f_A)
  - ***Move***(V_1,"Check Claim", _, _)
  - ***InsertAction***(V_1,"Retrieve add. Data", _, _)
- Alternative Fragment
  - Unstructured Concurrent Fragment
    - **InsertAction**(V_1,"Calculate Loss Amount", Fork, Join_2)
    - **InsertAction**(V_1,"Recalc. Cust. Contrib.", Fork, Join_2)
    - **InsertAction**(V_1,"Pay Out", Fork, Join_1)
    - **InsertAction**(V_1,"Send Letter", Join_2, Join_1)
  - **Delete**(V_1,"Close Claim")
  - **InsertAlternativeFragment**(V_1,"Reject Claim", Merge, f_K)
    - ***InsertAction***(V_1,"Call Customer", _, _)
    - ***InsertAction***(V_1,"Send Rej. Letter", _, _)

Fig. 14. Recomputed change log after applying a compound operation

Fig. 15 shows versions V_1 and V_2 of the business process model introduced earlier in this paper. The lower third of Fig. 15 illustrates the Difference View, which is divided into three columns. The left and right hand columns show versions V_1 and V_2 of the process model decomposed into SESE fragments. The middle column of the difference view displays the hierarchical change log as introduced previously. Using our prototype, differences between the two versions can be iteratively resolved using the change operations introduced in this paper.


Business Modeling - Process 1 - IBM WebSphere Business Modeler Advanced Version 6.1
File Edit Modeling View Navigate Search Project Data Window Help
Process 1
Elements
Check Claim
Record Claim
Decision
50.0% Yes
50.0% No
Settle Claim
Reject Claim
Close Claim
Diagram Specification Visual Attributes Page Layout
Connections
Annotation
Decision:2
Retrieve add. Data
Calculate Loss Amount
Recalc. Customer...
Pay Out
Decision:3
Call Customer
Send Rej. Letter
Errors (Filter matched 0 of 0 items) Comparison Explorer View Difference View

```
Root Fragment (Process 1)
  Alternative Fragment
    Sequence
      Settle Claim
    Sequence
      Close Claim
      Reject Claim
  Check Claim
  Record Claim
  Start Node
  Stop Node
```

```
Root Fragment (Process 1)
  Alternative Fragment
    Sequence
      Insert Unstructured Concurrent Fragment (Set
    Sequence
      Delete Close Claim
      Insert Alternative Fragment (Reject Claim,Mer
        Insert Sequence
          Insert Call Customer
        Insert Sequence
          Insert Send Rej. Letter
  Insert Cyclic Fragment (Record Claim,Decision)
    Insert Sequence
      Move-In Check Claim
```

```
Root Fragment (Process 1)
  Alternative Fragment
    Sequence
      Settle Claim
      Unstructured Concurrent Fragment
        Sequence
          Calculate Loss Amount
        Sequence
          Send Letter
        Sequence
          Recalc. Customer Contribution
        Sequence
          Pay Out
    Sequence
      Alternative Fragment
```

Fig. 15. Business Process Merging Prototype in the IBM WebSphere Business Modeler

As an initial validation, we applied our approach to the IBM Insurance Application Architecture (IAA) [9]. We focused on claim handling, a key process in insurance industry, which is modeled as a composition of several other processes in IAA and is far to large to identify differences without tool support. We introduced a set of differences in the claim handling process and its subprocesses. Using our prototype, users who were unaware of the differences, were able to identify the differences easily and additionally were able to resolve selected differences in order to create a consolidated version of the process model.

7 Related Work

Within the workflow community, the problem of migrating existing workflow instances to a new schema [3,15] has received considerable attention: Given a process schema change, it can be distinguished between process instances that can be migrated and those that cannot be migrated [16]. Rinderle et al. [16] describe migration policies for the situation that both the process instance as well as the process type has been changed. They introduce a selection of change operations and examine when two changes are commutative, disjoint or overlapping. Recent work by Weber et al. [21] provides a comprehensive overview of possible change patterns that can occur when process models or process instances are modified. These change patterns are used for evaluating different process-aware information systems [5] with respect to change support. Zhao and Liu [22] address the problem of versioning for process models and present a means of storing all versions in a version preserving graph, also relying on the existence of a change log. Grossmann et al. [7] show how two business processes can be integrated using model transformations after relationships have been established. Both the change operations by Rinderle et al. [16] and the change patterns for inserting, deleting and moving a process fragment are similar to our change operations. In contrast to the existing work, we describe an approach how to identify change operations in the case that no change log is available and how to use SESE fragments for ordering discovered change operations.

In the context of process integration where models do not originate from a common source model, Dijkman [4] has categorized differences of process models and van Dongen et al. [19] have developed techniques for measuring the similarity of process models. This work is complementary to our work where we assume that correspondences can be automatically derived. In process integration, correspondences must first be established using semantic matching techniques (see e.g. [6]) before our techniques for detecting differences can be applied.

In model-driven engineering, generic approaches for detecting model differences and resolving them have been studied [2,8]. Alanen and Porres [2] present an algorithm that calculates differences of two models based on the assumption of unique identifiers. The computation of differences has similarities to our reconstruction of change operations, however, in our work, we aim at difference resolution for process models with minimal user interaction.

8 Conclusion and Future Work

Detecting and resolving process model differences represents a standard functionality in process-aware information systems that provide a change log. In this paper, we have presented an approach for the detection and resolution of differences in the absence of a change log that is based on correspondences between process models and also makes use of the concept of a SESE fragment decomposition of process models. This SESE decomposition enables the detection of compound changes and the visualization of differences according to the structure of process models. The resolution of differences is performed in an iterative way, by applying change operations that automatically reconnect the control flow.

There are several directions for future work. The change operations are intended to preserve well-formedness and soundness of the process model which needs to be formally proven. Further work is needed to support all features of process models such as data flow. Afterwards, a detailed validation of our approach can be performed to show that dealing with compound changes saves time during modeling. Future work will also include the elaboration of our approach for merging process models in a distributed environment. In those scenarios, the concept of a conflict becomes important because one resolution can turn the other resolution non-applicable.

Acknowledgements. We would like to thank Jana Koehler, Ksenia Ryndina, and Olaf Zimmermann for their valuable feedback on an earlier version of this paper.

References

1. IBM WebSphere Business Modeler, http://www.ibm.com/software/integration/wbimodeler/
2. Alanen, M., Porres, I.: Difference and Union of Models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 2–17. Springer, Heidelberg (2003)
3. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data Knowl. Eng. 24(3), 211–238 (1998)
4. Dijkman, R.: A Classification of Differences between Similar Business Processes. In: EDOC 2007, pp. 37–50. IEEE Computer Society, Los Alamitos (2007)
5. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems. Wiley, Chichester (2005)
6. Grigori, D., Corrales, J., Bouzeghoub, M.: Behavioral matchmaking for service retrieval. In: ICWS 2006, pp. 145–152. IEEE Computer Society, Los Alamitos (2006)
7. Grossmann, G., Ren, Y., Schrefl, M., Stumptner, M.: Behavior Based Integration of Composite Business Processes. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 186–204. Springer, Heidelberg (2005)
8. Herrmann, C., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: An Algebraic View on the Semantics of Model Composition. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA 2007. LNCS, vol. 4530. Springer, Heidelberg (2007)
9. IBM Insurance Application Architecture, http://www.ibm.com/industries/financialservices/iaa
10. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Process Merging in Business-Driven Development. IBM Research Report RZ 3703, IBM Zurich Research Laboratory (2008)
11. Mitra, T.: Business-driven development. IBM developerWorks article, IBM (2005), http://www.ibm.com/developerworks/webservices/library/ws-bdd
12. Object Management Group (OMG). The Unified Modeling Language 2.0 (2005)
13. Pottinger, R., Bernstein, P.A.: Merging Models Based on Given Correspondences. In: VLDB, pp. 826–873 (2003)
14. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 145–160. Springer, Heidelberg (2006)
15. Reichert, M., Dadam, P.: ADEPT$_{\text{flex}}$-Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inf. Syst. 10(2), 93–129 (1998)
16. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 101–120. Springer, Heidelberg (2004)
17. Rinderle, S., Reichert, M., Jurisch, M., Kreher, U.: On Representing, Purging, and Utilizing Change Logs in Process Management Systems. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 241–256. Springer, Heidelberg (2006)
18. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)

19. van Dongen, B., Dijkman, R., Mendling, J.: Measuring Similarity between Business Process Models. In: CAiSE 2008, pp. 450–464. Springer, Heidelberg (2008)
20. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
21. Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
22. Zhao, X., Liu, C.: Version Management in the Business Process Change Context. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 198–213. Springer, Heidelberg (2007)

Diagnosing Differences between Business Process Models

Remco Dijkman

Eindhoven University of Technology, The Netherlands
r.m.dijkman@tue.nl

Abstract. This paper presents a technique to diagnose differences between business process models in the EPC notation. The diagnosis returns the exact position of a difference in the business process models and diagnoses the type of a difference, using a typology of differences developed in previous work. This in contrast to existing techniques for detecting process differences (by showing non-equivalence), which return simple true/false statements, or statements in terms of a formal semantics. Neither type of statement is helpful to a business analyst not versed in formal semantics. A case study illustrates the usefulness of the technique. It also shows that, although the technique has exponential complexity, it can be used in practice, because of repeated scoping of the models. The technique can be used, for example, to resolve differences between operational process in a merger between organizations.

1 Introduction

This paper presents a technique to diagnose differences between business processes. Such a technique is useful when, for example, faced with the task of merging (the business processes of) two organizations or the task of determining to what extent a business process deviates from a reference process.

Two business processes contain one or more differences if and only if they are not equivalent according to some notion of equivalence [9]. Therefore, techniques to determine equivalence play an important role in this paper. However, these techniques return information about non-equivalence of business processes in terms of the formal semantics of those processes (i.e. the state-space or the set of traces of those processes). While such information is useful for someone versed in formal methods it is less useful for the average business process analyst.

Therefore, the goal and contribution of this paper is a technique to diagnose differences between business processes, in such a way that a business process analyst, without training in formal methods, can understand the diagnosis. The diagnosis provides feedback about differences between processes, by showing exactly where they are in the processes and by identifying the types of the differences, using a typology developed in previous work [4]. The paper focuses on business processes modelled in the Event-Driven Process Chain (EPC) notation. Feedback is provided in terms of this notation, such that it is understandable by the business process analyst who is familiar with this notation.

The usefulness of the technique is illustrated by application to model pairs from the SAP reference model, which contains 604 business processes in the EPC notation.

This paper builds on results from previous work [4,6]. It extends the difference typology from [4] by formalizing the differences and developing an algorithm to detect the differences. The work described in [6] is complementary to this paper, because this paper focuses on pointing out differences, while [6] focuses on measuring the degree of similarity in terms of a number between 0 and 1.

The remainder of this paper is organized as follows. Section 2 introduces the EPC notation as well as a formal semantics and the formal notion of (in)equivalence that we use to determine equivalence between EPCs. Section 3 explains the technique. Section 4 shows the results of applying the technique in the context of the SAP reference model. Section 5 discusses related work and section 6 concludes.

2 Event-Driven Process Chains

Event-driven process chains (EPCs) [10] can be used to model business processes. Below, we define their syntax and semantics and we explain what feedback on differences between EPCs should look like.

2.1 EPC Syntax and Information Semantics

An EPC can be used to represent the flow of control in a process. It consists of *functions* that represent activities in a process, *events* that represent the preconditions and postconditions of functions and *connectors* that influence the flow of control in a process. These different types of *nodes* are connected by *arcs*. Three types of connectors are distinguished: AND ($\wedge$), XOR ($\times$) and OR ($\vee$). Each function has exactly one incoming and one outgoing arc. Each event has at most one incoming and at most one outgoing arc, but at least one arc. An event with no incoming arc is a *start event* and an event with no outgoing arc is an *end event*. Each connector either has one incoming arc and multiple outgoing arcs (*split connectors*) or multiple incoming arcs and one outgoing arc (*join connectors*). On each path of arcs functions and events must alternate, while connectors can appear on the path on arbitrary places.

Informally, the semantics of an EPC can be described as follows. An AND-split waits to get the control flow on its incoming arc before allowing the control flow to continue on all its outgoing arcs. An AND-join waits to get the control flow on all its incoming arcs before allowing the control flow to continue on its outgoing arc. An XOR-split waits to get the control flow on its incoming arc before allowing the control flow to continue on one of its outgoing arcs. An XOR-join waits to get the control flow on one of its incoming arcs before allowing it to continue on its outgoing arc. An OR-split waits to get the control flow on its incoming arc before allowing it to continue on one or more of its outgoing arcs. An OR-join waits to get the control flow on all incoming arcs that *can* get the control flow. It then allows the control flow to continue on its outgoing arc.

EPC functions can be decomposed into sub-processes. However, this is only a syntactical matter. Therefore, without loss of generality, we focus on *flat* EPCs, which do not contain sub-processes. Formally, an EPC is defined as follows.

Definition 1 (EPC). An flat EPC is a five-tuple (E, F, C, l, A) in which:

- E is the set of events ($E \neq \varnothing$);
- F is the set of functions ($F \neq \varnothing$, $F \cap E = \varnothing$);
- C is the set of connectors ($C \cap E = \varnothing$, $C \cap F = \varnothing$);
- $l : C \rightarrow \{and, or, xor\}$ labels connectors with their type; and
- $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$ is the set of arcs.

Definition 2 (Preset, Postset, Start and End Events). Let $N = E \cup F \cup C$ be the set of all nodes. The preset of a node n (denoted $\bullet n$) is the set of nodes from which there is an arc to n: $\bullet n = \{n' | (n', n) \in A\}$. Similarly, the postset of n (denoted $n\bullet$) is the set of nodes to which there is an arc from n: $\bullet n = \{n' | (n, n') \in A\}$. Start events ($E_s$) and end events ($E_e$) are $E_s = \{e \mid e \in E, |\bullet e| = 0\}$ and $E_e = \{e \mid e \in E, |e\bullet| = 0\}$.

Definition 3 (Paths and Chains). Let $N = E \cup F \cup C$ be the set of all nodes. For a, $b \in N$ a path $a \hookrightarrow b$ exists if and only if either $a = b$ or there exists a (possibly empty) collection of nodes $n_1, n_2, \ldots, n_k \in N$, such that $(a, n_1), (n_1, n_2), \ldots, (n_k, b) \in A$.

The correctness criterion that the technique in this paper requires is that an EPC can only start with events: $\{n \mid n \in N, |\bullet n| = 0\} \subseteq E$. See [15] for a more detailed list of correctness criteria for EPCs.

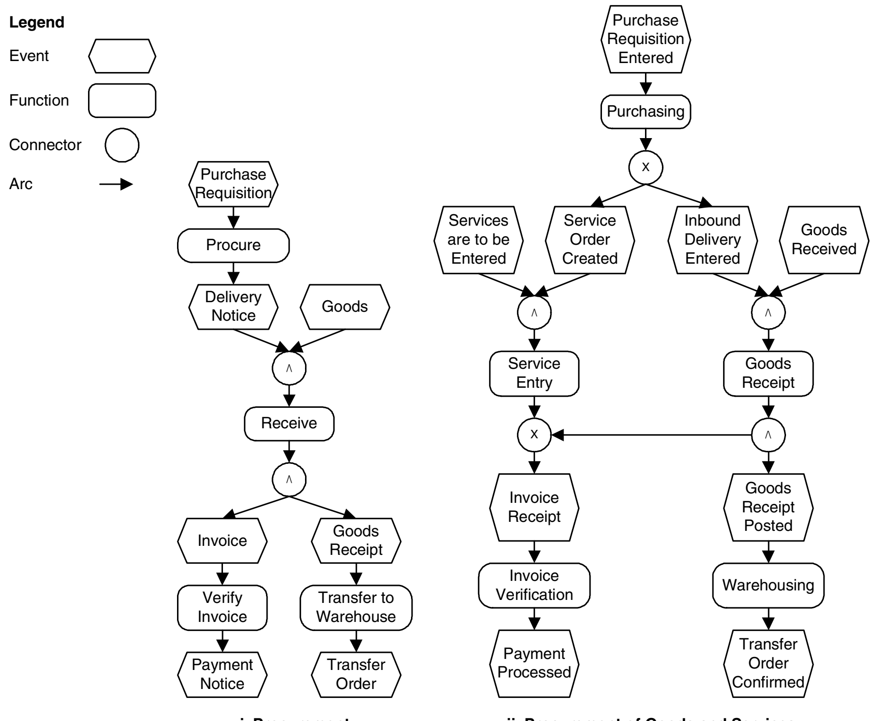


Fig. 1. Two EPCs

Figure 1 shows two examples of EPCs. One procurement process and one process for procuring both goods and services. The processes are represented graphically. Events are represented as hexagons, functions as rounded rectangles, connectors as circles and arcs as arrows. Upon reception of a *purchase requisition*, the procurement process starts to *procure* goods. After procurement is completed a *delivery notice* is created and when both this notice and the *goods* are received, it enters the administrative settlement of the goods reception. After that the invoice is verified and paid and the goods are transferred to the warehouse. The process for procuring both goods and services has a similar flow. However, the function and event names differ. Also, after purchasing goods or services, the process can continue to either receive goods or services (depending on what was ordered). After *goods receipt*, both *warehousing* and *invoice verification* is done, but after receiving services, only *invoice verification* is done.

2.2 EPC Formal Semantics

A formal semantics for EPCs can be described in terms of a transition system [11,13]. For ease of reading we use a simple semantics in this paper. However, the choice of semantics is not fundamental to the technique and it can easily be replaced by an advanced semantics (such as [11,13]), as long as it is based on a transition system. The difference between this semantics and a more advanced semantics is the semantics of the OR-join. An advanced semantics is able to decide whether an incoming arc to an OR-join *can* ever get the control. The semantics used here cannot. It uses a slightly different semantics for the OR-join: an OR-join waits to get the control flow on one or more incoming arcs. It then allows the control flow to continue on its outgoing arc. Such a semantics is also called a local semantics.

Definition 4 (Simple EPC Formal Semantics). We define the simple formal semantics of an EPC (denoted [EPC]) as a state-transition system $(S, \Sigma, \rightarrow, s_0, S_f)$ in which:

- S: $E_s \cup A \rightarrow \mathbb{N}$ is the set of states, defined as a marking of start events and arcs of the EPC with natural numbers;
- $\Sigma = N \cup \{\varepsilon\}$ is the alphabet, defined as a the set of nodes of the EPC and a special character $\varepsilon \notin N$ that represents that nothing of interest occurs;
- $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation, where $(s, a, s') \in \rightarrow$ (also written as $s \rightarrow^a s'$) represents that s can transition into s' when a occurs. The transition relation is defined by the transition rules from Figure 2. In this figure a black dot represents 'n+1' on a start event or arc. For example, the first rule represents that if, in state s, $s(e) \geq 1$ (because $s(e) = n+1$, $n \in \mathbb{N}$), then s can transition into a state s' when e occurs, where $s'(e) = s(e) - 1$ and $s'(arc) = s(arc) + 1$. (See [13] for a formalization of similar transition rules)
- $s_0 = E_s \times \{1\} \cup A \times \{0\}$ is the initial state, defined as the state in which all start events are marked with 1 and all arcs are marked with 0;
- $S_f = \{s \mid s \in S, \neg\exists s' \in S, a \in \Sigma: (s, a, s') \in \rightarrow\}$ is the set of final states, defined as the set of states from which no transition can be taken.

Consider the semantics of Figure 1.i as an example. Initially, the process is in a state in which the events 'Goods' and 'Purchase Requisition' are marked with 1 and all

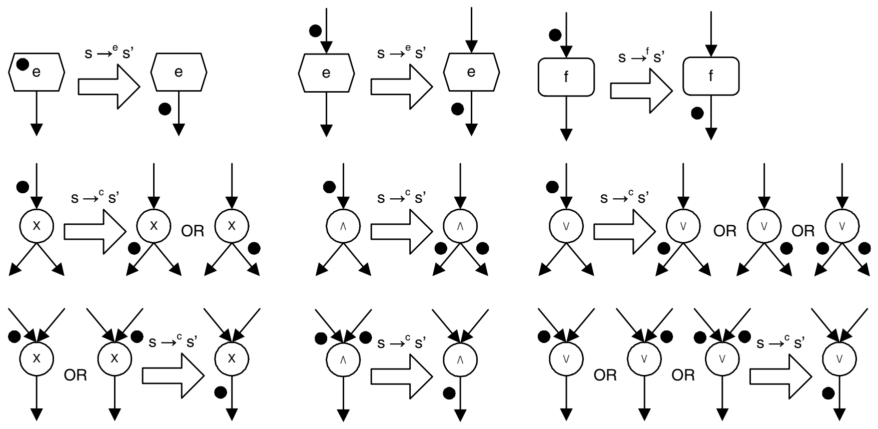


Fig. 2. Transition rules

arcs are marked with 0. It can then take a transition in which ‘Goods’ occurs and a transition in which ‘Purchase Requisition’ occurs, leading to states in which the arrows leaving the respective events are marked 1 and the events are marked 0. The and-join that has {‘Receive’} as its postset can occur if both the arrow leaving ‘Goods’ and the arrow leaving ‘Delivery Notice’ are marked greater than 0, so after both ‘Goods’ and ‘Delivery Notice’ have occurred.

The technique that diagnoses differences only works for EPCs for which the statespace is finite. For EPCs with a finite statespace, the transition system is a Non-deterministic Finite-state Automaton (NFA) with silent (ε) moves (NFA-ε) [19]. We use this property further on in this paper. We claim that the class of EPCs with a finite statespace is sufficiently interesting, in particular because none of the EPCs in our case study (see section 4) had an infinite statespace. The question whether the set of states is finite is a coverability problem and therefore decidable. The complexity of constructing the NFA-ε itself (and deciding whether the set of states is finite) is exponential. However, we will show below that by *restricting* an automaton to a subset of its alphabet (and subsequently reducing the restricted automaton) we can still use it practically.

Definition 5 (restriction). We can reduce an automaton $P = (S, \Sigma, \rightarrow, s_0, S_f)$ to a set of nodes $ns \subseteq \Sigma$ (denoted P / ns) by labelling nodes not in ns as ε, resulting in an automaton $(S, \Sigma', \rightarrow', s_0, S_f)$ in which:

- $\Sigma' = (\Sigma \cap ns) \cup \{\varepsilon\}$;
- $\rightarrow' = \{(s, n, s') | (s, n, s') \in \rightarrow, n \in ns\} \cup \{(s, \varepsilon, s') | (s, n, s') \in \rightarrow, n \notin ns\}$

Note that this semantics also works for *unsound* processes [1] (processes that contain a deadlock or livelock, do not terminate or leave dangling references).

2.3 Equivalence and Differences between EPCs

Two processes contain differences if and only if they are not equivalent. Therefore, it is necessary to use a notion of equivalence. We use the notion of completed trace equivalence. Stronger notions exist (such as the notion of 'branching bi-similarity' [9]). However, these notions can only return a single difference at a time, while we aim to return as many differences between the processes as possible. Researching differences with respect to stronger notions of equivalence remains for future work.

Notation. Given an automaton $(S, \Sigma, \rightarrow, s_0, S_f)$ we use the following notations:

- Σ^* is the set of sequences that can be formed from Σ
- $\varepsilon \in \Sigma^*$ is the empty sequence
- for $s, s' \in S$: $s \Rightarrow s'$ iff $s = s'$ or there exists an $s_1 \in S$ such that $s \rightarrow^{\varepsilon} s_1 \Rightarrow s'$
- for $s, s' \in S$, $a \in \Sigma$: $s \Rightarrow^{a} s'$ iff there exist $s_1, s_2 \in S$ such that $s \Rightarrow s_1 \rightarrow^{a} s_2 \Rightarrow s'$
- for $s, s' \in S$, $\sigma = a_1\, a_2\, a_3 \ldots a_n \in \Sigma^*$: $s \Rightarrow^{\sigma} s'$ iff
  there exist $s_1, s_2, \ldots \in S$ such that $s \Rightarrow^{a1} s_1 \Rightarrow^{a2} s_2 \Rightarrow^{a3} \ldots \Rightarrow^{an} s'$

Definition 6 (completed trace semantics). Given an automaton $P = (S, \Sigma, \rightarrow, s_0, S_f)$, the *completed trace semantics* of P, written $Tr(P)$, is defined as $Tr(P) = \{\sigma \in \Sigma^* \mid s_0 \Rightarrow^{\sigma} s_f, s_f \in S_f\}$.

Definition 7 (completed trace equivalence and differences). Two automata P and Q are completed trace-equivalent if and only if they have equivalent sets of completed traces $Tr(P) = Tr(Q)$. Clearly then two EPCs E_1 and E_2 are completed trace equivalent if and only if $Tr([E_1]) = Tr([E_2])$. They contain one or more differences if and only if $Tr([E_1]) \neq Tr([E_2])$.

For example, the completed trace semantics of the process in Figure 1.i, restricted to its functions is: {'Procure' 'Receive' 'Verify Invoice' 'Transfer to Warehouse', 'Procure' 'Receive' 'Transfer to Warehouse' 'Verify Invoice'}. ('Transfer to Warehouse' and 'Verify Invoice' can occur in arbitrary order because they are preceded by an and-split.) Clearly, this semantics is different from the semantics of Figure 1.ii, because the labels of the functions differ and because Figure 1.ii has an additional function 'Service Entry', which can occur in some of the traces.

3 Diagnosing Differences between EPCs

Diagnosing the differences between EPCs is done in five steps. This section explains these five steps successively. The technique is implemented as a plug-in in ProM[1].

3.1 Step 1: Pre-processing - Determine Function Completion Equivalences

In many realistic cases labels of equivalent functions will differ. Therefore, equivalences between functions must be determined manually. In particular the *completion equivalence relation* $R \subseteq F \times F$ must be defined, such that $f_1\ R\ f_2$ represents that the completion of a function f_1 coincides with the completion of a function f_2. In the remainder of the paper we use the reflexive, symmetric and transitive closure $(\sim\ \subseteq F \times F)$ of the completion equivalence relation.

[1] Freely available on www.processmining.org.

The completion equivalence relation can be inferred from a general *function correspondence relation*, which can be determined first by the business process analyst. The function correspondence relation is a relation $S \subseteq \wp F \times \wp F$, such that $fs_1\ S\ fs_2$ represents that fs_1 and fs_2 produce the same result and that there are no subsets of fs_1 and fs_2 for which that holds. Intuitively, fs_1 and fs_2 are equivalent sets of functions. R and S are related in the sense that for functions f_1 and f_2 that represent the completion of fs_1 and fs_2 for which $fs_1\ S\ fs_2$, it should hold that $fs_1\ R\ fs_2$. We call a function f that is not related by S (i.e. $f \notin \bigcup dom(S) \cup \bigcup ran(S)$) a *skipped function*, because it has no equivalent in the other process.

The business process analyst must determine these relations manually. However, he can be assisted in this task by techniques that can automatically determine similarity between functions. We developed such techniques earlier [6].

In Figure 1 the process analyst could, for example, relate ‘Procure’ to ‘Purchasing’, ‘Receive’ to ‘Goods Receipt’, ‘Verify Invoice’ to ‘Invoice Verification’ and ‘Transfer to Warehouse’ to ‘Warehousing’ both as being completion equivalent and as being corresponding. ‘Service Entry’ would then not be related to a function in the other process and therefore be a skipped function.

3.2 Step 2: Normalize EPCs

Normalization of a pair of EPCs resolves the differences between function labels, such that they can be compared using techniques that assume that function labels are syntactically the same for equivalent functions. In particular differences between function labels are resolved by replacing each function by its equivalence class.

Definition 8 (normalized EPC). The normalized behaviour of an $EPC = (E, F, C, l, A)$ induced by the reflexive, symmetric and transitive closure $\sim$ of the completion equivalence relation, denoted $[EPC]_\sim$, equals (E, F', C, l, A'), in which:

- $F' = \{[f]_\sim | f \in F\}$;
- $A' = \{(n_1', n_2') | (n_1, n_2) \in A,\ \ n_1' = [n_1]_\sim$ if $n_1 \in F$; $n_1' = n_1$ otherwise,
  $n_2' = [n_2]_\sim$ if $n_2 \in F$; $n_2' = n_2$ otherwise $\}$

In the normalized behaviour functions are treated the same if they are related by the equivalent completion occurrence relation, because functions are replaced by their equivalence class, which is by definition the same for equivalent functions.

For example, Figure 3 shows a part of the process models from Figure 1. In which functions have been replaced by their equivalence class. The figure shows that functions that have an equivalent in the other process (e.g. ‘procure’ and ‘purchasing’) are now the same in the sense that they have the same label, while functions that do not have an equivalent (e.g. ‘service entry’) will be treated differently.

3.3 Step 3: Restrict EPCs to Compare a Class of Equivalent Functions

The next steps are to focus on (a class of) equivalent functions and to compute the differences with respect to those functions (steps 4 and 5). We repeat these steps for

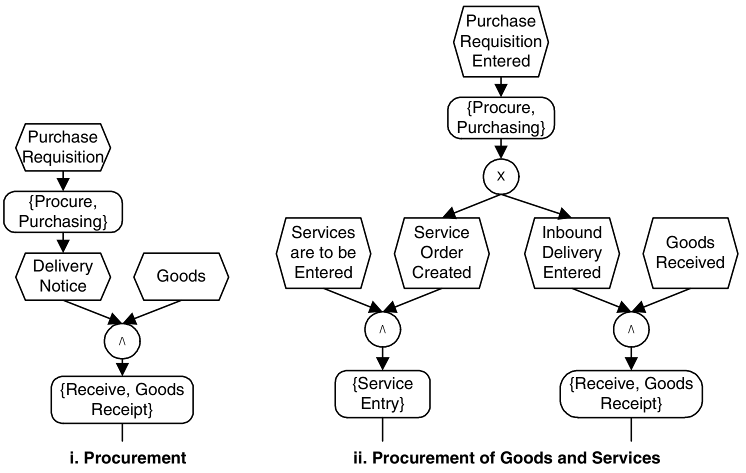


Fig. 3. Normalized EPCs

each class of equivalent functions. We repeatedly focus on a class of equivalent functions for two reasons.

1. By focusing on functions and their preceding functions, we can return the differences between EPCs in terms that are understandable to the process analyst.
2. By restricting (and subsequently reducing) the EPCs, the performance of the technique is acceptable in most practical cases in spite of its exponential complexity.

When focusing on an equivalence class of functions *fs*, we restrict the EPCs to the non-skipped functions from which we can reach a function in *fs* though a chain, consisting only of events, connectors or skipped functions. We call these functions the *potential enablers* of *fs*. We say that these function are the *potential enablers* rather than the *enablers*, because even if there is a chain from them to *fs* they may never enable the occurrence of *fs*. This is certainly true in potentially unsound EPCs, in which, for example, a deadlock may prevent *fs* from occurring as a result of one of its potential enablers. However, it is also true in sound EPCs, in which it may be necessary for a potential enabler of *fs* to first be succeeded by another potential enabler, before *fs* is really enabled. Skipped functions are ignored in the set of potential enablers, because they do not have a counterpart in the other process and therefore differences with respect to skipped functions are hard to track. However, skipped functions are implicitly considered, because the chains that they are on are considered. The *focus* of an equivalence class of functions *fs* consists of the set of its potential enablers of *fs* and *fs* itself.

For example, in Figure 1 {‘Receive’, ‘Goods Receipt’} can be reached via such a chain from {‘Procure’, ‘Purchasing’}. {‘Verify Invoice’, ‘Invoice Verification’} can be reached from {‘Receive’, ‘Goods Receipt’} and {‘Procure’, ‘Purchasing’} (because ‘Service Entry’ is skipped).

The restriction itself does not reduce the state-space of an EPC. However, it enables the use of reduction rules that do. Figure 4 shows the reduction rules that we use, we adapted them from the reduction rules presented in [7,14]. The reduction rules

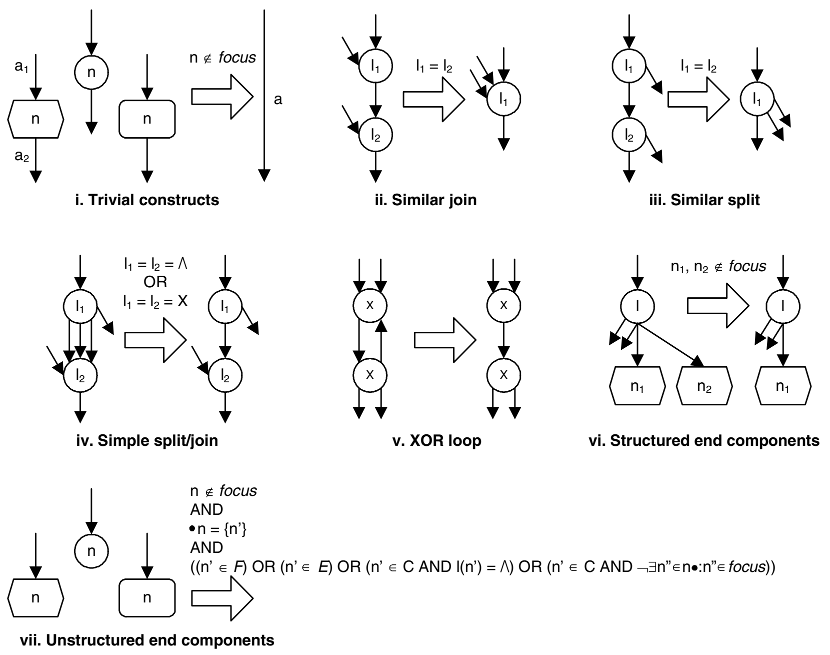


Fig. 4. Reduction rules that preserve completed trace equivalence

from Figure 4 reduce the state-space of an EPC. They lead to EPCs that are completed trace equivalent and therefore produce the same results when analyzing differences.

Rule 1 (trivial constructs). Reduces a node with one incoming and one outgoing arc to a single arc, provided that the node is not in the focus (control and event nodes never are). This reduction reduces the state-space, because it results in one less arc to maintain a token-state for. The reduction leads to an EPC that is completed trace equivalent with respect to the focus. This can be seen in Figure 4.i. Call the EPC before reduction E_1 and after reduction E_2. As indicated in the figure, call the arc pointing to the reduced node n a_1 and the arc originating from n a_2 and the reduced arc a. Because of the firing rules, tokens are put on a_1 if and only if they are put on a and E_1 can continue like E_2 after it has performed the firing rule $s \rightarrow^n s'$, leading to traces $Tr([E_1])$ that contain n's but are otherwise no different from the traces $Tr([E_2])$. Now, $Tr([E_1] \,/\, focus) = Tr([E_2] \,/\, focus)$, because $n \notin focus$ and therefore $s \rightarrow^n s'$ is replaced by $s \rightarrow^\varepsilon s'$.

Rule 2 (similar join). Reduces two similar join-nodes that are connected by an arc to a single join node. Both join nodes can have an arbitrary number of incoming arcs. The reduced EPC has the same behaviour with respect to tokens being consumed and produced by the construct and since the constructs do not lead to visible trace-output, the EPCs before and after the reduction are completed trace equivalent.

Rule 3 (similar split). Reduces two similar split-nodes that are connected by an arc to a single split node. Both split nodes can have an arbitrary number of outgoing arcs.

Rule 4 (simple split/join). Reduces a split-node connected to a join-node by an arbitrary number of arcs to nodes connected by a single arc, provides the nodes are of the same type and either AND or XOR. The split-node can have an arbitrary number of outgoing arcs and the join-node can have an arbitrary number of incoming arcs.

Rule 5 (XOR loop). Reduces a construct in which two XOR connector nodes that are in a loop that produces no visible output to two XOR connector nodes that are not in a loop. One connector node can have an arbitrary number of incoming arcs, the other can have an arbitrary number of outgoing arcs.

Rule 6 (structured end components). Reduces multiple end nodes preceded by a control node to a single end node, provided the end nodes are not in the focus set. Although the figure shows end events, the rule also applies to functions and connectors in any combination.

Rule 7 (unstructured end components). Reduces an end node that is not in the focus set, provided that the node that precedes it is: a function, an event, an AND control node, or a control node that is not succeeded by a node that is in the focus set.

For example, Figure 5 shows the EPCs from Figure 3 restricted to compare {'Receive', 'Goods Receipt'}. For this equivalence class, the equivalence class {'Service Entry'} is not in the focus set. Therefore, it can be removed using the 'trivial constructs' rule. Similarly, the events 'Delivery Notice', 'Service Order Created' and 'Inbound Delivery Entered' are not in the focus set (because events never are). Therefore, they can be removed using the 'trivial constructs' rule.

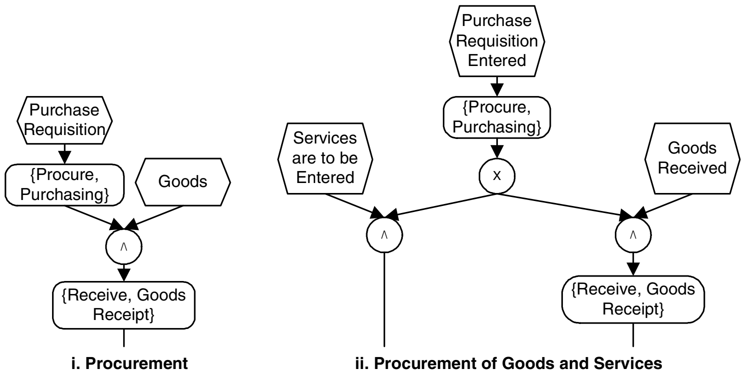


Fig. 5. EPCs restricted to compare {Receive, Goods Receipt}

3.4 Step 4: Compute Semantics of Restricted EPCs

The semantics of the restricted EPCs from the previous step can be computed, provided that the state space of the EPC is finite (which can be decided in a preprocessing step). The semantics can be computed, starting from the initial state of the EPC, by adding the transitions that can occur in each state and then recursively repeating this for the states that those transitions lead to.

3.5 Step 5: Diagnose Differences between Restricted EPCs

Now that the EPCs are normalized and restricted to a class of equivalent functions (and the functions that precede them), the differences between the EPCs with respect to those functions can be diagnosed. The diagnosis consists of two actions:

1. deciding whether the EPCs E_1 and E_2 are different ($Tr([E_1]) \neq Tr([E_2])$) with respect to this class of equivalent functions; and
2. deciding why they are different.

We use the following well-known and proven properties of NFA-ε [19].

- A deterministic finite-state automaton (DFA) is an NFA-ε $(S, \Sigma, \rightarrow, s_0, S_f)$ without ε transitions, in which every combination $s \in S$, $a \in \Sigma$ has at most one $s' \in S$ such that $s \rightarrow^a s'$.
- For each NFA-ε P it is possible to compute a DFA P' for which $Tr(P') = Tr(P)$. The algorithm to do so can be exponential in complexity.
- For each DFA P it is possible to compute a DFA P^C that represents the complementary set of traces over the alphabet Σ: $Tr(P^C) = \Sigma^* - Tr(P)$.
- For each two NFA-ε P_1 and P_2 it is possible to compute an NFA-ε $P_1 \cup P_2$ that represents the union of the traces of P_1 and P_2: $Tr(P_1 \cup P_2) = Tr(P_1) \cup Tr(P_2)$.
- For an DFA or NFA-ε P it is possible to compute whether $Tr(P) = \emptyset$.

Clearly these properties allow us to decide whether two restricted EPCs E_1 and E_2 are different ($Tr([E_1]) \neq Tr([E_2])$), by computing whether the following equivalent equation holds: $Tr(([E_1]^C \cup [E_2])^C) \neq \emptyset$ or $Tr(([E_1] \cup [E_2]^C)^C) \neq \emptyset$.

Subsequently, we use this equation to identify the type of the difference between the EPCs, using the typology from [4]. The type of difference provides information as to why the EPCs are different. Figure 6 shows the types of differences.

Different Conditions. A class of functions has different conditions in the two EPCs, E_1 and E_2, if it occurs in traces (i.e. under conditions) in E_1 that are not in E_2 *or* it occurs in traces in E_2 that are not in E_1. More precisely: $Tr(([E_1]^C \cup [E_2])^C) \neq \emptyset$ or $Tr(([E_1] \cup [E_2]^C)^C) \neq \emptyset$. Clearly, this means that 'different conditions' is the supertype of all differences. As an example, in Figure 6.i, c occurs in the traces abc, bac in E_1 and in the traces ac, bc in E_2. abc and bac are not in E_2 and ac and bc are not in E_1, such that c has different conditions in the EPCs.

Additional Conditions. A class of functions has additional conditions in one of the two EPCs, E_1 and E_2, if it *either* occurs in more traces in E_1 than in E_2 *or* in more traces in E_2 than in E_1. More precisely: $Tr(([E_1]^C \cup [E_2])^C) \neq \emptyset$ xor $Tr(([E_1] \cup [E_2]^C)^C) \neq \emptyset$. For example, in Figure 6.ii, c occurs in the traces abc, bac in E_1 and in the traces abc, bac, ac, bc in E_2. ac and bc are not in E_1, such that c has additional conditions in E_2.

Additional Start Condition. A class of functions, fs, has an additional start condition in one EPC (E_1 or E_2), if it can occur from the start in that EPC, but not in the other. Let F_{start} be the automaton that accepts all traces in which fs occurs from the start: $Tr(F_{start}) = fs\ \Sigma^*$, where Σ is the joint alphabet of E_1 and E_2 (this automaton can be constructed because $fs\ \Sigma^*$ is regular [19]). The *start* conditions of an EPC, E, are

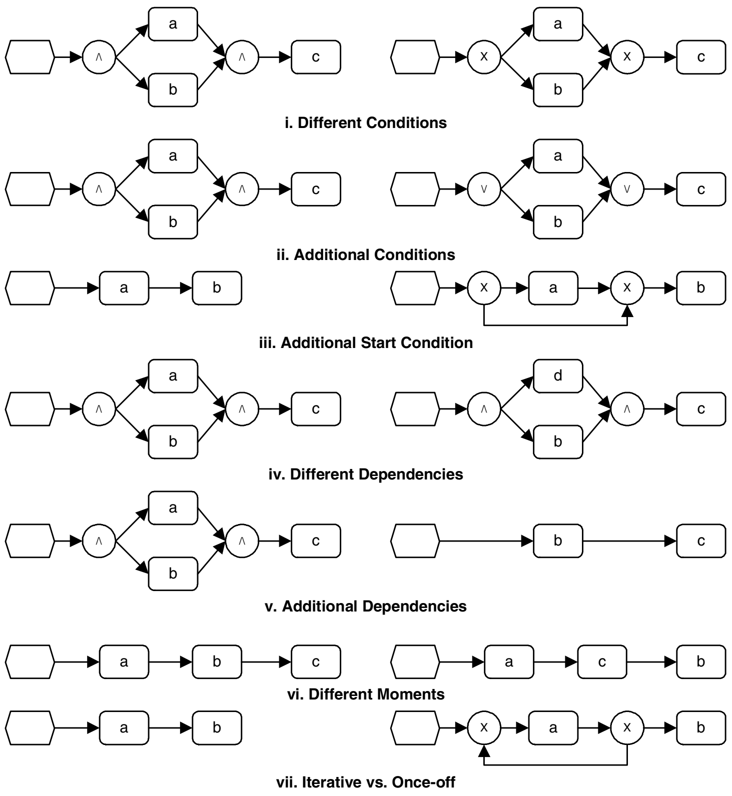

i. Different Conditions
ii. Additional Conditions
iii. Additional Start Condition
iv. Different Dependencies
v. Additional Dependencies
vi. Different Moments
vii. Iterative vs. Once-off

Fig. 6. Types of Differences

therefore: $Tr(F_{start}) \cap Tr([E])$. Hence additional start conditions can be detected by evaluating: $Tr(([E_1]^C \cup F_{start}{}^C \cup [E_2])^C) \neq \emptyset$ xor $Tr(([E_1] \cup F_{start}{}^C \cup [E_2]^C)^C) \neq \emptyset$. Figure 6.iii shows an example of additional start conditions. In the second EPC *b* can occur from the start, but in the first EPC it cannot.

Different Dependencies. A class of functions *fs* has different dependencies in two EPCs, E_1 and E_2, if the classes of functions that can enable the occurrence of *fs* (its dependencies) differ in E_1 and E_2. Let pe_1 be the potential enablers of *fs* in E_1 and pe_2 be the potential enablers of *fs* in E_2. We intersect the set of potential enablers of *fs* with the set of function classes that precede it in some trace, to derive the actual enablers of *fs* (remember that potential enablers may never actually enable a function). Let $e_1 = pe_1 \cap \{p \mid \text{for some } \sigma, \sigma' \in \Sigma^*: \sigma\ p\ fs\ \sigma' \in Tr([E_1])\}$ be the enablers of *fs* in E_1 and $e_2 = pe_2 \cap \{p \mid \text{for some } \sigma, \sigma' \in \Sigma^*: \sigma\ p\ fs\ \sigma' \in Tr([E_2])\}$ be the enablers of *fs* in E_2. Then *fs* has different dependencies in E_1 and E_2, if $e_1 - e_2 \neq \emptyset$ or $e_2 - e_1 \neq \emptyset$. Figure 6.iv shows an example of different dependencies for *c* (assuming that *a* and *d* are not skipped). The potential enablers of *c* in the first process are *a* and *b*, which are also the enablers of *c*, because there is a trace in which *a* directly precedes *c* and a trace in which *b* directly precedes *c*. The enablers of *c* in the second process are *d* and *b*. Hence, the sets of enablers differ.

Additional Dependencies. A class of functions *fs* has additional dependencies in one of two EPCs, E_1 and E_2, if it either has dependencies in E_1 that are not in E_2, or it has dependencies in E_2 that are not in E_1: $e_1 - e_2 \neq \varnothing$ xor $e_2 - e_1 \neq \varnothing$.

Different Moments. A class of functions *fs* occurs at different moments in EPC E_1 than in EPC E_2, if the sets of dependencies of *fs* in E_1 and E_2 are disjoint: $e_1 \cap e_2 \neq \varnothing$.

Iterative vs. Once-off. A class of functions *fs* occurs iteratively in one EPC, but once-off in another, if it is in a loop in the (automaton that describes) semantics of one EPC, but not in the other. This can be decided by constructing the spanning trees of these automata.

Some differences are subtypes of others (e.g. ‘additional conditions’ is a subtype of ‘different conditions’) while other types are disjoint (e.g. ‘additional dependencies’ is disjoint with ‘different moments’). A detailed overview of the relations between the types of differences is given in [5]. We use these relations to order the diagnosis, such that we do not report ‘useless’ differences (differences of which we can know whether they exist, based on the existence of other differences). The order is reversed with respect to the examples in Figure 6: vii – vi – v – iv – iii – ii – i. This means that we first look for a ‘vii’ (iterative vs. once-off), if it exists we return it and do not continue look for other differences, if it does not exist we continue to look for a ‘vi’ (different moments) and so on.

As an example, in Figure 1 an additional dependency will be returned for ‘Invoice Receipt’ on ‘Purchasing’. This reflects that in the second process invoices can be processed without having received goods (namely invoices for services). The skipped function ‘Service Entry’ is ignored.

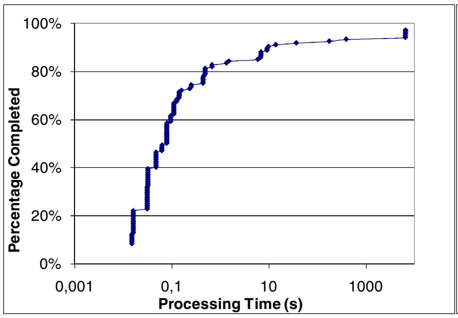
4 Case Study

We use a case study to:

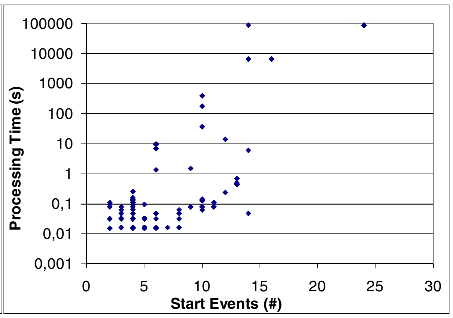
1. show the feasibility of the technique in spite of its exponential complexity, by showing that the computation times are acceptable; and
2. illustrate the usefulness of diagnosing differences, by showing:
   - a strong correlation between similarity of EPCs and differences found; and
   - the frequency of occurrence of each type of difference.

The case study is performed by applying the difference diagnosis technique to a subset of 132 pairs of EPCs from the SAP reference model. (Below we explain how these pairs were selected.)

Figure 7.i shows the percentage of EPC pairs for which the differences can be computed within a given number of milliseconds on a regular desktop computer. The figure shows that for 90% of the model pairs, computing the differences is a matter seconds. For 94% it is a matter of minutes. For 4 models (3%) we stopped computation after 3 hours. These results show that the technique can be applied in practice, in spite of exponential complexity. For the model pairs for which the computation took



i. Percentage completed within processing time



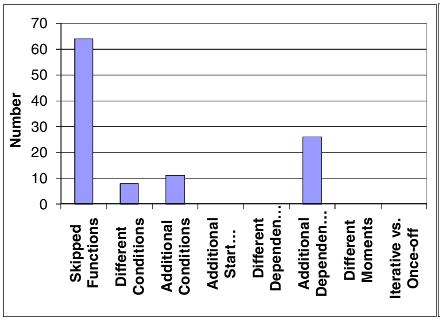
ii. Processing time as function of start events

Fig. 7. Processing Times

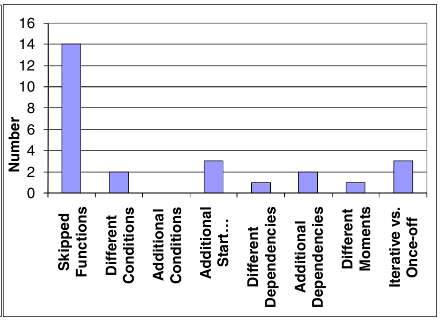
hours, this was invariably caused by a large (>15) number of start events. Figure 7.ii shows this. Therefore, the technique can be made practical by restricting the number of start events allowed in EPCs (a restriction that is used more often, see e.g. regular EPCs in [1]).

Since two processes are different if and only if they are not similar, there should be a strong correlation between the process similarity measure that we developed in previous work [6] and the differences found. To test this hypothesis, we applied the technique to a subset of 48 EPC pairs that are evenly distributed over the very dissimilar / very similar range (measured on a 0 to 1 scale). Indeed, we found a very strong Pearson correlation coefficient of 0.92 between the similarity score and the percentage of skipped functions. Interestingly, the same exercise showed that by far the most differences (other than 'skipped functions') occurred only in the 'somewhat similar' to 'very similar' range (0.5 to 1.0).

To illustrate the usefulness of the types of differences, we determined how many instances of each type can be found in the 'sales and distribution' branch of the SAP reference model, comparing only model pairs in the 'somewhat similar' to 'very similar' range (0.5 to 1.0), a total of 84 pairs. Figure 8.i shows the results. Based on these



i. In part of the SAP reference model



ii. In earlier case study [4]

Fig. 8. Frequency of Found Differences

results, we argue that the ‘additional dependencies’, ‘additional conditions’, ‘different conditions’ and ‘skipped function’ differences provide useful feedback about differences between EPCs. Since no instances are found of the other types, these types are arguably not useful.

However, we claim that, in spite of these results, the ‘iterative vs. once-off’ difference is useful. We argue this, because an earlier case study [4] showed a significant number of ‘iterative vs. once-off’ differences (see Figure 8.ii), considering that only 5 pairs of models were compared. Moreover, the ‘sales and distribution’ branch of the SAP reference model does not contain any process loops at all. For which reason it is not surprising that there are no differences with respect to the loop construct. However, in other process models this construct is quite common. Therefore, we claim that the ‘iterative vs. once-off’, although not used in this case study, can still prove its use in other cases. In the earlier case study there also was a strong relation between the ‘iterative vs. once-off’ difference and the ‘different start’ difference. Therefore, we claim that this difference can also still prove its use. The earlier case study also showed that the other two differences (‘different dependencies’ and ‘different moments’) are rare.

Interestingly, an ‘additional dependencies’ difference often occurs multiple times (2 times on average), *when* it occurs. This suggests that there can be another difference underlying this difference. This observation deserves further study.

5 Related Work

The work in this paper is based on the formal notions of behavioural equivalence. A survey on different notions of behavioural equivalence is given in [9]. Also, work on detecting differences based on the non-equivalence exists [3]. However, this work returns differences in terms of a formal semantics, while it is the goal of out technique to return them in terms that are more easily understood by a business analyst. The work in [3] also illustrates why we do not use a bi-similarity equivalence, because this work only returns a single difference for a pair of processes.

Only recently detection of differences is approached by identifying types of differences. Benatallah et al. [2] and Dumas et al. [8] developed typologies of differences between interacting business processes. Motari Nezhad et al. [16] also define detection techniques for such differences. Our work differs from theirs in that we focus on similar processes, while their work focuses on interacting processes.

The work on process conformance checking is also close to our work [18]. During conformance checking the differences between a process and the event traces of that process are identified. Event traces are traces that are observed in reality. Our work differs from the work on conformance checking, in that conformance checking diagnoses differences between a process and its event traces, while our work diagnoses differences between processes. However, it has been suggested that a representative set of event traces can be generated for a process, allowing diagnosis of differences between two processes (for which sets of event traces are generated.)

A possible future direction for detecting differences between business processes is using the notion of ‘edit-distance’, which measures the operations necessary to het from one process model to another [12].

Typologies of process differences have also been developed in related areas of research, such as process integration [17] (where differences are called heterogeneities) and process evolution [20].

6 Conclusion

This paper presents a technique to diagnose differences between EPCs. The diagnosis provides feedback by pointing out the EPC functions between which there is a difference and the type of the difference (one of 8 possible types). The technique is an extension to techniques for determining equivalence of processes. These techniques return differences between processes in terms of the formal semantics of those processes. However, returning the differences in terms of the processes themselves is more intuitive to a process analyst, who is not versed in formal semantics.

Although the technique is restricted to use on EPCs in this paper, we are experimenting with the use of the technique on other modelling languages [5].

Currently, the technique uses the notion of completed trace equivalence for determining differences. Stronger notions of equivalence exist, such as the notion of branching bi-similarity [9] which is popular for deciding equivalence between business processes. However, the drawback of these notions is that techniques to detect them only return a single difference (in terms of a pair of states in which processes differ), while it is our goal to yield as many differences as possible.

Clearly, our choice of semantics means that we cannot detect certain differences. In particular we cannot detect differences with respect to branching time of processes. Also, because of the repeated restriction that is necessary to detect the differences, there is a suspicion that not in all cases in which there is a difference (i.e. the processes are not completed trace equivalent) a difference will be detected. However, we have not been able to find any counter examples, nor have we been able to prove that they do not exist. If counter examples can be found then the typology of differences is incomplete in the sense that it does not classify all possible differences under completed trace equivalence. Regardless, the differences that *can* be detected remain useful.

A case study concerning 132 pairs of process models from the SAP reference model shows that 4 types of differences occur frequently, providing an argument for the usefulness of these differences. We attribute the non-occurrence of 2 types of differences to the absence of process loops in the SAP reference model. Typically, processes do contain loops and, therefore, differences with respect to loops can be expected in other processes. An earlier case study supports this claim [4].

The technique has exponential complexity. However, the case study shows that it can still be used in practice and will produce the differences between EPCs within seconds in 90% of the cases and within 6 minutes in 94% of the cases. Moreover, the case study shows that, if the number of start events is constrained, the differences can be produced within an acceptable timeframe in 100% of the cases.

References

1. van der Aalst, W.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41, 639–650 (1999)
2. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing Adapters for Web Services Integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
3. Cleaveland, R.: On Automatically Explaining Bisimulation Inequivalence. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 364–372. Springer, Heidelberg (1992)
4. Dijkman, R.M.: A Classification of Differences in Similar Business Processes. In: EDOC 2007, pp. 37–47 (2007)
5. Dijkman, R.M.: Feedback on Differences between Business Processes. BETA Working Paper WP-234, Eindhoven University of Technology, The Netherlands (2007)
6. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring Similarity between Business Process Models. In: CAiSE 2008, pp. 450–464 (2008)
7. van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Verification of EPCs: Using reduction rules and Petri nets. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 372–386. Springer, Heidelberg (2005)
8. Dumas, M., Spork, M., Wang, K.: Adapt or Perish: Algebra and Visual Notation for Interface Adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
9. van Glabbeek, R.: The Linear Time – Branching Time Spectrum I: The Semantics of Concrete Sequential Processes. Handbook of Process Algebra, pp. 3–99. Elsevier, Amsterdam (2001)
10. Keller, G., Nüttgens, M., Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
11. Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. Data & Knowledge Engineering 56, 23–40 (2006)
12. Lohmann, N.: Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In: BPM 2008 (accepted, 2008)
13. Mendling, J., van der Aalst, W.M.P.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 439–453. Springer, Heidelberg (2007)
14. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Ph.D. Thesis, Vienna University of Economics and Business Administration, Austria (2007)
15. Mendling, J., van der Aalst, W.: Towards EPC Semantics based on State and Context. In: EPK 2006, pp. 25–48 (2006)
16. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007, pp. 993–1002 (2007)
17. Preuner, G., Conrad, S., Schrefl, M.: View Integration of Behavior in Object-Oriented Databases. Data & Knowledge Engineering 36(2), 153–183 (2001)
18. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
19. Sudkamp, T.: Languages and Machines, 2nd edn. Addison-Wesley, Reading (1996)
20. Weber, B., Rinderle, S., Reichert, M.: Change Patterns and Change Support Features in Process-Aware Information Systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)

BPEL for REST

Cesare Pautasso

Faculty of Informatics, University of Lugano
via Buffi 13, 6900 Lugano, Switzerland
cesare.pautasso@unisi.ch

Abstract. Novel trends in Web services technology challenge the assumptions made by current standards for process-based service composition. Most RESTful Web service APIs, which do not rely on the Web service description language (WSDL), cannot easily be composed using the BPEL language. In this paper we propose a lightweight BPEL extension to natively support the composition of RESTful Web services using business processes. We also discuss how to expose the execution state of a business process so that it can be manipulated through REST primitives in a controlled way.

1 Introduction

With the goal of attracting a larger user community, more and more service providers are switching to REST [1] in order to make it easy for clients to consume their Web service APIs [2,3,4]. This emerging technology advocates a return to the original design principles of the World Wide Web [5] to provide for the necessary interoperability and enable integration between heterogeneous distributed systems. The HTTP standard protocol thus becomes the basic interaction mechanism to publish existing Web applications as services by simply replacing HTML payloads with data formatted in "plain old" XML (POX) [6].

In this context, many of the assumptions made by existing languages for Web service composition no longer hold [7]. Since most RESTful Web service APIs do not use the standard Web service description language (WSDL) to specify their interface contracts, it is not possible to directly apply existing languages, tools and techniques that are built upon this standard interface description language. Considering that REST prescribes the interaction with resources identified by URIs [8], languages that assume static bindings to a few fixed communication endpoints do not cope well with the dynamic and variable set of URIs that make up the interface of a RESTful Web service. Even the basic message-oriented invocation constructs for sending and receiving data cannot be consistently applied with the uniform interface principle of a RESTful Web service. Moreover, in some cases, also the assumption of dealing with XML data [9] may not apply when accessing resources represented in other, more lightweight, formats such as the JavaScript Object Notation (JSON [10]).

In this paper we argue that process-based composition languages can and should be applied to compose RESTful Web services in addition to WSDL-based ones. However, given the differences between the two kinds of services [11,12], we claim that native support for composing RESTful Web services is an important requirement for a modern service composition language. To address this requirement, we show how the Business Process Execution Language (WS-BPEL [13]) standard can be extended. Also, following the recursive nature of software composition (the result of a composition should be composable [14]), we propose to apply REST principles to the design of the API of a BPEL engine so that processes themselves (and a view over their execution state) can be published through a RESTful Web service interface.

The rest of this paper is structured as follows. The motivation for our work is presented in Section 2. We continue in Section 3 giving some background on RESTful Web services and outlining the challenges involved in composing this novel kind of services. In Section 4 we introduce our extensions to the BPEL language with an example loan application process. The extensions are then specified in Section 5. The relationship between the resource and the process abstractions is discussed in Section 6. Related work is outlined in Section 7. We draw our conclusions in Section 8.

2 Motivation

The following quote from the specification of the WS-BPEL 2.0 standard motivated the research towards the extensions presented in this paper.

> The WS-BPEL process model is layered on top of the service model defined by WSDL 1.1. [. . .] Both the process and its partners are exposed as WSDL services [13, Section 3].

In other words, the WS-BPEL and the WSDL languages are tightly coupled. The *partner link type* construct, introduced in BPEL to tie together pairs of matching WSDL port types, forces all external interactions of a process to go through a WSDL interface. Over time, this strong constraint on the original design of the composition language has produced a set of extensions (e.g., BPEL-SPE [15] for processes invoking other subprocesses, BPEL4People [16] for interaction with human operators, or – more recently – BPEL4JOB [17] with job submission and fault handling extensions to better deal with the requirements of scientific workflow applications, BPEL-DT [18] to compose data intensive applications, and BPEL^{light} [19] proposing a complete decoupling of the two languages in the context of message-oriented interactions). Along the same direction, in this paper we propose another extension to make BPEL natively support the composition of RESTful Web services.

Without loss of generality, we will ignore the differences between WSDL 1.1 and the latest WSDL 2.0 version and assume that WS-BPEL will soon be updated to support the latter. This is important because – as shown in Figure 1 – the new HTTP binding introduced in WSDL 2.0 can be used to wrap a RESTful Web service and describe its interface using the WSDL language [20]. Thus, the problem appears to be already solved: with this binding, a BPEL process could send and receive messages over the HTTP protocol without necessarily using the SOAP message format.

From a practical standpoint, however, the approach does not lead to a satisfactory solution for the following reasons. WSDL 2.0 is not yet widely deployed in the field, especially to describe existing RESTful Web service APIs, and there is very little evidence indicating that this will change in the future. Thus, at the moment the burden

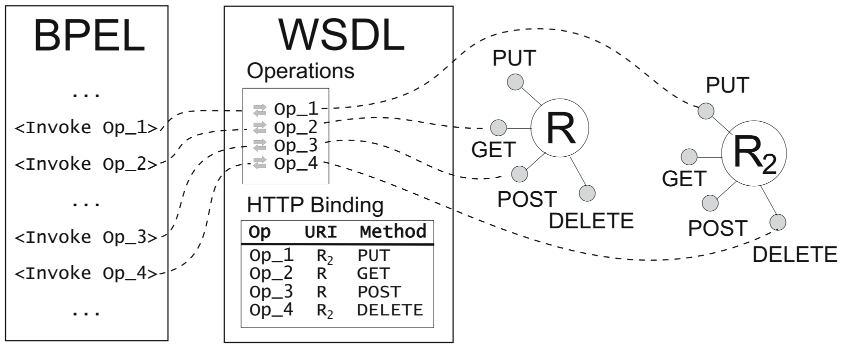


Fig. 1. Funneling RESTful Web services through a WSDL 2.0 interface in order to invoke them from a BPEL process

of going through the procedure of recreating a synthetic WSDL description for the RESTful Web service interface is shifted from the service provider to the BPEL developer [21]. Every supplementary artifact introduced in a solution makes it more complex and more expensive to maintain. The additional WSDL description needs to be updated whenever changes are made to the corresponding RESTful Web service.

From a theoretical point of view, this approach hides the resource-oriented interaction primitives of REST inside the service-oriented abstractions provided by WSDL. As we are going to discuss, the invocation of a WSDL operation by means of synchronous or asynchronous message exchange does not always fully match the semantics of a “GET”, “PUT”, “POST”, or “DELETE” request performed on a resource URI [11,22,23,24]. Thus, we argue that native support for explicitly controlling the interaction based on REST primitives would be beneficial to developers trying to apply the BPEL language to specify how to compose resources, as opposed to WSDL-based services (Figure 2).

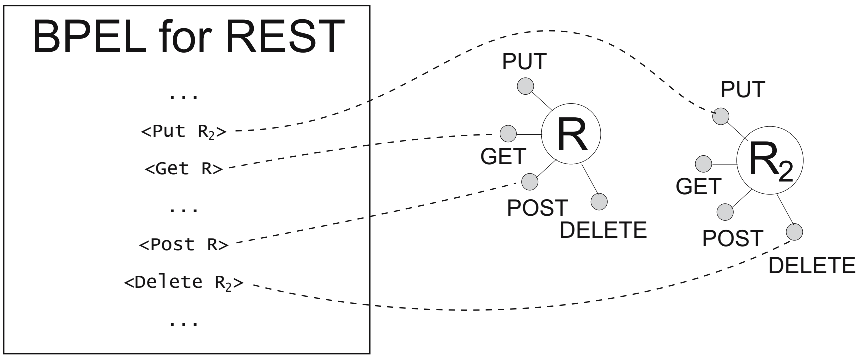


Fig. 2. Direct invocation of RESTful Web services using the BPEL for REST extensions

3 Composing RESTful Web Services

The composition of RESTful Web services is typically associated with so-called Web 2.0 Mashups [25,26], where this emerging technology helps to reduce the complexity and the effort involved in scraping the data out of HTML Web pages [27]. In this context, a consensus still needs to be reached in terms of how to describe the interface of such RESTful Web service. As we previously discussed it is not always convenient to use the HTTP binding of the latest version of the Web *service* description language (WSDL 2.0). Whereas the Web *application* description language (WADL [28]) has been recently proposed, most APIs still rely on human-oriented documentation. This typically includes interactive examples that help developers to infer how to use a particular service. If XML is employed as representation format, an XML Schema description is often associated with the textual documentation.

This uncertain situation makes it challenging to define a composition language for REST as it is currently not yet possible to assume that a particular service description language will be used. Thus, like [19], we choose not rely upon the presence of such a description language and introduce language constructs for service invocation that directly map to the underlying interaction mechanisms and patterns [29].

In the rest of this section we give an overview over the design principles followed by RESTful Web services [1,6] and discuss how these challenge the BPEL language in its current form. These principles and challenges have been taken into account during the design of the corresponding BPEL extensions presented in this paper.

Resource addressing through URI – The interface of a RESTful Web service consists of a set of resources, identified by URIs. From the client perspective, the interaction with a RESTful Web service requires to interact with a dynamic and variable set of URIs, which are not necessarily known nor identifiable in advance. From the service provider viewpoint, it should be possible to use languages such as BPEL to manage the lifecycle of arbitrary resources and to implement the state transition logic of composite resources [30].

Uniform interface – The set of operations available for a given resource is fixed by design to the same four methods: PUT, GET, POST, and DELETE. These CRUD-like operations apply to all resources with similar semantics. POST creates a new resource under the given URI; GET retrieves the representation of the current state of a resource; PUT updates the state of a resource (and may also initialize the state of a new resource if none was previously found at the given URI); DELETE frees the state associated with a resource and invalidates the corresponding URI. Each of these methods (or verbs) is invoked on a resource using a synchronous HTTP request-response interaction round. Thus, there is no need for the asynchronous (one-way) message exchange patterns supported by BPEL/WSDL. Also, since the set of operations is fixed to the four methods, there is no apparent need to explicit enumerate them using an interface description language.

Self-descriptive messages – Thanks to meta-data, clients can negotiate with service providers the actual representation format of a resource. Thus, it is not always possible to statically assign a fixed data type to the payload of an HTTP response. Also, since RESTful Web services may not always exchange XML data, BPEL variables need to accomodate a larger set of possible data representation formats. Additionally, meta-data is used to perform client and server authentication, access control, compression, and caching. Thus, from within a BPEL process it should be possible to access and control this meta-data, in a way similar to how SOAP message headers are configured.

Hypermedia as the engine of application state – Whereas every interaction is kept stateless using self-contained request and response messages, stateful interactions are based on the concept of explicit state transfer. Through hyperlinks, valid future states of the interaction can be embedded in the representation of a resource and discovered interactively by clients. From this principle, it follows that resource URIs may be dynamically generated by a service. Thus, a mechanism for extracting URIs from response messages is needed together with a construct for dynamic binding of activities to target resource URIs.

4 Example

Before giving a complete specification of the BPEL for REST extensions, in this section we informally introduce them within the example process illustrated in Figure 3.

The Loan application process exposes one resource, the `loan` that represents the state of a loan application. Clients can initiate a new loan application with a PUT request on this resource and retrieve the current state of their application with GET. A DELETE request immediately cancels the application if it is still in progress, otherwise it triggers the execution of a different loan cancellation process. The lifecycle of a loan application goes through several stages as specified by the sequence of activities triggered by the PUT request. In particular, the process invokes the RESTful Web services of two different banks to gather available interest rates and payment deadlines. It will then wait for the client to make a decision choosing between the two competing offers. Once the client has chosen an offer, the BPEL process will confirm the acceptance with a POST request that is dynamically bound to a URI provided by the bank service (indicated with `$accept` in the Figure) together with the offer.

For clarity, in the following we highlight the BPEL for REST extension activities in **`boldface`**. Also, to enhance the readability of the XML code, we have omitted namespace declarations and taken some liberty with the syntax of the `<assign>` activity (this simplified syntax is not part of the proposed language extension). Variable interpolation is indicated by prefixing the name of variables with the `$` sign.

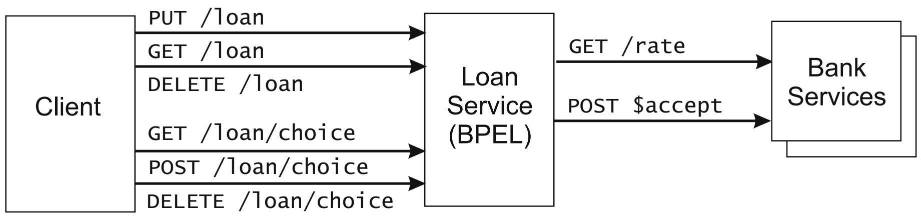


Fig. 3. Loan Application Process Example

```
<process name="LoanApplication">
<resource uri="loan">
<!-- State variables of the resource -->
 <variable name="name"/>
 <variable name="amount"/>
 <variable name="rate"/>
 <variable name="bank"/>
 <variable name="start_date"/>
 <variable name="end_date"/>
```

These variable declarations store the state of the `loan` resource declared within the `LoanApplication` BPEL process. Associated with the resource, the process specifies three request handlers: `onPut`, `onGet`, `onDelete`.

```
<!-- PUT /loan request handler -->
<onPut>
 <if><condition>$request.amount &gt 100000</condition>
  <then>
   <response code="400">
    Requested amount too large
   </response>
   <exit/>
  </then>
  <else>
   <sequence>
    <assign>
          name = $request.name;
        amount = $request.amount;
    start_date = $request.start_date;
    </assign>
    <response code="201">
     Processing loan application
    </response>
```

The PUT request is used to initialize the state of the `loan` resource. However, if the requested loan amount is too large, the client will be informed with a `response` carrying the HTTP code 400 (Bad Request) and the resource will be immediately deleted using the BPEL `<exit>` activity. Otherwise the state of the resource is initialized from the BPEL for REST predefined variable called `$request` which stores the input payload of the HTTP PUT request. After informing the client that the resource could be created (HTTP response code 201), the `onPut` request handler continues with the next step of the loan application process, when two different RESTful Web services are invoked to retrieve the available interest rates for the given loan request.

```
 <!-- Get rates from two different bank services -->
  <scope>
   <variable name="ubs_response"/>
   <variable name="cs_response"/>
   <variable name="url_accept"/>
   <variable name="accept_response"/>
   <flow>
    <get uri="http://www.ubs.ch/rate?chf=$amount&from=$start_date"
         response="ubs_response">
    <get uri="http://www.cs.ch/rates?amount=$amount&start=$start_date"
         response="cs_response">
   </flow>
```

The two services are invoked in parallel using the BPEL `<flow>` activity. They are invoked using a GET request on a URI that is constructed using the current state of the `loan` resource. The response of the services are stored in the corresponding variables, if the invocation is successful.

At this point, the client should decide which loan rate is preferred. To do so, the process dynamically creates a new resource at the URI `loan/choice`. A GET on this new resource will retrieve the current offers (represented in JSON) while a DELETE request can be used to reject both offers and cancel the entire loan application process.

```
<!-- Let client choose the preferred bank -->
     <while>
      <condition>TRUE</condition>
      <resource uri="choice">
       <onGet>
<!-- Return the rates offered by the banks -->
        <response code="200">
         <header name="Content-Type">application/json</header>
      [ {  bank:"cs",
           rate:"$cs_response.rate",
           end_date:"$cs_response.until" },
        {  bank:"ubs",
           rate:"$ubs_response.rate",
           end_date:"$ubs_response.end" } ]
        </response>
       </onGet>
       <onDelete>
<!-- Reject the offer and cancel the loan application -->
        <sequence>
         <response code="200"/>
         <exit/>
        </sequence>
       </onDelete>
       <onPost>
<!-- Store the client choice and continue -->
       <sequence>
        <assign>bank = $request.choice;</assign>
        <if><condition>bank == "cs"</condition>
         <then><assign>rate = $cs_response.rate;
                       end_date = $cs_response.until;
                       url_accept = $cs_response.accept</assign></then>
         <else><assign>rate = $ubs_response.rate;
                       end_date = $ubs_response.end;
                       url_accept = $ubs_response.accept</assign></else>
        </if>
        <response code="200"/>
        <activeBPEL:break/>
       </sequence>
      </onPost>
     </resource>
    </while>
```

To continue the execution of the process, the client must communicate its choice using a POST request. The process will update the state of the loan resource with the client decision and the corresponding information from the chosen offer: the `rate` and `end_date` of the loan and the URL to use in order to confirm the acceptance of the offer with the bank.

A successful (code 200) response is then returned to the client and the execution continues by exiting the `<while>` loop[1]. Once the execution exits the `scope` of the resource `choice` declaration, such resource is no longer available to clients. Further requests to its URI will result in a 404 (Not Found) code being returned by the BPEL for REST engine. To conclude the `onPut` request handler, the chosen bank service is informed by sending the client's name with a POST request dynamically bound to the acceptance URL that was returned with the terms of the offer.

```
<!-- Accept the loan offered by the chosen bank -->
    <post uri="$url_accept" request="$name" response="accept_response">
   </scope>
  </sequence>
  </else>
 </if>
</onPut>
```

It is important to point out that once the execution of the `<onPut>` request handler is completed, the state of the newly created `loan` resource is not discarded, but it will remain available to clients until the corresponding DELETE request is issued. As illustrated in the final part of the example, clients can retrieve such state at any time using a GET request. To cancel the loan application, clients may issue a DELETE request. However, depending on the state of the loan resource, canceling it may not always be possible and may require to execute the corresponding loan cancellation business process.

```
<!-- GET /loan request handler -->
<onGet>
<!-- Return the state of the loan application -->
</onGet>

<!-- DELETE /loan request handler -->
<onDelete>
 <if> <condition>bank == null</condition>
  <then>
   <response code="200"/>
   <exit/>
  </then>
  <else>
<!-- Start the loan cancellation process -->
   <invoke...>
  </else>
 </if>
</onDelete>
</resource>
</process>
```

5 BPEL for REST Extensions

As shown in the previous example, in this paper we propose two kinds of extensions. First, it should be possible to invoke a RESTful Web service directly from a BPEL

[1] For simplicity, the process calls the `<break>` activity (a non-standard BPEL extension introduced by the ActiveBPEL engine). However it would also be possible to do so by setting the appropriate flag to be tested in the loop condition.

```
<get uri="" response="">
  <header name="">*value</header>
  <catch code="" faultName=""?>*...</catch>
  <catchAll>?...</catchAll>
</get>
<post uri="" request="" response=""> ... </post>
<put uri="" request="" response=""?> ... </put>
<delete uri="" response=""?> ... </delete>
```

Fig. 4. BPEL for REST extensions to invoke a RESTful Web service

process. Also, we propose a declarative construct to expose parts of the execution state of a BPEL process as a resource. To do so, we choose the introduce a set of activities, constructs and handlers that are directly related to the REST uniform interface principle.

5.1 Invoking RESTful Web Services

To invoke a RESTful Web service using the HTTP (or HTTPS) protocol from a BPEL process, we add these four activities: `<get>`, `<post>`, `<put>`, `<delete>`.

As shown in Figure 4, the four activities use the `uri` attribute to specify the target resource address. The URI can be a constant value, but also be computed out of data currently stored in the process variables. Thus, BPEL for REST supports dynamic binding to invoke resource URIs that are only known at runtime. The only constraint on the structure of the URI is that it should target a resource accessible using the HTTP or the HTTPS protocols.

Following the convention of the existing BPEL `<invoke>` activity, the data for the request and response payloads is stored in variables that are referenced from the corresponding `request` and `response` attributes. In case of `<get>` and `<delete>` activities, there is no request payload as these REST primitives operate on the resource URI only. For `<put>` and `<delete>` the response attribute is optional, as some services may return an empty payload with these two methods.

The headers sent with the HTTP request can be controlled using the `<header>` child elements of each of the four invocation activities. Also in this case, their values can be set to constant values but also computed from information stored in BPEL variables.

Similar to standard BPEL `<invoke>` activities, `<get>`, `<post>`, `<put>`, and `<delete>` are equipped to deal with invocation failures. In particular, if an HTTP code indicating an error (i.e., 4xx or 5xx) is detected, the activity will fail and raise the corresponding BPEL `fault` that can be caught by a standard fault handler. As shown in Figure 4, fault handlers in BPEL for REST can be associated with specific HTTP status codes. Unless a specific fault handler is specified, all other HTTP codes (e.g., like 3xx used to indicate a redirection) will be transparently managed by the BPEL engine.

5.2 Publishing Processes as RESTful Web Services

To declaratively publish certain sections of a BPEL process as a resource we introduce the `<resource>` container element (Figure 5). This construct allows to dynamically

```
<resource uri="">
 <variable>*
 <onGet>?     ... </onGet>
 <onPut>?     ... </onPut>
 <onDelete>?  ... </onDelete>
 <onPost isolated="false"?>?    ... </onPost>
</resource>

<response code=""?>
 <header name="">*value</header>
  payload
</response>
```

Fig. 5. BPEL for REST extensions to declare resources within a process

publish resources to clients depending on whether their declarations are reached by the execution of the BPEL process. Once a process reaches the `<resource>` element, the corresponding URI is published and clients may start issuing requests to it. Once execution leaves the `scope` where the `<resource>` is declared, its URI is no longer visible to clients that instead receive an HTTP code 404 (Not found). Resources that are declared as top-most elements in a BPEL process never go out of scope and they are immediately published once the BPEL process is deployed for execution. As shown in the example, resource declarations can be nested. The URI of nested resources is computed by concatenating their `uri` attribute with the usual path (`/`) separator.

Similar to the BPEL `<scope>`, a `<resource>` may contain a set of `<variable>` declarations that make up the *state* of the resource found at a given `uri`. These state variables are only visible from within the resource declaration.

Like the BPEL `<pick>`, the `<resource>` contains a set of handlers that are triggered when the process receives the corresponding HTTP request. As opposed to `<pick>`, which contains one or more `onMessage`/`onAlarm` handlers, the request handlers found within a `<resource>` directly stem from the REST uniform interface principle. They are: `<onGet>`, `<onPost>`, `<onPut>`, `<onDelete>`. If a request handler for a given verb is not declared, requests to the resource using such verb will be answered by the BPEL engine with HTTP code 405 (Method not allowed). At least one request handler must be included in a resource declaration and a handler for a given request may appear at most once.

Another difference with `<pick>` is that there is no limit on the number of times one such handler may be concurrently activated during the lifetime of the resource it is attached to. If multiple clients of a BPEL process issue in parallel a GET request on a resource declared from within the process, the execution of the corresponding `onGet` request handler will not be serialized. Since only POST requests are not meant to be *idempotent*, the `<onPost>` handler may be flagged to guarantee proper isolation[2] with respect to the access of the resource state variables. To ensure that GET requests on a resource are indeed *safe*, the `onGet` request handler only has read-only access to the state variables of the corresponding resource.

[2] Similar to the BPEL isolated scope [13, Section 12.8].

The behavior of a request handler can be specified using the normal BPEL structured activities (i.e., `<sequence>`, `<flow>`, etc.). However, control-flow `links` across different handlers are not supported.

To access the data sent by clients with the request payload, a pre-defined variable called `$request` is available from within the scope of the request handler. Likewise, a variable called `$request_headers` gives read-only access to the HTTP request headers.

Results can be sent back to clients using the BPEL for REST `<response>` activity. Its `code` attribute is used to control the HTTP response status code that is sent to clients to indicate the success or the failure of the request handler. The response headers can be set using the same `header` construct introduced for the invocation of a RESTful Web service. The payload of the response is embedded within the body of the element, but could also be precomputed in a variable (i.e., by inlining a reference to the `$variable` in the body of the element). Whereas at least one `response` element should be found within a request handler, in more complex scenarios it could be useful to include more than one (e.g., to stream back to clients over the same HTTP connection multiple data items as they are computed by the BPEL process). In this case, only the first `response` element should specify the HTTP code and the headers of the response. The connection with the client will be closed after the last `response` element is executed. As shown in the `onPut` request handler of the example, a `response` does not need to be placed at the end of the request handler, as the handler execution may continue even after the response has been sent to the client.

5.3 Minor BPEL Extensions and Changes

In this section we discuss a few minor extensions and small changes to the semantics of existing BPEL activities to round off the support for REST in the language.

As shown in the example, the BPEL `<exit/>` activity – in addition to completing the execution of the process – has the additional effect of discarding the state of all resources that were associated with the process. Since nested resources are implicitly discarded as execution moves out of their declaration scope, `exit` has only an effect on the state of the top-level resources, which would remain accessible to clients even after the normal execution of the process has completed.

Given the absence of a static interface description for RESTful Web services, and the lack of strong typing constraints on the data to be exchanged, BPEL for REST is a dynamically typed language. Thus, static typing of `<variable>` declarations becomes optional [13, SA025]. In particular, the attribute `messageType` – being directly dependent on WSDL – is not used, while the `type` or `element` attributes may still be used in the presence of an XML schema description for the RESTful Web service.

6 Discussion

In BPEL for REST, the concept of business process is augmented with notion of resource. We have introduced the `resource` declarative construct to specify that at a certain point of the execution of a process, parts of its state can be published to clients using a RESTful Web service interface. Thus, it is important to understand what is the relationship between the state of a BPEL process instance and the state of such resources.

The lifecycle of typical BPEL process instances begins with the execution of a so-called *start activity*, which may be either an *instantiating receive* or a `pick` configured with a `createInstance="Yes"` attribute. Once a process has been instantiated, its state consists of the values assigned to its variables together with an "instruction pointer" indicating which subset of its activities are currently active. During execution, all messages exchanged are correlated with a particular process instance based on their content and on the correlation sets declared in the process. Execution of a process instance proceeds until it reaches an `exit` activity or the process simply runs out of activities that can be executed. The state of a process instance is typically discarded once execution has completed.

The lifecycle of resources should follow the REST uniform interface principle. Resources are created (or initialized) with a POST/PUT request. Once a resource has been created, clients may read its current state using GET requests, update its state using PUT requests, and discard its state using DELETE requests.

In BPEL for REST, the state of a resource is accessed and manipulated from within the resource request handlers. A new resource instance is created by initializing the resource state variables from within the `<onPut>` or `<onPost>` request handler. To let clients identify a specific resource instance, in the simplest case, an HTTP Cookie can be automatically generated by the BPEL engine handling PUT requests. The engine may intercept responses carrying the HTTP status code 201 (Created) and add the cookie with a unique identifier. Clients will send the cookie for all future interactions (e.g., GET, PUT, and DELETE) with the resource URI and the engine will use the cookie to correlate the requests with the correct state of the resource instance.

As proposed in [31,30], a cookie-free solution based on URI rewriting that involves the template-based generation of resource identifiers is also possible. This would work as follows: given a top-level URI resource (e.g., `/loan`) corresponding to a process model, if a POST request is answered with a 201 (Created) status code, the engine detects this and adds a `Location: /loan/i` redirection header (where `i` is a unique identifier of the newly created instance). Further GET, PUT, and DELETE requests from clients will have to be directed to the specific resource instance URI `/loan/i`. Nested resource URIs are still constructed by concatenating their `uri` attributes, but should now also include the resource instance identifier. A GET request to the original resource URI `/loan` could be then used to return hyperlink references to all active process instances managed by the engine.

BPEL for REST distinguishes between two aspects of a resource that can be published from a process: its URI and its state stored in the variables declared within the resource. If a resource is declared as a top-level element of a BPEL process, clients can interact with its URI as soon as the BPEL code is deployed for execution, no matter whether a process instance has yet been started. If a resource is declared from within a local scope of the process, its URI is published only once the execution of a particular process instance reaches the particular scope. By introducing this distinction between top-level and local resource declarations, BPEL for REST supports a pure resource-oriented style of composition, where the result of a BPEL process is a resource that is accessed through the REST uniform interface and can be instantiated multiple times. Nevertheless, BPEL processes can also be instantiated using standard compliant *start activities* and publish resources during their execution that expose part of their state to clients also using the REST uniform interface.

All in all, the goal of the BPEL for REST extensions is to follow a declarative approach to embedding resources within processes so that developers do not have to worry about correlating requests with resource instances, a feature that should be handled transparently by the engine.

7 Related Work

BPEL for REST builds upon several existing research contributions within the area of Web service composition languages [32,33]. Also, from the practical side, ad-hoc support for invoking RESTful Web services is currently being discussed for some BPEL engines.

In [30], BPEL is proposed as a suitable language to model the internal state of the resources published by RESTful Web services. To do so, BPEL scopes are used to represent different states of a resource and POST requests trigger the transition between different scopes/states. GET, PUT, and DELETE are mapped to `<onMessage>` event handlers that access, update and clear the values of the BPEL variables declared within the currently active scope. The XPath language embedded in BPEL `assign` activities is extended with functions to compute URI addresses. Unlike the extensions presented in this paper, the resulting "resource-oriented" BPEL does not support the invocation and the composition of external RESTful Web services, but only the publishing of a BPEL process as a resource (or, more exactly, the implementation of a resource state transition logic using BPEL).

BPEL^{light} [19] is an attempt to remove the tight coupling between BPEL and WSDL by identifying the BPEL activities that are closely dependent on WSDL abstractions and subsuming them with a generic messaging construct (the `<interactionActivity>`). We take a similar, but less generic, approach, that introduces a specific set of *resource-oriented* activities to provide native and direct support for the interaction with RESTful Web services.

The idea of a RESTful engine API to access the execution state of workflow instances has been described in [34]. In this paper we provide explicit language support to control which parts of a process becomes exposed through a similar kind of API.

Bite [31] (or the IBM Project Zero assembly flow language) can be seen as a BPEL with a reduced set of activities specifically targeting the development of composite Web application workflows. As in BPEL for REST, the language supports the invocation of RESTful Web services and the corresponding runtime allows to automatically publish processes as resources. Unlike BPEL for REST, the Bite language does not give a direct representation of the REST interaction primitives, as those are condensed within a single `<invoke>` activity, which – as opposed to the one from BPEL – can be directly applied to a URI. Also, with Bite it is not possible to dynamically declare resources from within a process, so that clients may access their state under different representations in compliance with the REST uniform interface principle (e.g., the PUT verb is not supported). Still, the `<receive-reply>` activity in Bite can seen as a form of the combination `<resource><onPost>` in BPEL for REST, since it makes processes wait for client POST requests on a particular URI.

Within the Apache Orchestration Director Engine (ODE) project, a wiki-based discussion regarding RESTful BPEL extension has been recently started[3]. The proposed extension overrides the semantics of existing BPEL activities (i.e., `<invoke>`) to handle the invocation of RESTful Web services. Non-standard attributes are introduced to store the required metadata and bindings to URIs. The ability of declaring and instantiating resources is provided through extensions of the `<onEvent>` and `<receive>` activities. It is worth noting that this solution does not follow the one based on the WSDL 2.0 HTTP binding presented in Section 2. In this paper we proposed a different approach that clearly separates the RESTful activities from the standard ones used to interact with WSDL-based services.

8 Conclusion

This paper contributes to the ongoing discussion on how to best capture the notion of stateful resource in Web service composition languages originally based on the concepts of business processes and of message-based service invocation. It focuses on the research problems that stem from the interaction between two current emerging technologies: WS-BPEL and RESTful Web services. Given their lack of formally described interfaces and the possibility of not always using XML messages, RESTful Web services are challenging to compose through the WSDL-based invocation abstractions required by WS-BPEL. The paper presents in detail using the classical "loan application" example a new extension to the BPEL standard with the goal of providing native support for the composition of RESTful Web services. The extension turns the notion of "resource" and the basic RESTful interaction primitives (GET, POST, PUT, and DELETE) into first class language constructs. With these, a BPEL process can directly interact and manipulate the state of external resources and declaratively publish parts of its state through a RESTful Web service API.

Acknowledgements

This work is partially supported by the EU-IST-FP7-215605 (RESERVOIR) project. The author would also like to thank Domenico Bianculli and the anonymous reviewers for their insightful comments.

References

1. Fielding, R.: Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
2. Vinoski, S.: Serendipitous reuse. IEEE Internet Computing 12(1), 84–87 (2008)

[3] Linked from `http://ode.apache.org/bpel-extensions.html`, last checked on June 13th, 2008.

3. O'Reilly, T.: REST vs. SOAP at Amazon (April (2003), http://www.oreillynet.com/pub/wlg/3005
4. Programmable Web: API Dashboard (2007), http://www.programmableweb.com/apis
5. Fielding, R.: A little REST and Relaxation. The International Conference on Java Technology (JAZOON07), Zurich, Switzerland (June 2007), http://www.parleys.com/display/PARLEYS/A%20little%20REST%20and%20Relaxation
6. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly, Sebastopol (2007)
7. Laskey, K., Hgaret, P.L., Newcomer, E., (eds.): Workshop on Web of Services for Enterprise Computing, W3C (February 2007), http://www.w3.org/2007/01/wos-ec-program.html
8. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): generic syntax. IETF RFC 3986 (January 2005)
9. Florescu, D., Gruenhagen, A., Kossmann, D.: XL: An XML programming language for Web service specification and composition. In: Proc. of the 11th International World Wide Web Conference (WWW 2002), Honululu, Hawaii, USA (May 2002)
10. Crockford, D.: JSON: The fat-free alternative to XML. In: Proc. of XML 2006, Boston, USA (December 2006), http://www.json.org/fatfree.html
11. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. Big Web Services: Making the right architectural decision. In: Proc. of the 17th World Wide Web Conference, Beijing, China (April 2008)
12. Haas, H.: Reconciling Web services and REST services. In: Proc. of ECOWS 2005, Växjö, Sweden, Keynote Address (November 2005)
13. OASIS: Web Services Business Process Execution Language (WSBPEL) 2.0 (2006)
14. Assmann, U.: Invasive Software Composition. Springer, Heidelberg (2003)
15. IBM, SAP: WS-BPEL Extension for Sub-Processes (October 2005)
16. Active Endpoints, IBM, Oracle, SAP: WS-BPEL Extension for People (August 2005)
17. Tan, W., Fong, L., Bobroff, N.: BPEL4JOB: A fault-handling design for job flow management. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 27–42. Springer, Heidelberg (2007)
18. Habich, D., Richly, S., Preissler, S., Grasselt, M., Lehner, W., Maier, A.: BPEL-DT - data-aware extension of BPEL to support data-intensive service applications. In: Emerging Web Services Technology, vol. II. Birkhäuser, Basel (September 2008)
19. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL^{light}. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
20. Chinthaka, E.: REST and Web services in WSDL 2.0 (May (2007), http://www.ibm.com/developerworks/webservices/library/ws-rest1/
21. Pasley, J.: Using Yahoo's REST services with BPEL. Cape Clear (2008), http://developer.capeclear.com/video/httpwizard/httpwizard.html
22. Snell, J.: Resource-oriented vs. activity-oriented Web services. IBM developerWorks (October 2004), http://www-128.ibm.com/developerworks/webservices/library/ws-restvsoap/
23. Vinoski, S.: Putting the "Web" into Web services: Interaction models, part 1: Current practice. IEEE Internet Computing 6(3), 89–91 (2002)
24. Vinoski, S.: Putting the "Web" into Web services: Interaction models, part 2. IEEE Internet Computing 6(4), 90–92 (2002)

25. Wikipedia: Mashup (web application hybrid), http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)
26. Maximilien, M., Nielsen, D., Tai, S. (eds.): 1st International Workshop on Web APIs and Services Mashups (September 2007)
27. Schrenk, M.: Webbots, Spiders, and Screen Scrapers. No Starch Press (2007)
28. Hadley, M.J.: Web Application Description Language (WADL) (2006), http://wadl.dev.java.net/
29. Barros, A., Dumas, M., ter Hofstede, A.H.: Service interaction patterns. In: Proc. of the 3rd International Conference on Business Process Management. LNCS, vol. 3694. Springer, Heidelberg (2005)
30. Overdick, H.: Towards resource-oriented BPEL. In: 2nd ECOWS Workshop on Emerging Web Services Technology (November 2007)
31. Curbera, F., Duftler, M., Khalaf, R., Lovell, D.: Bite: Workflow composition for the web. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. Springer, Heidelberg (2007)
32. Dustdar, S., Schreiner, W.: A survey on web services composition. International Journal of Web and Grid Services (IJWGS) 1(1), 1–30 (2005)
33. Pautasso, C., Alonso, G.: From Web Service Composition to Megaprogramming. In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 39–53. Springer, Heidelberg (2005)
34. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing web services choreography standards - the case of REST vs. SOAP. Decision Support Systems 40(1), 9–29 (2005)

Scaling Choreography Modelling for B2B Value-Chain Analysis

Thomas Hettel[1,2], Christian Flender[1], and Alistair Barros[2]

[1] Faculty of Information Technology,
Queensland University of Technology,
Brisbane, Australia
t.hettel@qut.edu.au, c.flender@qut.edu.au
[2] SAP Research CEC Brisbane, Australia
t.hettel@sap.com, alistair.barros@sap.com

Abstract. The modelling of B2B scenarios focuses on conversations between key partners to establish a common business context for their collaboration. With the prevalence of Web services, attention has turned to service choreographies as a means of message exchange ordering between collaborating participants, from a global (or shared) understanding. As such, the message ordering in a choreography model can then be used to determine the message ordering behaviour of each participant's process. In this paper, we extend the suitability of choreography modelling for the early phase of analysis, where the participants and the nature of interactions develops under the flux of requirements acquisition. In particular, we develop a structural view of interactions and stepwise refinement, leading to behavioural considerations, reminiscent of classical techniques. In addition, we introduce contextualisation of intent behind message exchanges in the form of speech acts. This, we show, can be used to automatically detect conflicts in conversations, in the business sense, namely negotiation or provision breakdowns - prior to technical implementations of choreographies. Model abstraction and refinement is based on Semantic Object Model (SOM), and a mapping to the Business Process Modelling Notation (BPMN) is shown.

1 Introduction

The modelling of interactions between collaborating parties in B2B domains is crucial across the early stages of information systems analysis to detailed design. At the outset, the key parties and their shared interactions, constituting a common business context, needs to be established. This provides the scope and structure of the organisational system being modelled, out of which further details can be refined, as analysis unfolds and design sets in. Ultimately, the interoperability of services underpinning B2B collaborations should be traceable to high-level requirements and the business context, and be free of message exchange conflicts.

With the prevalence of Web services technologies, attention has turned to service composition languages, proposed through the standards arena, as the basis for model-driven development of service-based applications. The Web Services Business Process Execution Language (WS-BPEL or BPEL) proposal allows message exchanges to be captured within process-based descriptions of individual services. Once completed, the collaboration across different services, derived through their locally specified message exchanges, can be verified for consistency. Such an approach proves unwieldy the larger the number of collaborating parties and the more complex the service interactions.

Accordingly, the need for service *choreography* modelling has emerged to bring into a global ("birds-eye") perspective, message exchanges across all collaborating services. As seen in Web Services Choreography Definition Language (WS-CDL), ordered message exchanges across *all* collaborating services are modelled as first class elements in a choreography model without the need for exposing orchestration logic of the individual services. As such, choreography models introduce a particular composition context for reuse of existing services or implementing new ones. This is a strong characteristic of analysis and design specific to the service-oriented setting, where global conversation protocols derive the required behaviour of collaborating services, which in turn needs to be adapted to their preexisting behaviour [1,2,3,4].

For the earlier stages of analysis and design, however, current choreography modelling languages pose major limitations. Efforts during this stage are concerned with implementation issues of collaborations rather than the organisational context - the *intent* - in they are enacted. Since value chains in practice feature tens to hundreds of stakeholders, the identification of collaborating parties is iterative. Some parties come to the fore in light of the operations of others. Other parties fade into the background as their operations are seen as ancillary. Only when the system landscape stabilises around common functions can detailed modelling of collaborations proceed.

Classical analysis and design techniques manage the complexity of models and their *organisational embedding* variously. Structured techniques support stepwise refinement of models, allowing models to be captured at different levels of complexity. They rely on abstraction mechanisms to refine organisational collaborations into technical service interactions. The lowest level of structural descriptions could be transformed into service composition where orchestration could be detailed. Action-oriented techniques [5,6] were proposed to explicitly contextualize business intent of collaborations only informally addressed in structured techniques. Speech acts formalise the social meaning of collaborations, e.g. initial requests, promises or obligations to act, and ensuing action. Consequently, they formalise negotiation patterns inherent in reciprocal collaborations, enhancing modelling and validation.

This paper harnesses different strengths from classical techniques to extend the suitability of choreography modelling beyond detailed and technically focussed service interactions. It uses the Semantic Object Model (SOM) [7] which allows stepwise refinement of choreography models, in particular actors and their collaborations. At detailed levels, behavioural models are generated from refined structural models, with detailed orchestration logic defined for each role.

We demonstrate how contextualised collaboration together with behavioural descriptions can be used to improve analysis of choreography models with respect to inconsistencies of negotiations. We propose conversational analyses beyond the classical techniques of prescriptive orchestrations enforcing local behaviour. Finally we show that SOM orchestrations can be mapped into BPMN, a widely used language for modelling business processes. As part of the mapping, we demonstrate how racing interactions can be detected from structural models and be managed in behavioural models.

In the next section, we refer to related attempts to choreography design, analysis and implementation. Both action-oriented modelling as well as notations closer to implementation have their strengths and weaknesses. Based on shortcomings identified, in Section 3, we introduce SOM by means of a logistics reference model suitable to devise breakdown analyses in Section 4. Here, we formalise potential threats or breakdowns of business cases in complex conversational scenarios. Having refined the logistics reference model down to an appropriate level of detail, we map its behavioural view to BPMN.

2 Related Work

Choreography modelling can be divided into two classes according to their proximity to implementation. Action-oriented languages draw from speech act theory [8,9] and conversational analysis [10] in order to model the coordination of actors, communication acts and commitments. Organisations are considered as networks of interrelated conversations. Similarly, emerging standards for choreography modelling derive their constructs from networks of interrelated Web services [11]. Accordingly, design and analysis techniques differ with respect to implementation. However, it is the sound integration of design and analysis techniques across all stages of information systems development that most approaches aim at.

In line with the specification of interrelated conversations, actors communicate or speak by uttering words or sentences. According to speech act theory, *intents* or so called illocutionary acts complement utterances (e.g. make offer, request quote). Additionally, propositional acts refer to what it is being talked about (e.g. an offer of a product, a service to be delivered in the future). Perlocutionary acts trigger subsequent communication acts so as to spread conversational relations towards more complex networks of interrelated actions.

For instance, DEMO [6] leverages speech acts to model interactions between partners and divides conversations into three main stages: offer, execution and result. The offer phase is made up of two intentional acts, namely request, i.e. initiator *requests* something from executor, and promise, i.e. executor *promises* to provide a service. In the execution phase, executor enacts what has been promised, whereupon he *states* the fulfilment of a promise in the result phase. Initiator then *accepts* the execution so far it matches the promise. DEMO supports the gradual refinement of conversations via transaction patterns like request, promise, state and accept. However, the gradual refinement or replacement of acts is only one side of the coin. As networks of conversations evolve with new participating partners inseparably entangled within the transactional network, *actor decomposition* becomes salient. DEMO lacks the guided decomposition of actors *and* transactions toward implementable models.

Another approach uses Action Diagrams to model conversations [5]. Action Diagrams explicitly refer to the development of implementable models. Here, actions, performers, information flows and material flows are covered. It is shown that requirements specification from the perspective of the actors involved leads toward more accurate and relevant implementations. Rittgen (2006) follows this line of argument and specifies a language mapping framework using DEMO and UML to support the transition from actor networks to system models having a closer proximity to implementation [12]. Other prominent action languages are Conversation for Action [13] or Action Workflow loops [14].

As opposed to action-oriented modelling, traditional approaches are derived from machine languages and therefore have a closer proximity to implementation. Much effort has been put into these languages to provide sensible abstractions from less relevant details.

One of these languages is Let's Dance [15]. Here, interactional relations between actors are not intentional acts rather than simple message exchanges. Taking the view of a global observer, the modeller defines dependencies between interactions, which may be preceded or inhibited by other interactions. Unfortunately, not all specified dependencies can be translated into local behaviour [16]. This is known as the local enforceability problem. Behaviour of each actor is derived from the interaction model and therefore, in the first place, it is assumed to be global, irrespective of its actual point of departure which is always an actor. However, choreography, as natural language, describes the coordination of behaviour of autonomous, distributed and loosely-coupled actors and not some global behaviour [17,18].

Drawing from the ideas of Let's Dance, Decker et al. (2007) propose interaction modelling for BPMN [19]. Here, the idea is to use standard BPMN graphical elements, such as gateways, start, end and intermediate events to compose interactions (depicted as message events) between actors to a global process. However, extending BPMN in this way also runs into the local enforceability problem. Moreover, its verbose notation and the lack of abstractions or refinements prohibit its use in larger scenarios.

It is beyond implementation that choreography models can become very complex with many actors being involved interacting in many different ways. To make these complex choreography scenarios tractable, Barros et al. (2007), propose three different abstraction layers, so-called viewpoints [20]. However, each viewpoint comes with its own modelling language and paradigm and the steps necessary to get from one viewpoint to another are rather big. No guidance is given to perform this refinement. Moreover, it is arguable that fix abstraction layers serve well in all cases. They may be too coarse grained in larger scenarios.

Drawing from systems theory and its recursive definition of systems, Quartel et al. (2004) propose an approach that supports the guided derivation of complex multi-layered networks of services [21]. This allows complex systems to be gradually decomposed into smaller systems by revealing their inner structure. This approach is very powerful as it can be applied from the level of value chains down to implementation. Moreover, it provides guidelines supporting the modeller to gradually refine systems.

In summary, there is a gap between action-oriented approaches [5,14,6,12,13] and traditional modelling techniques [15,19,20,21] having a closer proximity to implementation. More recently, this has been recognised [12,11], however an integrated approach to the design, analysis and implementation of choreographies is still missing. Action-oriented modelling puts much emphasis on human actors and their social relations, thus understanding information systems as vehicles to support human interactions [5]. However, in the end, implementation support becomes salient enforcing the shift from conversational networks of interrelated actors to networks of collaborating Web services. Classical approaches to choreography modelling look at information systems from a more technical perspective presupposing choreography to be a special case of orchestration. This maintains the applicability of many sophisticated analysis techniques but also introduces new problems such as local enforceability. High-level concepts such as speech acts are not considered, neglecting that in the end it is an human actor who commits to the provision or consumption of services. An approach is needed that bridges both worlds supporting the guided design and analysis of collaborating business partners down to implementation specific process models.

3 The Semantic Object Model

To overcome the gap between business analysis and systems design, the Semantic Object Model (SOM) [7] was developed in the 1980s and 1990s and provides a methodology to address recent problems facing choreography. Its core is an enterprise architecture that divides a complex system into the three main layers enterprise plan, business process model and resources (human and machine actors). In bridging enterprise plan and resource layer, the business process model gets decomposed into components, so-called business objects, without losing track of the architecture as a whole. Interactions between business objects define conversational relations (transactions) from a structural perspective. From a behavioural view, business objects change system states according to their relative involvement in conversations. As opposed to structure preserving task decompositions (e.g. the creation of subprocesses), negotiation and feedback-control coordination principles guide modellers in the gradual refinement of business objects and transactions toward multi-layered models so as to reduce overall complexity. Synchronisation mechanisms track changes of model layers and keep the overall architecture in sync. Maintaining such multiple layers enables assignments of resources (e.g. legacy applications) with diverse granularities. Hence, SOM provides a holistic framework for application integration. Due to its conceptual similarity to BPMN, SOM is highly susceptible to put established choreography models within the broader social context of interacting parties.

The next sections will introduce SOM on the basis of a case study derived from VICS's logistics reference model[1].

3.1 Scenario

To replenish low stocks a *Buyer* places an order with a *Supplier* and provides necessary delivery details identifying the *Consignee* (delivery point) which may be a distribution centre or a store. The *Buyer* then notifies the *Consignee* about the expected delivery while the *Supplier* contacts a *Shipper* passing on order details and the destination. The *Shipper* then establishes a shipment with a *Carrier*. This shipment is transported to its final destination and passed through *Customs and Quarantine*. From there goods are collected and dispatched by a *Consolidator* to the *Consignee*.

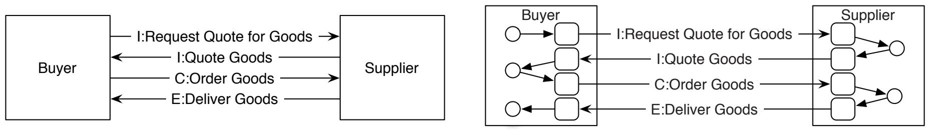


Fig. 1. Layer 1: Initial structural and behavioural view

3.2 Modelling the Scenario with SOM

In SOM, modelling starts at the enterprise plan level. On this abstraction layer only a minimal number of actors is involved, namely *Buyer* and *Supplier* (cf. Fig. 1). Both are equal partners having to engage in negotiations before any services can be provided or consumed (the so-called negotiation principle). Such an interaction has to follow three sequential steps: (1) *Initialising (I)* where both actors exchange information about the provided service; (2) *Contracting (C)* where both actors negotiate the terms of the service delivery/consumption; and finally (3) *Enforcing (E)* where the negotiated services are provided/consumed. *I* and *C* may be optional depending on whether both objects already know each other and whether a basic agreement has been established between both. Continuing with the scenario, *Buyer* uses an *I* transaction to get a quote from *Supplier* for a specific product he is interested in purchasing. With the *C* transaction the *Buyer* places an order based on the previous quote which is confirmed. In the next step, *Buyer* and *Supplier* commit to provide and consume a service with respect to the negotiated terms. This service, namely the delivery of the ordered goods, is enforced using the *E:Deliver Goods* transaction.

In terms of speech act theory, *I, C* and *E* identify the type of the illocutionary act of the performed speech act. Transactions have names that consist of two parts: (1) a verb identifying the intention (illocutionary act), e.g. *order*, *request*, *confirm*, etc.; (2) a noun identifying what is being talked about (propositional content), e.g. *goods*, *delivery*, etc.

[1] http://www.vics.org/committees/logistics/LogisticsModel.pdf

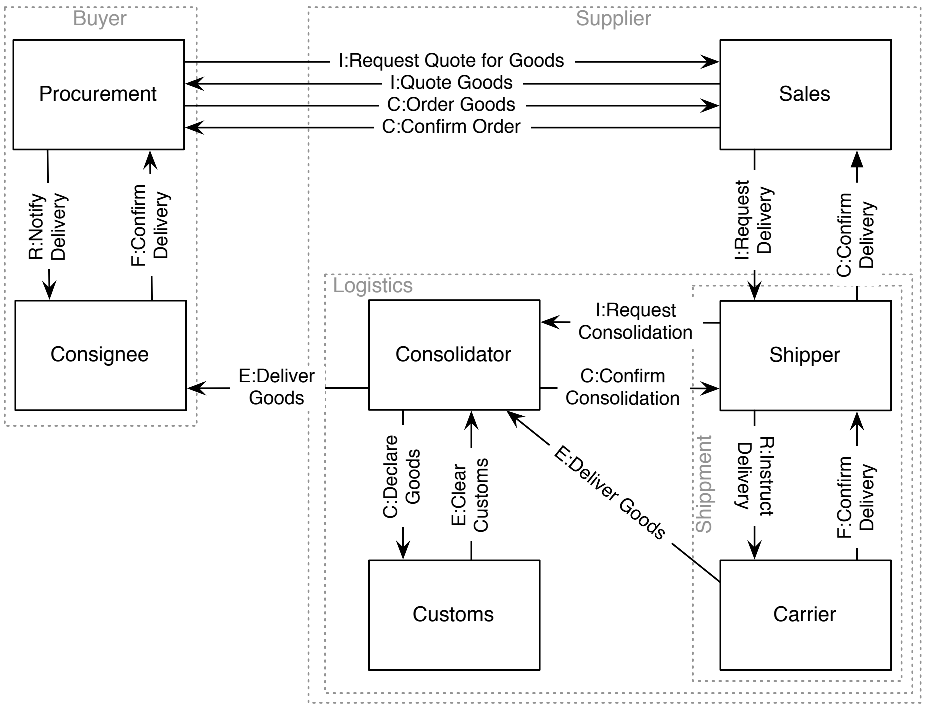


Fig. 2. Layer 4: Detailed structural view, derived from Fig. 1

To complement the coordination shown in the structural view, SOM provides a behavioural view that specifies the sequence of transactions (cf. Fig. 1). Here, the modeller specifies how each actor acts and reacts with respect to transactions. Tasks (rounded rectangles) can either be used to model incoming or outgoing transactions and may have any number of pre or post events (circles). A transaction always connects two tasks belonging to different business objects. The modeller may annotate a task with *XOR* ('X') or *AND* ('&') if it is known that either one or all pre events are necessary to trigger a task. Similary, tasks can be annotated to show that only one or all post events are triggered. With regard to implementation, a heuristic rule is to specify more complex control-flow behaviour once a system is sufficiently detailed.

To add more details to this high-level model, objects can be decomposed to reveal further actors and their coordination using the recursive definition of systems as proposed by systems theory. To get to the lowest layer as depicted in Fig. 2 a number of decompositions had to be applied[2]. The dotted boxes give hints with respect to the decomposition process. On the left hand side, *Buyer* was decomposed into *Procurement* and *Consignee* interacting according to the feedback-control principle, the second interaction principle besides the

[2] Due to lack of space not all layers and all decomposition steps can be shown.

negotiation discussed before. Here, the management object *Procurement* regulates (R) the operational object *Consignee* by sending an advice to receive goods, whereupon *Consignee* replies (F for feedback) by confirming the receipt of the delivery. On the right hand side, *Supplier* was first decomposed into *Sales* and *Logistics*. Furthermore, *Logistics* was decomposed into *Shipper*, *Carrier*, *Consolidator* and *Customs*. Accordingly, transactions initially defined between *Buyer* and *Supplier* connect *Procurement* and *Sales* as well as *Consolidator* and *Consignee*. This does not contradict the previous models. *Buyer* and *Supplier* have not ceased to exist, only their inner structure has been revealed.

Similar to the decomposition of business objects, transactions (and tasks respectively) can be decomposed as well. This is shown in Fig. 3 which is a refinement of Fig. 1. Here, *C:Order Goods* was decomposed to reveal the actual negotiation process. This was achieved by decomposing *C:Order Goods* sequentially (1) into the following sub-transactions.

- *C:Propose Delivery Details*, where *Procurement* proposes details such as date, quantity, quality and price.
- *C:Confirm Or Propose Alternative Details*, where *Supplier* confirms the details or proposes alternative details.

In a sub-sequential step (2), *C:Confirm Or Propose Alternative Details* was in turn decomposed into the parallel sub-transactions *C:Propose Alt Del Details* and *C:Confirm Del Details*. Here, *Supplier* has the choice between one of the aforementioned transactions. In turn, *Procurement* has a choice between accepting the alternative details or proposing new details again.

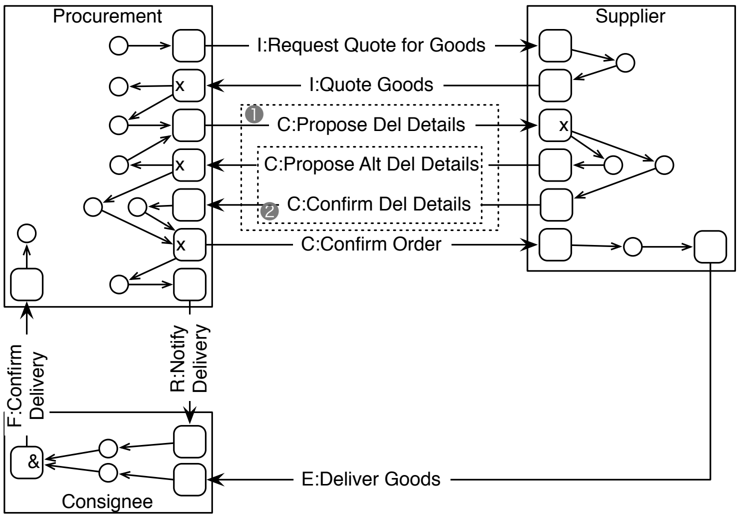


Fig. 3. Layer 2 behavioural view showing the decomposition of Buyer into Procurement and Consignee as well as the decomposition of C:Order Goods

3.3 Summary

As seen in the case study above, the gradual refinement process is a very powerful tool, supporting the modeller in managing complex interaction scenarios with many different actors. Arbitrary layers can be utilised to provide a consistent and reasonable abstraction of the system that facilitates understanding and navigating the system.

4 Breakdown Analysis

Choreography models can become complex with many partners interacting in many different ways to achieve a common goal. Such a complex system can suffer many different kinds of breakdowns. Apart from deadlocks and livelocks and the like that can be identified using classical workflow analysis techniques [22], there are other breakdowns on a higher level. Especially in choreography modelling where autonomous and independent partners interact, participants may choose not to take part and withdraw their support at some stage. This is most likely during negotiations where each participant may decide to withdraw for various reasons at any time. However, after a successful negotiation, both partners are committed to providing a service or consuming it. In most cases such a commitment cannot be easily withdrawn without attracting financial penalties or legal charges. In front of this backdrop, partners engaged in negotiations may want to run the following analysis:

– Are there any external factors that may cause this negotiation to fail? In other words: what are the sub-sequential negotiations necessary to successfully close this deal? (Negotiation Breakdown Analysis)
– Are there any external factors that may prevent a committed service from being provided? (Provision Breakdown Analysis)

Particularly, the second question may be of interest as once an agreement is made, products and services have to be delivered or consumed. A failure in doing so may result in legal charges against the violator who may also be liable to pay compensation. Such a situation may have been introduced by inapt modelling and therefore needs to be rectified. If it is, for some reason, not possible to model the choreography differently, the affected actors should at least be aware of such possible situations and should have a risk mitigation strategy at hand to reduce the danger or prevent such situations from happening altogether. The next sections will elaborate on these analysis techniques in more detail.

4.1 Prerequisite

In complex scenarios, model queries as needed for breakdown analyses require precise definitions. Therefore, we formalise a SOM model and the conversations it is meant to express.

A SOM Model. is defined to be a directed graph consisting of three types of nodes: Tasks T, Events E and Business Objects O and two kinds of edges: Transactions $Trans$ and pre/post event relations $EventAssoc$.

$$\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S) = (T \cup E \cup O, Trans \cup EventAssoc)$$

Furthermore, let $bo : T \to O$ be the function that returns the business object in which a task is contained, let $\tau : Trans \to \{\mathsf{I}, \mathsf{C}, \mathsf{E}, \mathsf{R}, \mathsf{F}\}$ be the function that returns the type of a transaction and sg,$jg : T \to \{\mathsf{xor}, \mathsf{and}\}$ be the functions that return the type of the implicit split (sg) or join gateway (jg) of a task.

A Conversation. is composed of all transactions that were derived from an initial ICE or RF transaction between two actors. On a lower layer, a conversation may span several actors. Using the trace log of the refinements, different transactions can be combined to one conversation. For instance, traces can be used to group the transaction *E:Deliver Goods* between *Consolidator* and *Consignee* and the other transactions between *Procurement* and *Sales* together to one conversation as they all originate from the same *ICE* (cf. Fig. 1, Fig. 2).

Let the trace log be a directed graph $\mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$ with $\mathcal{V}_T = Trans$ and $\mathcal{E}_T = \{\langle v_1, v_2 \rangle\}$ $v_1, v_2 \in \mathcal{V}_T$. When decomposing a transaction t into a number of new transactions $t_1, \ldots, t_n$, edges $\langle t, t_i \rangle$ $i \in \{1, \ldots, n\}$ are added to $\mathcal{E}_T$.

Using the trace log $\mathcal{G}_T$, an *ICE*-conversation $\mathcal{C}_{ice}$ is defined as the set of transactions that are leafs in the trace log and can be reached by following the trace from either i, c or e:

$$\mathcal{C}_{ice} = \{t | (t \in \{i, c, e\} \vee \exists p = \langle x, \ldots, t \rangle \in \mathcal{E}_T, x \in \{i, c, e\}) \wedge \nexists p' = \langle t, t' \rangle \in \mathcal{E}_t\}$$

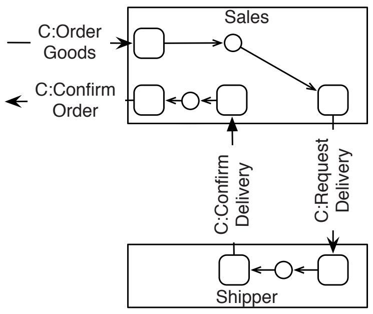


Fig. 4. Relevant part of the behaviour involving *Sales* and *Shipper* as shown in Fig. 2

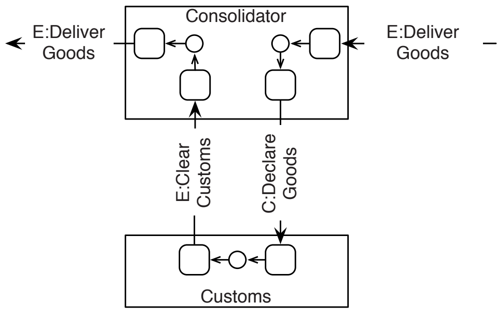


Fig. 5. Relevant part of the behaviour involving *Consolidator* and *Customs* as shown in Fig. 2

4.2 Negotiation Breakdown

Requirements for successful negotiations may be other sub-sequential negotiations necessary to arrange additional services needed to provide the overall service. As choreographies model the collaboration of loosely coupled and autonomous actors, participants may withdraw from negotiations at any time, therefore causing it to fail. Such failures may cascade through the model and cause encompassing negotiations to fail as well – a so-called *negotiation breakdown*. A possible negotiation breakdown may be caused by *Shipper* (cf. Fig. 4 and Fig. 2), as an unsuccessful negotiation between *Sales* and *Shipper* may impact on the negotiation between *Procurement* and *Sales* and may cause it to fail, too.

The negotiation breakdown analysis leverages SOM's typed transactions to find sub-sequential negotiations between third parties that are encompassed in another negotiation. Therefore, the following set theoretic expressions identify the set of partners that may be involved in a negotiation breakdown N with respect to a conversation $\mathcal{C}_{ice}$.

A negotiation breakdown $N_{\mathcal{C}_{ice}}$ with respect to the conversation $\mathcal{C}_{ice}$ may be caused by business objects $o_1 = bo(t_1)$, $o_2 = bo(t_2)$ engaged in a negotiation $n = [t_1, t_2]$, where t_1, t_2 are tasks, that is not part of the conversation $\mathcal{C}_{ice}$. This negotiation n must be reachable from the first C-transaction $c' \in first^{\mathsf{C}}_{\mathcal{C}_{ice}}$ in $\mathcal{C}_{ice}$ and must lead to the first E-transaction $e' \in first^{\mathsf{E}}_{\mathcal{C}_{ice}}$ in $\mathcal{C}_{ice}$. This way n can impact on the negotiation in $\mathcal{C}_{ice}$ and cause it to fail as well. To denote a connection between nodes v and v' regard of direction $[v, v']$ is used.

$$N_{\mathcal{C}_{ice}} = \{d | c' \in first^{\mathsf{C}}_{\mathcal{C}_{ice}}, e' \in first^{\mathsf{E}}_{\mathcal{C}_{ice}}, [t_1, t_2] \in Trans_{\mathsf{C}}(c', e'),$$
$$[t_1, t_2] \notin \mathcal{C}_{ice}, d = \{bo(t_1), bo(t_2)\}\}$$

where the following auxiliary sets and functions were used: $first$ and $last$ determines the first respectively the last transaction of a type x within a conversation $\mathcal{C}_{ice}$. $Trans_x(t', t'')$ determines all transactions of a type x that can be reached from t' and lead to t''.

$$\begin{aligned} first^x_{\mathcal{C}_{ice}} &= \{c \in \mathcal{C}_{ice} | \tau(c) = x, \nexists c' \in \mathcal{C}_{ice} : \tau(c') = x, \exists \langle c', \ldots, c\rangle \in \mathcal{E}_S\} \\ last^x_{\mathcal{C}_{ice}} &= \{c \in \mathcal{C}_{ice} | \tau(c) = x, \nexists c' \in \mathcal{C}_{ice} : \tau(c') = x, \exists \langle c, \ldots, c'\rangle \in \mathcal{E}_S\} \\ Trans_x(t', t'') &= \{t \in Trans | \tau(t) = x, t = [v_i, v_{i+1}] \in [t', v_1, \ldots, v_n, t''] \in \mathcal{E}_S\} \end{aligned}$$

4.3 Provision Breakdown

Once, two actors have agreed upon consumption and delivery, the service has to be provided and consumed. However, it may happen that *after* committing to a service provision additional negotiations for supplementary services are required. If any of these negotiations fail, it may not be possible to provide the promised service, causing a *provision breakdown*. For instance, such a breakdown may be caused by *Consolidator* and *Customs* in the example depicted in Fig. 2. Fig. 5 shows that *Consolidator* does only talk to *Customs* *after* it received the goods from *Carrier*. If customs cannot be cleared for these goods, then the promised delivery cannot be made. This may pose a serious problem to other partners as they may be held liable to pay compensation for violating the contract. This scenario may be the result of unapt modelling and therefore needs to be rectified by turning a possible provision breakdown into a possible negotiation breakdown. However, it may not always be possible to model the choreography differently to avoid such situations. Customs cannot be cleared upfront without having the actual delivery inspected. In this case the affected actors may consider a risk mitigation strategy to counter such scenarios.

In order to identify potential provision breakdowns the following set theoretic expression can be used. Provision breakdown $P_{\mathcal{C}_{ice}}$ with respect to the conversation $\mathcal{C}_{ice}$ may be caused by business objects $o_1 = bo(t_1)$, $o_2 = bo(t_2)$ engaged in a negotiation $n = [t_1, t_2]$ that is not part of the conversation $\mathcal{C}_{ice}$. If this negotiation n is reachable from the last C-transaction $c' \in last^{\mathsf{C}}_{\mathcal{C}_{ice}}$ in $\mathcal{C}_{ice}$ and leads to the last E-transaction $e' \in last^{\mathsf{E}}_{\mathcal{C}_{ice}}$ in $\mathcal{C}_{ice}$, n can prevent the provision of the service in $\mathcal{C}_{ice}$ and therefore may cause a provision breakdown:

$$P_{\mathcal{C}_{ice}} = \{d | c' \in last^{\mathsf{C}}_{\mathcal{C}_{ice}}, e' \in last^{\mathsf{E}}_{\mathcal{C}_{ice}}, [t_1, t_2] \in Trans_{\mathsf{C}}(c', e'), \\ [t_1, t_2] \notin \mathcal{C}_{ice}, d = \{bo(t_1), bo(t_2)\}\}$$

5 Mapping to BPMN

So far, it has been shown how a detailed model can be derived through repeated decompositions of business objects and transactions. However, one of the main benefits of SOM is its potential to derive implementation specific models. Therefore, a formal mapping from SOM's behavioural view to BPMN is presented that allows for the automatic generation of behavioural interfaces, the point of departure for an implementation.

5.1 Prerequisite

In addition to the SOM model defined in 4.1, we also define a BPMN diagram to be a directed graph $\mathcal{G}_B$ consisting of the nodes Pools P, Activities A, Gateways G and edges sequence flow $SeqFlow$ and message flow $MsgFlow$.

$$\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B) = (P \cup A \cup G, SeqFlow \cup MsgFlow)$$

Gateways are further disjointly subdivided into XOR gateways G_X, parallel gateways G_P and event-based gateways G_E. $G = G_X \cup G_P \cup G_E$. Moreover, let $pool : (A \cup G) \rightarrow P$ be the function that assigns activities or gateways to the pool in which they are contained.

5.2 Mapping Rules

For the following mapping two auxiliary sets of tuples linking SOM tasks to BPMN activities or gateways are used: $linkTo, linkFrom \subseteq T \times (A \cup G)$. The following mapping is based on predicate logic expressions, where the left-hand side matches a pattern in the SOM model and the right-hand side asserts the existence of the corresponding pattern in the BPMN model. A number of injective functions[3] f_X is used to relate SOM to BPMN elements.

The first rule creates a BPMN activity for each SOM task and a pool for each business object and links the activity to the pool. The other rules deal with the implicit gateways in SOM which have to be made explicit in BPMN. Here the auxiliary sets $linkTo$ and $linkFrom$ are used to keep track of where sequence flow has to be attached to, i.e., the activity or its preceding or succeeding gateway.

$$\begin{aligned}
t \in T, o \in O, bo(t) = o &\Rightarrow f_{t2a}(t) = a \in A, f_{bo2pool}(o) = p \in P, p = pool(a)\\
t \in T, \neg split(t) &\Rightarrow f_{t2a}(t) = a \in A, (t,a) \in linkTo\\
t \in T, \neg join(t) &\Rightarrow f_{t2a}(t) = a \in A, (t,a) \in linkFrom\\
t \in T, xorSplit(t) &\Rightarrow f_{t2a}(t) = a \in A, f_{t2sg}(t) = g \in G_X, (t,g) \in linkFrom,\\
&\quad\ \langle a, g\rangle \in SeqFlow\\
t \in T, andSplit(t) &\Rightarrow f_{t2a}(t) = a \in A, f_{t2sg}(t) = g \in G_P, (t,g) \in linkFrom,\\
&\quad\ \langle a, g\rangle \in SeqFlow\\
t \in T, xorJoin(t) &\Rightarrow f_{t2a}(t) = a \in A, f_{t2jg}(t) = g \in G_X, (t,g) \in linkTo,\\
&\quad\ \langle a, g\rangle \in SeqFlow\\
t \in T, andJoin(t) &\Rightarrow f_{t2a}(t) = a \in A, f_{t2jg}(t) = g \in G_P, (t,g) \in linkTo,\\
&\quad\ \langle a, g\rangle \in SeqFlow
\end{aligned}$$

The second set of rules adds implicit control flow, message flow and data-based gateways.

For each transaction between SOM tasks, a message flow between the corresponding activities must exist.

$$t_1, t_2 \in T, \langle t_1, t_2\rangle \in Trans \Rightarrow \langle f_{t2a}(t_1), f_{t2a}(t_2)\rangle \in MsgFlow$$

For two tasks of the same business object in SOM that are connected via a sequence of transactions and events, a sequence flow edge must connect the corresponding activities.

$$\begin{gathered}
t_1, t_2 \in T, bo(t_1) = bo(t_2), connected(t_1, t_2), (t_1, a_1) \in linkFrom, (t_2, a_2) \in linkTo,\\
\neg(\langle t_1, e, t_3\rangle \in \mathcal{E}_S, connected(t_3, t_2)) \Rightarrow \langle a_1, a_2\rangle \in SeqFlow
\end{gathered}$$

For two tasks of one business object that are (indirectly) connected via an XOR split in another business object, an event-based gateway has to be introduced and linked to the corresponding activities:

$$\begin{gathered}
t_1, t_2, t_3 \in T, bo(t_1) = bo(t_3) \neq bo(t_2), xorSplit(t_2), connected(t_1, t_2),\\
connected(t_2, t_3), (t_1, a_1) \in linkFrom, (t_3, a_3) \in linkTo\\
\Rightarrow \langle a_1, g\rangle \in SeqFlow, \langle g, a_2\rangle \in SeqFlow, f_{t2eg}(t2, bo(t_1)) = g \in G_E
\end{gathered}$$

[3] An actual definition of those functions is not necessary as only their injective nature is of interest.

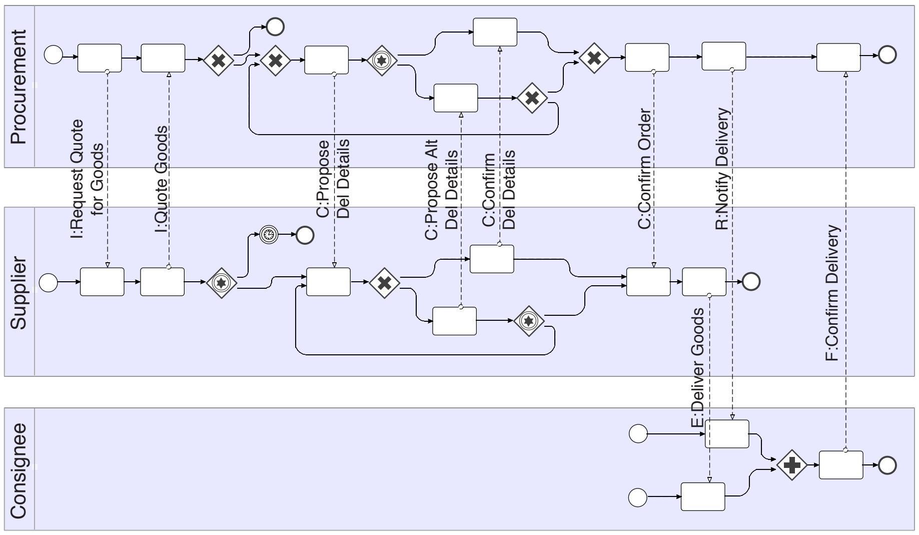


Fig. 6. BPMN diagram generated from the SOM behavioural model Fig. 3

Moreover, the following predicates were used and need to be defined:

$$\exists p_1 = \langle t, t_1, \ldots t_n, t' \rangle \in E_S : \forall t_i : \neg split(t_i), bo(t_i) \neq bo(t), i \in \{1, \ldots, n\}$$
$$\Rightarrow connected(t, t')$$

$$\exists \langle t, e \rangle, \langle t, e' \rangle \in EventAssoc, e \neq e' \Rightarrow split(t)$$

$$\exists \langle e, t \rangle, \langle e', t \rangle \in EventAssoc, e \neq e' \Rightarrow join(t)$$

$$split(t), sg(t) = \mathsf{xor} \Rightarrow xorSplit(t)$$

$$join(t), jg(t) = \mathsf{xor} \Rightarrow xorJoin(t)$$

$$split(t), sg(t) = \mathsf{and} \Rightarrow andSplit(t)$$

$$join(t), jg(t) = \mathsf{and} \Rightarrow andJoin(t)$$

This transformation was implemented using the Tefkat model transformation language and the result of running it on the SOM behavioural view depicted in Fig. 3 is shown in Fig. 6. In particular, it shows that racing conditions (event-based gateways) have been derived from the behavioural view and complement each decision (XOR-split) made by a particular partner. Although being implemented, rules concerning start and end events can not be shown due to lack of space.

Eventually, it is this mapping that closes the gap between high-level models and implementation. We started from the level of value chains or conversational networks, refined the model through gradual decompositions and transformed a flat behavioural representation into an executable template (at least as far as control flow is concerned).

6 Conclusion and Future Research

In this paper, we introduced SOM for the design and analysis of choreographies involving several business partners. From the enterprise plan down to a model detailed enough to be translated into BPMN behavioural interfaces, we decomposed business objects and transactions. Engaged in the complex exercise of systems design and analysis, the modeller is guided and supported by a set of decomposition rules.

Due to the nature of autonomous and loosley-coupled actors, breakdowns may occur that cause negotiations to fail, or even worse, prevent the provision of a service despite agreement. Violations may attract financial penalties and even legal charges. Analysis techniques were presented to help modellers becoming aware of possible breakdowns and therefore allowing them to change the model. If revisions are not possible risk mitigation strategies will increase in attractiveness.

A requirement of our analysis was SOM building upon speech acts as reflected in the types of transactions between actors. Intentional acts enable distinctions between negotiation and service provision. Furthermore, a crucial concept utilised in the analysis was the gradual refinement process that allowed individual transactions spread between different partners to be grouped together into a single conversation. Grouping of conversational acts can hardly be achieved with traditional modelling languages such as BPMN as there are no decomposition guidelines. Moreover, understanding choreography models as coordinated behaviour of autonomous actors, and not as some global behaviour, avoids SOM being prone to the local enforceability problem.

Our proposal of a combined top-down and bottom-up approach to choreography modelling will be extended toward higher degrees of flexibility. Ongoing research addresses model synchronisation, in particular it looks at ways of how higher layers of an existing model can be changed with minimal impact on existing models on lower layers. Moreover, future work will investigate the trace log, a side product of the refinement process. Trace logs capture the history of a model whose evolutionary decomposition steps could mirror sophisticated navigation in complex models. New results will continously add to an integrated modelling environment which was initialised with the implementation of a SOM/BPMN editor according to the research presented in this paper.

References

1. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. Int'l Journal of Cooperative Information Systems 13(4), 337–368 (2004)
2. Baïna, K., Benatallah, B., Casati, F., Toumani, F.: Model-Driven Web Service Development. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 290–306. Springer, Heidelberg (2004)
3. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of service-oriented analysis and design (2004), www.ibm.com/developerworks/library/ws-soad1
4. Papazoglou, M., Van Den Heuvel, W.: Service-oriented design and development methodology. Int'l Journal of Web Engineering and Technology 2(4), 412–442 (2006)

5. Agerfalk, P., Eriksson, O.: Action-oriented conceptual modelling. European Journal of Information Systems 13(1), 80–92 (2004)
6. Dietz, J.: The deep structure of business processes. Communications of the ACM 49(5), 58–64 (2006)
7. Ferstl, O.K., Sinz, E.J.: Foundations of Information Systems (in German), 5th edn. Oldenbourg (2006)
8. Austin, J.L.: How to do things with words. Oxford Uni. Press, Cambridge (1962)
9. Searle, J.R.: Speech acts. Cambridge Univ. Press, Cambridge (1969)
10. Sacks, H.: Lectures on Conversation. Blackwell Publishers, Malden (1995)
11. Umapathy, K., Purao, S.: A theoretical investigation of the emerging standards for web services. Information Systems Frontiers 9(1), 119–134 (2007)
12. Rittgen, P.: A language-mapping approach to action-oriented development of information systems. European Journal of Information Systems 15(1), 70–81 (2006)
13. Winograd, T.: A language/action perspective on the design of cooperative work. Human Computer Interaction 3(1), 330 (1988)
14. Denning, P., Medina-Mora, R.: Completing the loops. Interfaces 25(3), 42–57 (1995)
15. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: Fourteenth Int'l Conference on Cooperative Information Systems (CoopIS), Montpellier, France. LNCS, vol. 4275, pp. 145–162. Springer, Heidelberg (2006)
16. Zaha, J.M., Dumas, M., ter Hofstede, A., Barros, A., Decker, G.: Service Interaction Modeling: Bridging Global and Local Views. In: IEEE Int'l Conference on Enterprise Distributed Object Computing (EDOC). IEEE, Los Alamitos (October 2006)
17. Maturana, H., Poerksen, B.: From being to doing: the origins of the biology of cognition. Carl Auer Verlag, Heidelberg (2004)
18. Winograd, T., Flores, F.: Understanding computers and cognition. Ablex Publishing Corp. Norwood, NJ (1986)
19. Decker, G., Barros, A.: Interaction Modeling using BPMN. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 208–219. Springer, Heidelberg (2008)
20. Barros, A., Decker, G., Dumas, M.: Multi-staged and Multi-viewpoint Service Choreography Modelling. In: Proceedings of the Workshop on Software Engineering Methods for Service Oriented Architecture (SEMSOA), Hannover, Germany. CEUR Workshop Proceedings, vol. 244 (May 2007)
21. Quartel, D., Dijkman, R., van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 1–10. ACM, New York (2004)
22. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)

Evaluation of OrViA Framework for Model-Driven SOA Implementations: An Industrial Case Study

Sebastian Stein[1], Stefan Kühne[2], Jens Drawehn[3], Sven Feja[4], and Werner Rotzoll[5]

[1] IDS Scheer AG
Altenkesseler Str. 17, 66115 Saarbrücken, Germany
sebastian.stein@ids-scheer.com
http://www.ids-scheer.com/soa/
[2] Universität Leipzig, Institut für Informatik, Betriebliche Informationssysteme
Johannisgasse 23, 04103 Leipzig, Germany
kuehne@informatik.uni-leipzig.de
http://bis.informatik.uni-leipzig.de/
[3] Fraunhofer Institut für Arbeitswissenschaft und Organisation
Nobelstr. 12, 70569 Stuttgart, Germany
jens.drawehn@iao.fraunhofer.de
http://www.swm.iao.fraunhofer.de/
[4] Christian-Albrechts-Universität zu Kiel
24098 Kiel, Germany
sven.feja@email.uni-kiel.de
http://www.informatik.uni-kiel.de/
[5] DVZ Datenverarbeitungszentrum Mecklenburg-Vorpommern GmbH
Lübecker Str. 283, 19059 Schwerin, Germany
w.rotzoll@dvz-mv.de
http://www.dvz-mv.de/

Abstract. Today, most business processes are at least partially supported by IT systems. An integration of those IT systems is required, because a business process usually involves several IT systems. The OrViA framework suggests a model-driven approach to solve this integration problem. Platform independent business processes are modelled and transformed into executable ones. To ensure compliance to internal and external policies, the OrViA framework suggests using model checking technologies.

We present an industrial case study evaluating the OrViA framework in context of a model-driven SOA implementation in the E-Government domain. We were able to successfully apply the OrViA framework, but we also identified several problems. Our case study shows how model-driven approaches can be successfully applied in real-world projects.

1 Introduction

1.1 Research Background

Companies and organisations have been working on optimising their business processes in past years. This effort in the field of business process management

(BPM) [1] enables them to control and coordinate the activities within and across an organisation's boundaries, which is necessary to ensure cost-effectiveness and high quality.

Today, most business processes are supported by IT systems. Therefore, it is important to find an efficient way to implement the target business processes based on the existing IT systems and infrastructure. However, business process models focus on the business aspect and do not contain all information needed by an IT expert implementing an appropriate solution. Following the idea of Model Driven Architecture (MDA) [2], we need an approach starting with the business process model as a platform independent model (PIM) and derive a platform specific model (PSM) containing all information needed for implementation.

Many business processes span multiple IT systems, which must be integrated to deliver the desired result. In most organisations, there are many IT systems with different architectures, programming languages, and interfaces. A standard mechanism is necessary to realise the connections between the IT systems efficiently. We make use of the ideas of a service-oriented architecture (SOA) [3, see e. g.] to see business processes as a complex interaction of services provided by IT systems. Following this idea, process design is the description of a service interaction.

An iterative methodology supporting the realisation of business processes in the phases of analysis, design, and implementation is provided by the OrViA framework [4,5], which was developed by some of us. It supports the transformation of business process models into executable process models and makes use of model checking technologies to ensure the consistency of the model information on all levels. We[1] wanted to evaluate the applicability of the OrViA framework in a real-world scenario. Therefore, we conducted an industrial case study, which is presented in this paper.

1.2 Querying the Register of Residents

We conducted the industrial case study in the E-Government domain, whereas "the term *E-Government* focuses on the use of information and communication technologies (ICT) by governments applied to the full range of governmental and administrational functions" [6]. The use of ICT is expected to enable execution of business processes, integration of back-office systems among the public (and private) sector, and provisioning of fully customised and personalised electronic services to the different stakeholders. For instance, municipal processes include more than 1.000 interconnected and interdependent services and underlying processes for citizens, companies, and other administrational parties [7].

The real-world E-Government scenario used in the case study is provided by the German company Datenverarbeitungszentrum Mecklenburg-Vorpommern GmbH (DVZ M-V[2]) based in Schwerin, Germany. DVZ M-V maintains the

[1] This research was funded by the German federal ministry of education and research within the public research project OrViA – http://www.orvia.de/

[2] http://www.dvz-mv.de/

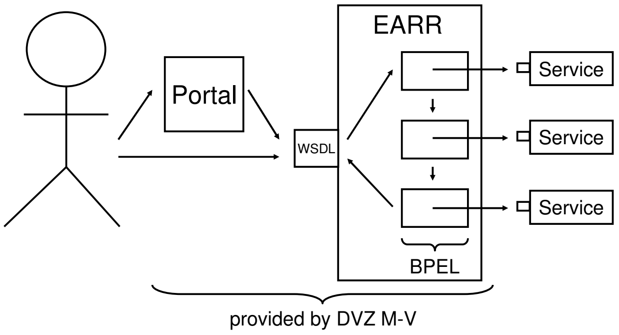


Fig. 1. Simple Electronic Access to Register of Residents (EARR) Service

IT systems of the public administration in the German region Mecklenburg-Vorpommern. They also support and implement new administrational processes.

Public authorities or commercial users utilise the governmental registration service for different purposes. The *simple electronic access to the register of residents* (EARR) service shown in Fig. 1 is used to check the validity of name and address data of a single person. The prerequisites and conditions under which the EARR takes place are defined by several legal regulations like the law *Landesmeldegesetz Mecklenburg-Vorpommern*. Different public authorities take part in this service as service providers and service consumers, using different IT systems. For example, there are 100 different registration applications by 6 different vendors used in the region Mecklenburg-Vorpommern. To ensure interoperability between these systems, standardisation is needed. In the field of governmental registration services, the public *XMeld* standard[3] defines the messages to be exchanged between the different systems. The implementation of the EARR must comply with those legal regulations and standards. Unfortunately, several versions of the XMeld standard are available and must be supported by the EARR service.

The process of validating an address and name of a single person against the register of residents starts with an incoming XMeld message describing the request to validate one resident's data. Next, it is checked if the request can be answered complying with the legal regulations like data protection. If this is the case, access is granted and the responsible IT system is determined, i. e. the register containing the requested data. For example, if the person is not living in the German region Mecklenburg-Vorpommern, the request is forwarded to the IT systems of the corresponding region. This is done through a so called intermediary service. There are several IT systems in Mecklenburg-Vorpommern are several IT systems, because the register of residents is organised peripherally. The request is forwarded to the designated system and the system answers with another XMeld message. This answer is passed back to the customer. Each request is documented.

[3] http://www.osci.de/xmeld132a/xmeld-132a.zip version 1.3.2a.

Following the idea of SOA, all described tasks are executed by a business process execution engine invoking several WSDL[4] based web services. As the tasks are orchestrated in the shape of a BPEL [8] process, the entire process can easily be reused as a service in more complex settings. For example, the EARR service can be used to validate a list of persons and addresses instead of creating a single request for each person address pair.

Before this case study was started, DVZ M-V already implemented the EARR service manually. The implementation used the BPEL execution engine Microsoft BizTalk Server and the BPEL orchestration was created with Microsoft Orchestration Designer. This previous experience allows us to compare the usage of the OrViA framework against a manual approach.

In this paper, the application of the OrViA framework in the industrial use-case EARR is described. We explain our research design in Sect. 2 and give a short overview of the OrViA framework in Sect. 3. Sect. 4 describes in detail how we applied the OrViA framework to the given use-case. An extensive discussion explaining advantages as well as disadvantages of the application of the OrViA framework in context of model-driven SOA implementations is presented in Sect. 5.

2 Research Design

Scientific rigor requires carefully designing and carrying out research [9]. While designing our research according to the top-down approach described by Creswell [9], we first locate our epistemological standpoint. In general, we have a post positivism standpoint, implying that we can generate knowledge through empirical observation and measurement. However, we extend our standpoint beyond standard post positivism by taking pragmatic knowledge claims into account. We are interested in a good solution for the given use-case, but we are aware that having found a solution for a specific use-case does not mean our work can be generalised.

Our epistemological standpoint allows us to freely select between different research methods [9] and to apply a mixed methods strategy of inquiry. This means, we can use quantitative as well as qualitative research methods based on their usefulness to support our research.

Before we can select any research method, we must formulate our research question and analyse which kind of data or insight we need. In general, we are interested investigating if the OrViA framework as described by Kühne et al. [4,5] can be successfully used as a guiding principle for model-driven SOA implementations. In order to prevent favouring the OrViA framework by relativising or ignoring criticism, we follow critical rationalism by Popper [10] and turn our research question around into the following hypothesis:

Hypothesis. It is impossible to successfully use the OrViA framework as a guiding principle for model-driven SOA implementations.

[4] See http://www.w3.org/TR/wsdl/

To make the hypothesis operational, we define the application to be "successful" if less budget is required, quality is improved, project length is shortened, or a combination of those three factors. Based on that hypothesis, we can further detail the research question and ask what is missing that it does not work, what shortcomings exist, where is commercial tool support missing, which parts of the OrViA framework are not integrated, and where is future research needed?

We are not interested in a theoretical discussion, but instead want to test the OrViA framework in a real-world scenario. However, we are not aware of practitioners using the OrViA framework as a guiding principle for model-driven SOA implementations. Therefore, it is impossible to use research methods like surveys or interviews to extract the experience gained by others. Instead, we first have to create this experience on our own. We are not interested in experiences for certain parts of the OrViA framework's application, but instead how all the different parts work together in a real-world setting. Therefore, we decided to conduct a case study, because it allows us to gather multiple experiences. We also investigated the possibility to conduct controlled experiments as described by Wohlin [11], but we found that the scope is too broad for an experiment to cover all aspects. Also, conducting several experiments each focusing on a small part does not give us the overall insight we are seeking. Our decision to do a case study is supported by all partners in our research project, because it allows them to gather experience, which they can use for their own business. Generating this individual know-how fosters the transfer of the research insights into industry, strengthening the competiveness of the involved companies, and allowing fast practical adoption of the developed technologies and methods.

3 Overview OrViA Framework

Transferring business processes to distributed execution environments involves different actors and roles with different skills such as business analysts, IT architects, integration specialists, and software developers. The co-operation between business-oriented and more technology-affine actors requires artefacts enabling the negotiation and coordination of requirements, design decisions, and implementation results. The OrViA framework [4,5] is based on process models on different levels of abstraction as basic communication mechanism. An outline is given in Fig. 2. It provides a conceptual framework whose building blocks may be represented by several appropriate methods and technologies.

Comparable to the four abstraction levels of Model Driven Architecture (MDA) [2], the OrViA framework proposes four levels. The business requirements level provides a computation independent view on the integration scenario. Below, the business process level comprises a platform independent design of the integration solution, i. e. its content is rather conceptual. The orchestration level covers the technical refinement of business process models. Finally, on the implementation level the platform-specific orchestration models are bound to resources such as E-Business services and are deployed on execution environments. The intended execution environments are domain-specific and service-oriented,

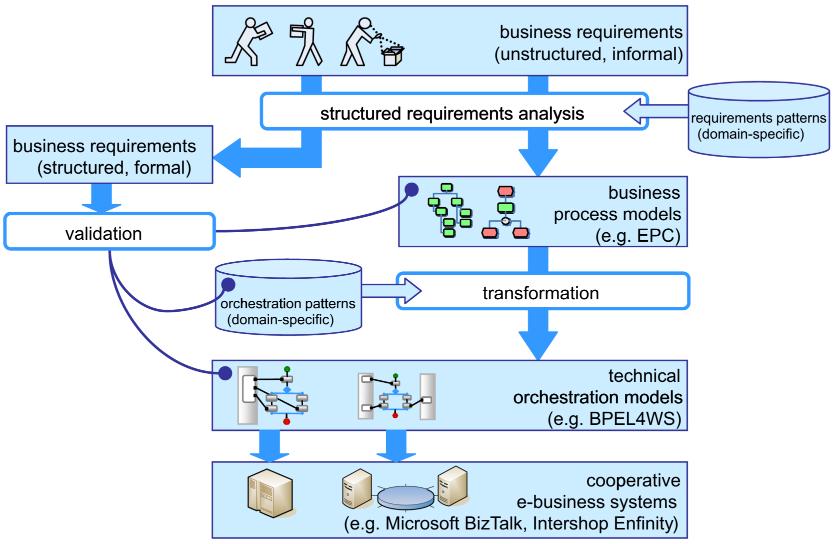


Fig. 2. The OrViA framework

i. e. services are first class entities. The services may be basic services provided by domain-specific E-Business components, e. g. E-Government components, as well as composed and orchestrated services relying on other services.

The modelling on each abstraction level is multi-perspective. This is due to the cognitive complexity of the focused inter-organisational E-Business integration scenarios. The process-oriented approach of the framework requires static views such as organisational, functional, data, and resource views, interconnected by a dynamic process view covering control-flow aspects. To enable added value through model operations the modelling concepts, such as model elements and model types, are formally defined in a well defined meta language.

In terms of a methodology, the OrViA framework proposes a top-down procedure that stepwise refines abstract models to more technology-oriented ones. Business-oriented process models tend to be incomplete representations of processes according to implementation relevant details. Exception handling for unexpected events, such as special cases from a business point of view or technical failures, is often omitted [12]. A pure migration transformation from a business-oriented process notation to a more technology-oriented notation is therefore not an adequate approach. Instead, the OrViA framework follows a pattern-based approach, i. e. combinations of business concepts are identified and replaced by appropriate technical exception-aware representations. Furthermore, the relationships between models of different abstraction levels are many-to-many relationships according to views, i. e. the refinements involve merging steps where different views are combined.

Another usage of formal models by automated model operations is validation. Besides the identification of structural and syntactical failures, the validation of process models on high and low levels of abstraction involves the checking of business requirements according to dynamic aspects. The process models describing conditional and parallel control flows are reduced to state-based representations that are input to model checkers. This reduction may suffer from state explosion, which makes a complete validation impossible. Nevertheless, a partial validation of process models against relevant business rules induced by customers, legal regulations or organisational directives reduces the efforts required in late testing phases.

The full potential of the OrViA framework of handling process-based integration scenarios is only gained if the building blocks work effectively together. The structured requirement analysis provides the functional input, which is consumed by validation and transformation. The validation checks models against functional requirements before and after transformation steps. Furthermore, the aspects of formal modelling and transformation improve the functional documentation of technical implementations, traceability of design decisions, and reproducibility of results.

4 Case Study

4.1 Overview

According to our research question described in Sect. 2, it is our aim to evaluate if and how the OrViA framework can be used successfully as a guiding principle for a model-driven SOA implementation. The case study implements the electronic access to the register of residents service as described in Sect. 1.2. This section describes the case study and is structured according to the three main building blocks of the OrViA framework, namely: structured requirements analysis, validation, and transformation (including deployment and execution).

4.2 Structured Requirements Analysis

The previous manual implementation of the EARR service was directly modelled in BPEL by DVZ M-V. This modelling resulted in a very complex model, because business details as well as technical details were mixed in one model. This model was platform specific (PSM) not allowing exchanging the underlying technology. The OrViA framework instead recommends to first do structured requirements analysis on a business-oriented platform independent level (PIM) and to derive the platform specific level through model transformation and stepwise refinement. This allows separating business and technical details.

To elicit the requirements, we interviewed the domain experts of DVZ M-V and studied the relevant laws and regulations. After, we created a first version of the process model and reworked it together with the domain experts of DVZ M-V in several workshops. We also interviewed the lead developer, who created the previous implementation of EARR and we analysed his BPEL model. The existing implementation gave us additional insights so that we could prevent

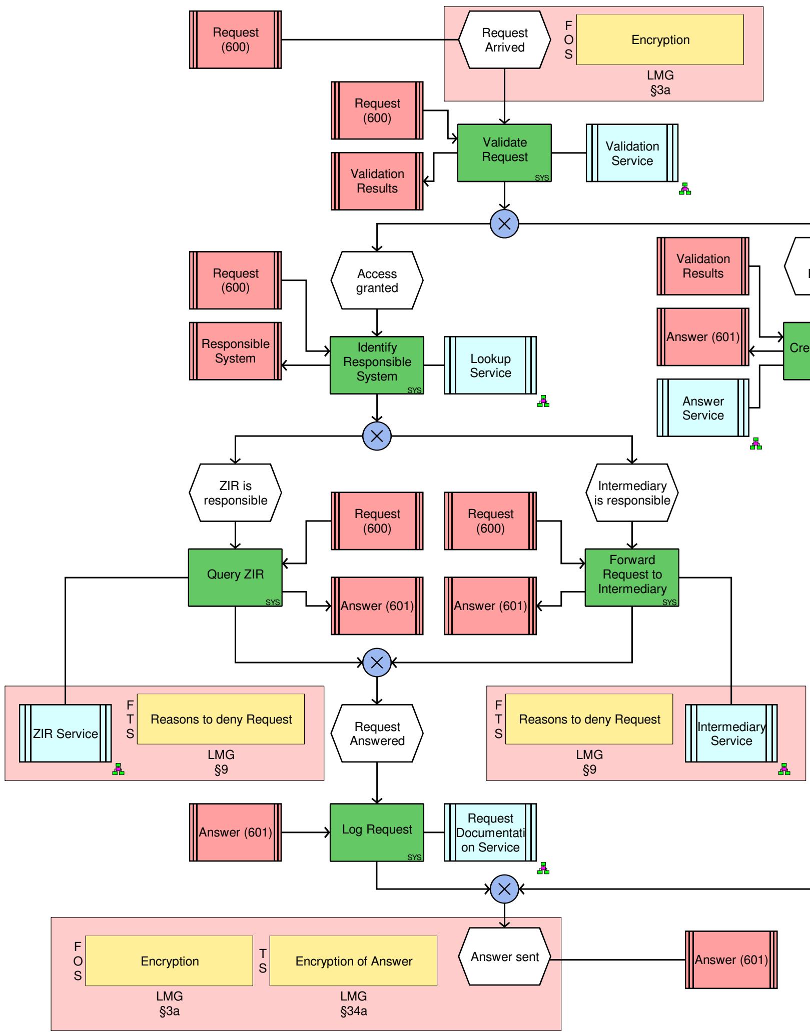


Fig. 3. Business Process Model of EARR in EPC Notation

doing the same mistakes again like adding handling for different XMeld versions directly to the process model.

To formally define the requirements, we used the off-the-shelf modelling software ARIS SOA Architect[5] by IDS Scheer AG. We followed the vendor's SOA methodology documented in [13]. The methodology formulates 10 steps starting with a platform independent process model and ending with a platform specific BPEL model. We used the EPC notation [14] to model the EARR process. The resulting business process model is shown in Fig. 3. The model consists of events

[5] http://www.aris.com/soa/

(white colour), business functions (green colour), software services (blue colour), input and output data (red colour), and legal annotations (yellow colour on a red layer). The process flow is defined by the directed connections between events, business functions, and operators (gray colour).

An important detail is the software services given in the EPC model. Each software service is still independent of any implementation technology and does not directly represent a WSDL web service. The modelling tool supports the business analyst in selecting a matching web service [15].

We used the standard EPC notation and extended it to cover domain-specific details, namely the links to the corresponding laws and regulations. This allows navigating from the process model to the laws and regulations, which explain in detail why and how a certain business activity must be performed. The resulting process model is detailed enough to be transformed later to a platform specific process model. It also documents all business (legal) requirements and it contains enough details to be validated against business policies.

4.3 Validation

The main task of validation is to check if the modelled business process fulfils the requirements stated in the analysis. The validation is done with model checkers technologies. They check if a model satisfies a given temporal statement. The software tool validating a model is called model checker. An overview of some of these technologies can be found in [16]. When we tried to apply model checking for the use-case EPC business process we faced the problem that current model checking technologies are hardly useable by business analysts. One reason is the complicated logical rules, which have to be defined textual in a temporal logic like the Computation Tree Logic (CTL) [17]. Another reason is the abstract type of models needed by model checkers. For instance, the model checker we used (called SMV [18]) needs a finite state machine as a Kripke Structure [19] as input.

To solve the first problem, we developed a graphical notation for CTL formulas. This notation extends the elements of EPCs with temporal operators and is called Graphical Computation Tree Logic (G-CTL) [20]. To use G-CTL for ARIS EPCs, we developed an extension for the ARIS Platform. This extension allows modelling the requirements in conjunction with the modelled EPC. An example G-CTL rule for the EARR process is shown in Fig. 4. This temporal rule expressed in CTL is `AG(E_Access_granted -> AF(F_Log_Request))` meaning that on every (AG) process path beginning from the event *Access granted* the function *Log Request* has to exist on all paths in the future (AF). In combination with the business process, this rule ensures the business requirement "Every *Access granted* has to be logged" is satisfied. The result of the validation should be *TRUE* if all temporal rules are satisfied. Otherwise the model checker gives one counter example where a rule is not fulfilled.

To validate the use-case EPC business process, the models (EPC and G-CTL) have to be translated in a format supported by the model checker. Both ARIS models are exported as *ARIS Markup Language* (AML) file format. The

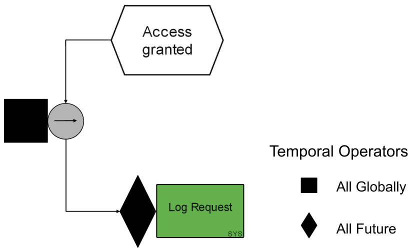


Fig. 4. An example graphical validation rule for the EARR process

translation has to handle each element of the EPC and G-CTL AML to receive the Kripke Structure and the CTL formula. We performed this transformation using the operator hierarchy concept [21]. We were able to successfully validate the business process models created in the case study. However, we did not validate any other artefacts produced like the BPEL model manually refined by the user after transformation. This will be the task of future research activities.

4.4 Transformation and Execution

After the successful validation of the business process model, we generated the executable BPEL model using the built-in model transformation of ARIS SOA Architect [22, see]. This model transformation is based on identifying workflow patterns [23] in the EPC and replacing them with corresponding BPEL elements. A static validation is performed before the transformation to ensure the EPC model can be transformed and to instruct the user how to model the EPC correctly. The transformation ignores events and considers them as a business documentation with no execution counterpart. Business objects given in the EPC model are transformed to BPEL variables if the business objects are mapped to XML Schema definitions. The transformation also provides merge functionality so that manual changes done in the generated BPEL model can be preserved in many cases.

It was not possible to directly deploy the generated BPEL model, but instead we had to fix e. g. the variable definitions and add additional namespaces to the BPEL header. Still, no significant reworks were needed. The BPEL process was deployed on the Oracle SOA Suite[6]. We were not able to deploy the generated model on Microsoft BizTalk, because BizTalk is not standard conformant. The web services were not hosted on the Oracle server, because in reality they are not hosted on the same machine either. The web services were deployed on the Java servlet container Apache Tomcat[7]. We were not allowed to directly use the web services provided by DVZ M-V, because of data protection reasons. We therefore implemented the web services as well as the end-user portal[8] on

[6] http://www.oracle.com/technologies/soa/soa-suite.html
[7] http://tomcat.apache.org/
[8] See https://service.mv-regierung.de/web/emrauser/emra

our own following the implementation done by DVZ M-V. The end-user only interacts with the portal.

5 Discussion

In the previous section, we have outlined how we instantiated the OrViA framework to do a model-driven SOA implementation for the given E-Government use-case. We conducted this case study to check if the hypothesis formulated in Sect. 2 holds. The hypothesis postulates that the OrViA framework cannot be used as a guiding principle to do a model-driven SOA implementation. After conducting the case-study, we slightly tend to reject the hypothesis under the conditions discussed in this section.

The case study was done with a use-case from the E-Government domain. Therefore, we can only reject the hypothesis for the E-Government domain if the selected use-case is representative for this domain. We are confident that the use-case is representative, because it was selected by a well-established use-case partner working in this domain. The use-case partner tried in the past to implement the same use-case, but in contrast to our case study the use-case partner tried a manual approach without using structured requirements analysis. The use-case is directly implementing a public law and it involves different actors like the register of residents, gateways to other German regions, and an end-user portal. As the use-case implements the business process as it is defined by the law, the use-case is not too simplistic but instead a realistic one. Even though we have just done one case study in the E-Government domain, we covered the most important technologies, because the used technologies were selected by the use-case partner and not by us.

As we have done the case study only for a use-case in the E-Government domain, we have to do additional use-cases in other domains to see if the OrViA framework can be applied there as well. We have already completed a similar study in the automotive industry, but we have not analysed the results yet. We have prepared other case-studies in the E-Commerce domain as well as in industrial engineering. We will also do an additional case study in the E-Government domain, but with the main difference that the implementation is not an executable process model but ordinary software instead. As those studies are not finished yet, their results are not reported and discussed in this paper.

One important building block of the OrViA framework is structured requirements analysis. In order to be able to reject the hypothesis, we must show the usefulness and applicability in case of the use-case. We used the EPC modelling notation as a base and modelled the process defined by the law as a business process. To better represent the requirements, the standard EPC notation was extended to cover domain-specific elements like clauses from the law. This approach of combining the advantages of a standard notation with a domain-specific language proved to be very successful. Using a standard notation has the big advantage that it is supported by commercial tools, which also provide the necessary transformations to follow a model-driven SOA implementation approach. On the other hand, extending such a standard notation by domain-specific elements allows to better capture the domain and to adapt the used language to the language used by the domain experts. This increases the comprehensibility of the models for the domain experts and results in a higher acceptance by them. It was possible to model all aspects important for the SOA implementation and to later derive the implementation through automated model transformation.

We were able to identify additional benefits of doing structured requirements analysis. For example, as the SOA implementation is directly derived from the business process model, the use-case partner can use the business process model for proving the compliance of the implementation to the law. This is possible, because structured requirements specification and implementation are not mixed in a single BPEL model, but instead the implementation is derived out of the structured requirements specification through an automated transformation. Therefore, it is enough to check the structured requirements specification during an audit in contrast to also checking the actual implementation.

Another important element of the OrViA framework is the transformation used to derive the IT implementation out of the structured requirements model. We used the transformation available in ARIS SOA Architect for this purpose. We also tend to reject the hypothesis in case of the transformation, but there are a few more concerns to be discussed. We were able to use the transformation to create the BPEL model. The structure of the business process model as well as the selected software services were correctly transformed into the corresponding BPEL constructs. We can confirm that this helps to speed up the implementation step, because creating all those constructs manually requires much more effort and is an error-prone task. On the other hand, the current transformation has some shortcomings, which result from bugs in the transformation as well as conceptual problems. For example, transforming business object descriptions given in the business process model into data definitions (given as XML schema definitions) is not working as expected. However, this seems to be a bug in the transformation used and not related to any fundamental conceptual problems in the approach. We were able to write small scripts fixing the wrongly created constructs and we expect those bugs to be fixed in a future release of the transformation software.

A more pressing issue is related to the transformation of conditions in the control flow of the business process model. As BPEL is an executable language, the conditions for loops and branches must be defined with a strict syntax like XPath expressions. However, a business process model usually does not contain such formal expressions nor can we expect that a business analyst is able or willing to create such expressions. This would be wrong from a conceptual view point, as well. XPath expressions are a concrete technology and should therefore not be added to a platform independent model like the business process model. We therefore must investigate, how such conditions can be expressed in a technology independent way. A possible solution might be adding business rules to the EPC, but this needs further investigations. Besides those problems, we were able to deploy and execute the generated BPEL models without having to change a lot.

The third core element of the OrViA framework is validation. According to the OrViA framework, validation should be applied to different artefacts like the business process model, the transformation rules, and the generated executable process model. We have not validated the transformation rules, because they are packaged in the transformation software and are therefore not accessible to us. We do not consider this to be a shortcoming of our case study, because we have to rely on the software used as a real-world project would have to do. As we have discussed in the previous section, we have validated the business process model. Based on the law, we created a set of rules, which must be enforced like that each access to the register of residents must be logged. Afterwards, we checked the created business process model to see if it complies with the rules using model checking technologies. From an algorithmic and technological standpoint we can confirm that validation works. However, a more interesting question is whether the approach is feasible in real-world projects. Our first concern was to check if business analysts are able to formulate the rules. We did a modelling workshop together with another use-case partner, explained the graphical rule modelling notation, and did some exercises. We learnt from this workshop that someone able to formulate a correct business process model is able to formulate the rules using our graphical rule notation.

Our second concern was about how the rules can be integrated with the process models. For example, it must be possible to reference a process model element like an event or a business function in the rules. The current solution is not satisfying, because the dependency between rule and process model is too tight. If a rule specifies that a business function must occur after a certain event, the model checker will be only able to validate this rule if the business function and the event in the process model are named exactly as in the rule. If business analysts are allowed to freely name model elements while creating a business process model, this is very unlikely to happen. Therefore, the vocabulary allowed for the model elements must be defined and naming conventions must be enforced. At the current point, we see this as a major problem and we will investigate how we can relax this constrain in the future.

The OrViA framework suggests validating the generated executable process model in order to ensure that manual changes have not changed the semantics and that the executable model is still implementing all business requirements. We have not done this validation step, because the approach would have been similar to validating the business process model and the same limitations would apply. In summary, we can reject the hypothesis in case of validation, even though we found this to be the most problematic part.

We do not consider it to be a threat to the validity of our study that we only used one specific set of tools (mainly ARIS SOA Architect, Oracle BPEL Process Server, and Apache Tomcat), because our focus is on evaluating if the OrViA framework can be applied for such an implementation and not if it works with any kind of tool combination. However, it will be interesting to see if such a tool chain can also be built using Open Source software. We will investigate that in a future case study.

In general, we found that integrating the different tools to form a complete tool chain is still challenging, even though there are public standards like BPEL and WSDL. Making the top-down approach work is possible, but implementing a roundtrip scenario is almost impossible. For example, if the BPEL model is changed in ARIS SOA Architect as well as in Oracle JDeveloper, it is hard to merge those changes. The OrViA framework only provides a top-down path with no backward links, because this makes the OrViA framework simple and easy to understand. On the other hand, it might be a too simplistic view for real-world projects, which is another point why we only slightly reject the hypothesis.

Besides the problems and limitations discussed above, there are also some clear advantages of applying the OrViA framework. The OrViA framework clearly divides the necessary tasks into packages. Each package requires specific skills like having profound business knowledge for structured requirements analysis or having software engineering skills for deployment and execution of the generated executable process model. This clear separation helps to reduce the overall complexity, because people only need a part of the overall required skill set to handle the part they are assigned to. The complexity is further reduced by step-wise refinement. Each step only adds few aspects to the models and is therefore easier to handle. For example, during business process modelling, software services are discovered and selected but providing the correct binding information is done at a later step.

The OrViA framework supports in providing different perspectives on the overall solution, which is another advantage and success factor for real-world projects. Another advantage lies in the fact that the OrViA framework is agnostic of the software engineering methodology used. It does not matter if the project is done following the Waterfall model or using an agile approach. This is an important fact, because companies usually have their own methodologies, which often cannot and should not be replaced. Therefore, being independent of concrete methodologies supports the adoption of the OrViA framework. On the other hand, the OrViA framework is conceptual and therefore it cannot be used out of the box. Companies wishing to use the OrViA framework have to conduct a pilot project to see how the framework must be tailored for their needs.

In summary, we can reject the hypothesis that the OrViA framework cannot be used successfully as a guiding principle for model-driven SOA implementations.

6 Summary

In this paper, we presented a case study done in the E-Government domain to evaluate the applicability of the OrViA framework as a guiding principle for model-driven SOA implementations. Besides introducing the real-world use-case, we have discussed in detail our research design. Following the idea of rationalism criticism, we formulated a negative hypothesis and tried to prove that the OrViA framework is not applicable.

Our research results show we have to slightly reject this hypothesis if the conditions discussed in the paper are taken into account. For example, we can only say that the OrViA framework is applicable in the E-Government domain. Currently, there is a lack of integration between validation and structured requirements analysis and the transformation of condition expressions is not solved. Based on those findings, we formulate several points for further investigations. Still, we are confident that the OrViA framework is very useful to bring model transformation technologies to the business.

References

1. Smith, H., Fingar, P.: Business Process Management: The Third Wave, 1st edn. Meghan-Kiffer Press, Tampa (2003)
2. Miller, J., Mukerji, J.: MDA guide. Technical Report omg/2003-06-01, Object Management Group (OMG) Version 1.0.1 (June 2003)
3. McGovern, J., Sims, O., Jain, A., Little, M.: Enterprise Service Oriented Architectures. Springer, Dordrecht (2006)
4. Kühne, S., Thränert, M., Speck, A.: Towards a methodology for orchestration and validation of cooperative e-business components. In: Rutherford, M.J. (ed.) 7th GPCE Young Researcher Workshop, pp. 29–34 (2005)
5. Fähnrich, K.P., Kühne, S., Speck, A., Wagner, J.(eds.): Integration betrieblicher Informationssysteme: Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering. Leipziger Beiträge zur Informatik, vol. IV. Eigenverlag Leipziger Informatik-Verbund (LIV), Leipzig, Germany (2006)
6. Lau, E.: E-government: Analysis framework and methodology. Puma(2001)16/ann/rev1, OECD (2001), `http://www.olis.oecd.org/olis/2001doc.nsf/LinkTo/NT00000936/$FILE/JT00118445.PDF`
7. Algermissen, L., Delfmann, P., Niehaves, B.: Experiences in process-oriented reorganisation through reference modelling in public administrations - the case study regio@komm. In: 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy (ECIS) (2005)
8. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. Technical report, OASIS (May 2003), `http://www-128.ibm.com/developerworks/library/ws-bpel/`
9. Creswell, J.W.: Research design: Qualitative, quantitative, and mixed method approaches, 2nd edn. Sage Publications, Inc, Thousand Oaks (2002)
10. Popper, K.: Logik der Forschung, 11th edn. Mohr Siebeck, Tübingen (1934)
11. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wessén, A.: Experimentation in software engineering: an introduction. International Series in Software Engineering. Kluwer Academic Publishers, Norwell (2000)
12. Dehnert, J., van der Aalst, W.M.P.: Bridging Gap between Business Models and Workflow Specifications. International Journal of Cooperative Information Systems 13(3), 289–332 (2004)

13. Stein, S., Ivanov, K.: Vorgehensmodell zur Entwicklung von Geschäftsservices. In: Fähnrich, K.P., Thränert, M. (eds.) Integration Engineering – Motivation, Begriffe, Methoden und Anwendungsfälle. Leipziger Beiträge zur Informatik VI. Eigenverlag Leipziger Informatik-Verbund (LIV), Leipzig, Germany (2007)
14. Scheer, A.W., Thomas, O., Adam, O.: Process Modelling Using Event-Driven Process Chains. In: Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M. (eds.) Process-Aware Information Systems, pp. 119–146. Wiley, Hoboken (2005)
15. Stein, S., Barchewitz, K., El Kharbili, M.: Enabling Business Experts to Discover Web Services for Business Process Automation. In: Pautasso, C., Gschwind, T. (eds.) 2nd Workshop on Emerging Web Services Technology, Halle, Germany, pp. 19–35 (November 2007)
16. Pfeiffer, J.H., Rossak, W.R., Speck, A.: Applying model checking to workflow verification. In: ECBS 2004: Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2004), Washington, DC, USA, pp. 144–151. IEEE Computer Society, Los Alamitos (2004)
17. Clarke, E.M., Draghicescu, I.A.: Expressibility results for linear-time and branching-time logics. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 428–437. Springer, Heidelberg (1989)
18. McMillan, K.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
19. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking, 3rd edn. The MIT Press, Cambridge (2001)
20. Feja, S., Fötsch, D., Stein, S.: Grafische Validierungsregeln am Beispiel von EPKs. In: Software Engineering 2008, Fachtagung des GI-Fachbereichs Softwaretechnik, München,GI, February 22. LNI (2008) (to appear)
21. Fötsch, D., Speck, A., Hänsgen, P.: The Operator Hierarchy Concept for XML Document Transformation Technologies. In: 3. Berliner XML-Tage 2005 (BXML 2005), Berlin, Germany, pp. 59–70 (2005)
22. Stein, S., Ivanov, K.: EPK nach BPEL Transformation als Vor aussetzung für praktische Um setzung einer SOA. In: Bleek, W.G., Raasch, J., Züllighoven, H. (eds.) Software Engineering 2007. Gesellschaft für Informatik (GI), Hamburg, Germany, March 2007. Lecture Notes in Informatics (LNI), vol. 105, pp. 75–80 (2007)
23. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)

Efficient Compliance Checking Using BPMN-Q and Temporal Logic

Ahmed Awad, Gero Decker, and Mathias Weske

Business Process Technology Group
Hasso-Plattner-Institute, University of Potsdam, Germany
{ahmed.awad,gero.decker,weske}@hpi.uni-potsdam.de

Abstract. Compliance rules describe regulations, policies and quality constraints business processes must adhere to. Given the large number of rules and their frequency of change, manual compliance checking can become a time-consuming task. Automated compliance checking of process activities and their ordering is an alternative whenever business processes and compliance rules are described in a formal way. This paper introduces an approach for automated compliance checking. Compliance rules are translated into temporal logic formulae that serve as input to model checkers which in turn verify whether a process model satisfies the requested compliance rule. To address the problem of state-space explosion we employ a set of reduction rules. The approach is prototypically realized and evaluated.

1 Introduction

Business processes and their explicit representation in business process models are important assets to understand how companies work. To be in line with their business goals, but also with legal regulations, companies need to make sure that their operations satisfy a set of policies and rules, i.e., they need to design compliance rules and implement compliance checking mechanisms.

Compliance rules originate from different sources and keep changing over time. Also, these rules address different aspects of business processes, for example a certain order of execution between activities is required. Other rules force the presence of activities, e.g. reporting financial transactions to an external entity. The obligation of adhering to rules ranges from gaining competitive advantage to employing strategies that protect businesses from failure, e.g. Basel II (http://www.Basel-II.info) in the field of risk assessment in the banking sector; other regulations come as quality standards like ISO 9000. Regulations might also come with legal regulations like the Sarbanes-Oxley Act of 2002 [1]. Violation could lead to penalties, scandals and loss of business reputation.

The changing nature of rules (e.g. due to changes in policies) calls for checking business processes each time a rule is added or changed. As a result, organizations need to hire compliance experts auditing their process models. In the case of manual auditing, a considerable amount of time is consumed in identifying the

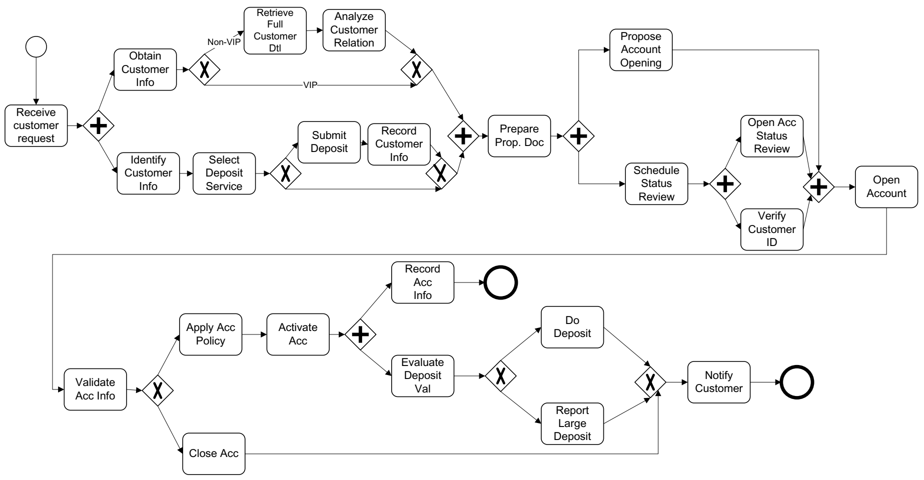


Fig. 1. Banking business process model adapted from [18]

set of process models affected by each rule before revising them for compliance; something that may lead to failure to meet the deadline for declaring compliance.

The contribution of this paper is twofold. First, it presents an automated approach for checking compliance of business process models regarding ordering constraints for activities. Automation is achieved by expressing rules as queries in a visual language we developed earlier [2]. These queries determine the set of process models that are candidates for further compliance checking. We assume that candidate process models contain all activities mentioned in the rule. If this is not the case i.e. the process model contains only a subset of the activities in the rule, we simply know that the process model is no compliant. The hard part is to be sure that candidate process models do satisfy the rule. This activity is already very time consuming in manual auditing. Efficiently determining whether process models comply to the rules using model checking is the second contribution. Our approach is not limited to processes defined in BPMN, it can be applied to any graph based process definition language as described below.

The remainder of this paper is structured as follows. In Section 2 we discuss a scenario in the banking business and derive a set of rules. Section 3 discusses how we adapted BPMN-Q to express compliance rules as queries. Details of applying model checking to formally verify compliance is given in Section 4. Related work is reported in Section 5, before we conclude the paper in Section 6.

2 Compliance Example

In this section, we introduce a business process from the banking sector. It will serve as example throughout the paper. In the financial sector, banks are obliged to conform to several compliance rules. Consider the process model in Figure 1 (expressed in BPMN notation) for opening a bank account.

The process starts with ”Receive customer request” to open an account. Customer information is obtained from the request and identified. In case of a non-VIP customer, her detailed information is retrieved and its relation to the bank is analyzed. If the customer selects to open a deposit account, an extra deposit form must be submitted and the customer's information is recorded. With all previous steps completed, a proposal document is prepared by a bank's employee for further analysis. A proposal's status review is scheduled along with the proposal of opening an account. Later on, the customer's identity is verified and the status of the account is reviewed. An account is opened at that point followed by a validation of account information. If validation fails the account is closed. With valid account information, an account policy is applied before the account is activated. Finally the account information is recorded and the account is ready for transactions. Large deposit transactions (”Evaluate Deposit Val”) are reported to a central bank to track the possibility of money laundering. At the end of the process the customer is notified.

The process has to comply with several rules. Among them are rules to prevent money laundering. The rules demand that an account is not opened until certain checks have been completed. We selected the following two rules to guide the discussion in this paper.

– Rule 1: Before opening an account, customer information must be obtained and verified. The rule delays the activity ”open account” until information about the customer has been verified e.g. checking absence from black lists containing undesirable customers. Violation of this rule may lead to opening an account for individuals that are known to be on the black list of e.g. the central bank.
– Rule 2: Whenever a customer requests to open a deposit account, customer information must be recorded before opening the account. Information must be recorded to ease future tracking of their transactions and identifying them in case they run large deposit transactions.

3 Declarative Representation of Compliance Rules

As mentioned in Section 1, the first contribution of this paper is the automated discovery of process models that are relevant to a given rule (see Definition 3). To determine these relevant process models, we express rules concerned with ordering of activities as queries in BPMN-Q. Before we go further with the details of using BPMN-Q to express compliance rules, we briefly introduce BPMN-Q. Next, we discuss how to adapt it for compliance checking.

BPMN-Q [2] is a visual language based on BPMN. It is used to query business process models by matching a process graph to a query graph. Figure 2 sketches an overview of the steps of processing a query. In addition to the sequence flow edges in BPMN, BPMN-Q introduces the concept of path (see edge in Figure 3 (b)). When matching a process graph (like the one in Figure 3 (a)) to the query in (b), the result of the path edge is the sub-graph of the matching process in which the two nodes along with nodes in between are contained (Figure 3 (c)).

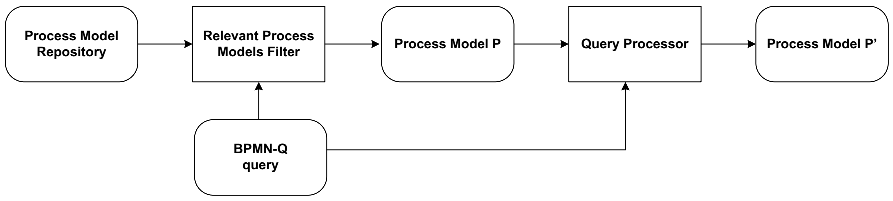


Fig. 2. Overview of query processing in BPMN-Q

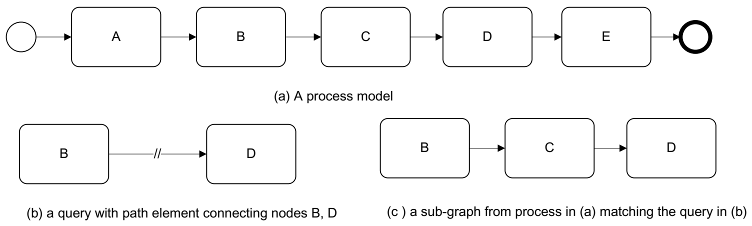

A
B
C
D
E
(a) A process model
B
//
D
(b) a query with path element connecting nodes B, D
B
C
D
(c) a sub-graph from process in (a) matching the query in (b)

Fig. 3. Example of a BPMN-Q query

We now define process graphs, where the set of nodes $\mathcal{N}$ can be either activities, events or gateways. Set $\mathcal{F}$ represents the sequence flow edges that can connect nodes. The definition is adapted from [4].

Definition 1. *A process graph is a tuple PG= ($\mathcal{N}$, $\mathcal{A}$, $\mathcal{E}$, $\mathcal{G}$, $\mathcal{F}$) where*

- *$\mathcal{N}$ is a finite set of nodes that is partitioned into the set of activities $\mathcal{A}$, the set of events $\mathcal{E}$, and the set of gateways $\mathcal{G}$*
- *The set of events $\mathcal{E}$ can be further partitioned into:*
  - *Start events $\mathcal{E}^s$ i.e. nodes with no incoming edges.*
  - *Intermediate events $\mathcal{E}^i$.*
  - *End events $\mathcal{E}^e$ i.e. nodes with no outgoing edges.*
- *$\mathcal{F} \subseteq (\mathcal{N} \setminus \mathcal{E}^e) \times (\mathcal{N} \setminus \mathcal{E}^s)$ is the sequence flow relation between nodes.*

BPMN-Q provides more types of edges to connect nodes. It is possible to express in the query what we call paths as discussed earlier.

Definition 2. *A query graph is a tuple QG= ($\mathcal{NQ}$, $\mathcal{AQ}$, $\mathcal{EQ}$,$\mathcal{GQ}$, $\mathcal{S}$, $\mathcal{PA}$) where*

- *$\mathcal{NQ}$ is a finite set of nodes that is partitioned into the set of activities $\mathcal{AQ}$, the set of events $\mathcal{EQ}$, and the set of gateways $\mathcal{GQ}$*
- *$\mathcal{S} \subseteq \mathcal{NQ} \times \mathcal{NQ}$ is the set of sequence flow edges between nodes.*
- *$\mathcal{PA} \subseteq \mathcal{NQ} \times \mathcal{NQ}$ is the set of path edges between nodes.*

A process graph is relevant to a query graph only if the set of activity nodes in a process graph is a superset of the activity nodes in a query graph.

Definition 3. *A process graph PG= ($\mathcal{N}$, $\mathcal{A}$, $\mathcal{E}$, $\mathcal{G}$, $\mathcal{F}$) is relevant to query graph QG= ($\mathcal{NQ}$, $\mathcal{AQ}$, $\mathcal{EQ}$,$\mathcal{GQ}$, $\mathcal{S}$, $\mathcal{PA}$) iff $\mathcal{A} \supseteq \mathcal{AQ}$*

To address the execution ordering between activities , a process graph is divided into a set of execution paths. An execution path is a sequence of nodes starting from one of the process start node(s) and ending at one of its end node(s).

Definition 4. *An execution path exp is a sequence of nodes* $(n_0, \ldots, n_k)$ *where* $n_0, \ldots, n_k \in \mathcal{N}$, $n_0 \in \mathcal{E}^s$ *and* $n_k \in \mathcal{E}^e$. EXP *is the set of all execution paths in a given process graph.*

We can determine the execution order between two nodes a, b in a process graph with respect to an execution path *exp* by finding the precedence between the *first occurrence* of node a and occurrence of node b. We emphasize on the *first occurrence* of node a because a might appear more than once in case the execution path includes loops. We need to be sure that a executed at least once before the execution of b.

Definition 5. *An execution ordering relation between nodes on an execution path exp is defined as* $<_{exp} = \{(n', n'') \in \mathcal{N} \times \mathcal{N} : n' \in exp \wedge n'' \in exp \wedge \exists i, j (n' = n_i \wedge n'' = n_j \wedge i < j \wedge \nexists j'(n'' = n_{j'} \wedge j' < i))\}$, *where* $n \in exp$ *means that the node n resides on the execution path exp.*

The evaluation of path edge as shown in Figure 3 conforms to the following definition.

Definition 6. *A function subgraph (a, b, p_i):= PSG'(N',E'), where p_i is a process graph and a, b $\in \mathcal{N}_i$, constructs the process sub-graph of p_i where:*

- $N' = \{x : \forall\ exp_i \in EXP_i\ (\ a \in exp_i \wedge b \in exp_i \wedge x \in exp_i \wedge (\ x = a \vee x = b \vee (\ a <_{exp_i} x \wedge x <_{exp_i} b)\ ))\ \}$
- $\forall\ x,y \in N'\ (\ (x,y) \in \mathcal{F} \rightarrow (x,y) \in E')$

A relevant process graph to a query graph is said to match it if it satisfies all sequence flow and path edges as in Definition 7.

Definition 7. *A process graph PG= ($\mathcal{N}$, $\mathcal{A}$, $\mathcal{E}$, $\mathcal{G}$, $\mathcal{F}$) matches a query graph QG= ($\mathcal{NQ}$, $\mathcal{AQ}$, $\mathcal{EQ}$,$\mathcal{GQ}$, $\mathcal{S}$, $\mathcal{PA}$) iff:*

- $\mathcal{NQ} \subseteq \mathcal{N}$.
- $\mathcal{S} \subseteq \mathcal{F}$.
- $\forall\ (n, m) \in \mathcal{PA}\ (subgraph(n,m,p) \neq \emptyset)$.

To express rules as queries, we add an activity node in the query for each activity mentioned in the rule. To express the ordering relationship between two activities, a path element connects a source node in the query to a destination node. Figure 4 shows how the rules from Section 2 can be visually represented.

If BPMN-Q does not find a match i.e. it fails to find an execution path from *Record Customer Info* to *Open Account* then we are sure that the answer to rule 2 is "NO". On the other hand if the matching succeeds and a path is found, we cannot be sure that this execution path is activated in all possible execution scenarios. That is because the path element does not consider the semantics of

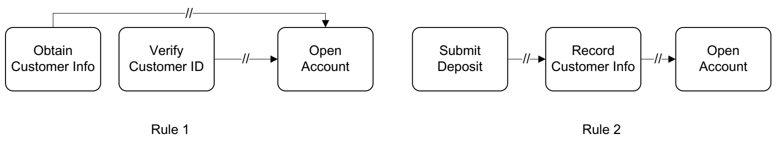


Fig. 4. Queries capturing rules

control nodes in execution paths. We record this as a limitation that we will resolve in this paper.

Another limitation is the ability to express the direction of the execution dependency between activities. For instance, determining the order of execution between two activities A and B in Figure 5 by finding path from A to B, are we interested in being sure that every execution of activity A will *lead to* execution of Activity B, or on the other hand each time activity B is executed it must have been *preceded* by an execution of activity A. The two situations are different. In the first one we state a constraint over the future states of a process execution while in the second one we state this constraint over its past execution states. From this simple process fragment, we can see that activity B is preceded by activity A but activity A does not lead to activity B.

The two concepts of *precedes* and *leads to* are not distinguishable in queries. To overcome these limitations we decided to:

- Extend BPMN-Q with *precedes* and *leads to* qualifiers to solve the second limitation.
- Use Model checking as a formal approach to verify constraints against process sub-graphs to solve the first limitation.

To visually differentiate between the *precedes* and *leads to* semantics we simply added them under the arc representing the path operator like ≪*precedes*≫ and ≪*leads to*≫ respectively in a way similar to the stereotypical extension in UML. Now queries from Figure 4 will look as in Figure 6. The formalism behind these two concepts is given within the context of this section along with necessary supporting definitions. To say that node A in a process graph leads to (see Definition 8) node B, we just need to be sure that every execution path going through A also goes through B. On the other hand node A precedes (Definition 9) node B only if all execution paths gone through B have gone through node A before.

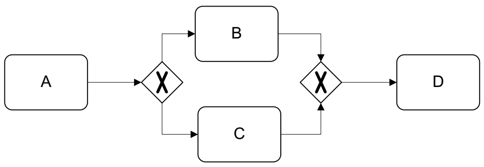


Fig. 5. A fragment of a process model to show difference between leads to and precedes concepts. A *precedes* B, but A does not *lead to* B.

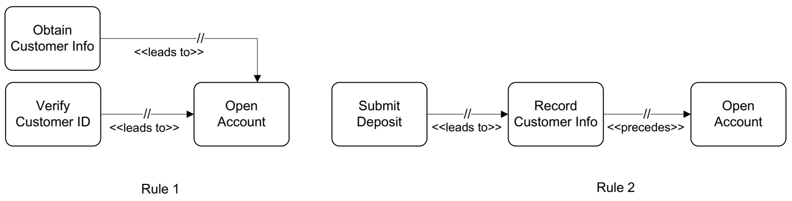


Fig. 6. Queries refined

Definition 8. *Node m leads to node n iff $\forall exp \in EXP(m \in exp \Rightarrow n \in exp \wedge m <_{exp} n)$*

Definition 9. *Node m precedes node n iff $\forall exp \in EXP(n \in exp \Rightarrow m \in exp \wedge m <_{exp} n)$*

We can read queries as follows:

– Rule 1: The execution of "Obtain Customer Info" and "verify customer ID" always leads to execution of "Open Account" i.e. *leads to (Obtain Customer Info, Open Account) and leads to (Verify Customer ID, Open Account).*
– Rule 2: The execution of submit deposit leads to recording customer info, which must precede opening an account i.e. *leads to (Submit Deposit, Record Customer Info) and precedes (Record Customer Info, Open Account).*

The effects of these extensions are more than just a visual differentiation between the *leads to* and *precedes* concepts. One further effect is the temporal expression generated from each rule, shown in the next section. Another effect is on the query itself. In case of *precedes* paths we add a start event to the query graph (in case the query does not already have one) and a path operator between the start event and the destination of the *precedes* operator. This path is added to allow the query to match all possible paths from the beginning of the process to the destination activity of the *precedes* operator in order to give the model checker the possibility to find violations (if any).

4 Efficient Analysis Using Temporal Logic

This section discusses how compliance checking on BPMN process models can be carried out. The compliance rules are formulated as BPMN-Q queries. Our approach can be divided into the following steps, which are also illustrated in Figure 7.

1. **Retrieval of BPMN sub-graphs.** A query processor takes a BPMN-Q query as input and retrieves a number of BPMN sub-graphs from a BPMN process model repository. Only those process models are considered that structurally match the BPMN-Q query.
2. **Graph reduction.** The sub-graphs are reduced, mainly removing activities and gateways that are not relevant to the query.

3. **Petri net generation.** The reduced sub-graphs are translated into Petri nets.
4. **State space generation.** The Petri nets are checked for boundedness and the reachability graph is calculated.
5. **Generation of temporal logic formulae.** The BPMN-Q query is translated into temporal logic formulae.
6. **Model checking.** The finite state machines and the temporal logic formulae are fed into a model checker. The model checker verifies whether the temporal logic formulae are respected by the given state machines. As a result, it is detected which process models comply to the initial BPMN-Q queries.

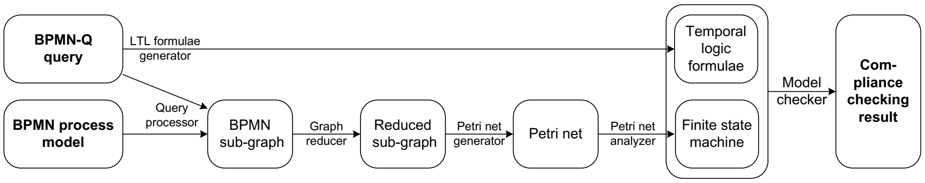


Fig. 7. Compliance checking approach

Retrieval of BPMN Sub-graphs

The major role of BPMN-Q is to select the set of process models which are relevant for the query. As a first step, BPMN-Q selects all process models in each of which the set of activity nodes is a superset for the activities in the query. With each of these processes, the ordering between activities, expressed in the query as sequence flows and/or paths, are tested. If the process graph fails to satisfy any of these, it is dropped from the answer set. For more details about query processing of BPMN-Q, please refer to [2].

The result of queries (rules) 1 and 2 in Figure 6 against the process model in Figure 1 is shown in Figure 8 where all nodes between "Obtain Customer Info" and "Open Account" are included in the result. Since the activity "Verify Customer ID" already resides on the path from "Obtain Customer Info" to "Open Account" , the evaluation of path from "Verify Customer ID" to "Open Account" will not introduce new nodes or edges to the result.

We can notice in the result of query 2 (as shown in Figure 9) that nodes preceding the activities "Submit Deposit" and "Record Customer Info" were

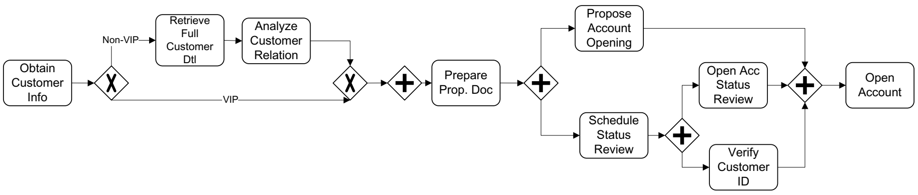


Fig. 8. Process graph matching rule 1

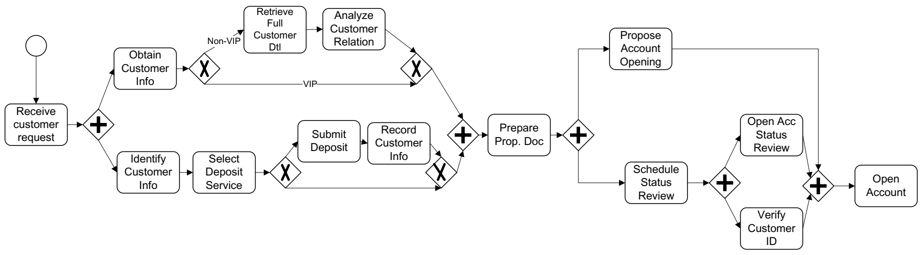


Fig. 9. Process graph matching rule 2

also included. This is due to the implicit inclusion of a start node with a path to "Open Account". This inclusion occurred because of the *precedes* between "Record Customer Info" and "Open Account" as discussed in Section 3.

Graph Reduction

Reduction rules have been successfully used either as a stand alone approach [23,24], or as an engineering approach used to reduce the complexity of the process models [27,19] to verify correctness of process models. We adopt the reduction approach to reduce the state space for model checking in a way to work around the state space explosion [3]. Unlike the aforementioned approaches which focused on simplifying the underlying control graph, our approach respects and depends on the set of activities included in the rule to be verified. So, the reduction result differs depending on the rule to be verified.

We have already discussed the difference between precedes and leads to dependencies. As a result, the reduction rules applied must respect these dependencies. This is especially important when considering decision points (XOR/OR-splits) and merging behavior (XOR/OR-joins).

- R1: Reduction of Activities, and intermediate events: all activities that are not of interest to the compliance rule (query) are removed.
- R2: Reduction of Structured Blocks. A block of split gateway that is directly connected to a join gateway of the same type and both have no other connections is replaced with a sequence flow edge.
- R3: Merging of similar gateways. if two splits or two joins of the same type follow each other, for example an XOR-split A is followed by XOR-split B and B has no other incoming edges, B is removed and the sequence flow edge between A and B. A inherits all outgoing sequence flow edges of B.
- R4: Reduction of structured loops. Loops having an XOR/OR-join node as an input and an XOR/OR-split as an output with backward edge from split node to join node are reduced by removing the backward edge.
- R5: Reduction of Single activity parallel blocks. Whenever an activity that is of interest to the compliance rule (query) lies in parallel block, after the reduction of other activities that are parallel to it, parallel block is removed and the activity is retained connected with nodes before and after the block. if the parallel block contains in only one of its branches activities of interest

that are somehow in a nested structure, the direct edges from the AND-Split to the AND-join are removed.

- R6: Reduction of Single output AND-Split and single input join nodes. Due to the nature of pattern matching based query processing of BPMN-Q (see [2] section 6), and to application of other reduction rules, a situation where an AND-Split with single outgoing edge or a join gateway that is either preceded or followed by an Activity of interest to the query may occur. The rule replaces the node with a single sequence flow between the source of its incoming sequence flow and the destination of its outgoing sequence flow.
- R7: Reduction of start events. Depending on whether the query contains start events (either explicitly by the user or implicitly added as described in 3, we reduce start events in case two or more start events are the input for an AND-join. We remove the set of start events along with sequence flow edges to the AND-join and the AND-join itself and introduce a single start event and a sequence flow edge from that event to the destination of the outgoing sequence flow edge of the AND-join.
- R8: Reduction of single activity selection block. The application of this rule depends on the type of path operator the activity is involved in. The reduction rule is applicable only If this activity is involved in only *leads to* paths as a source, otherwise we cannot apply the rule.
- R9: Reduction of BPMN-specific activities. This includes the MI activity, loop activity and ad-hoc activities. For MI and ad-hoc activities we assume that there is only one token produced from the activity after all running instances complete. In case of Loop activities we follow the mapping shown in [4].

Rules from R1 to R4 are adapted from previous work using reduction rules to verify correctness [27,19,23]. Unlike other reduction approaches, the reduced graph always contains the set of activities mentioned in the query graph.

Applying reduction rules to the process graph of Figure 8 will result in reduced graph shown in Figure 10. We elaborate more details on applying reduction rules to the result of query 2 shown in Figure 9. R1 is applied to remove all activities and intermediate events that are not of interest to the compliance rule. The result is shown in Figure 11 (a). Applying rules R2, R3 to the graph resulting from (a) yields the reduced graph in (b). Applying R2 again removes the parallel block immediately before the "Open Account" activity. A special case of R5 (as discussed earlier) is applied where the direct edges from the first AND-Split to the AND-join are removed resulting in graph (c). A final application of R6 produces graph in (d).

Generation of Temporal Logic Formulae

Linear Temporal Logic allows expressing formulae about the future of systems. In addition to logical connectors ($\neg, \vee, \wedge, \rightarrow, \Leftrightarrow$) and atomic propositions, it

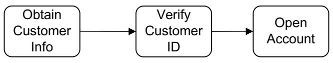


Fig. 10. Reduced graph for rule 1

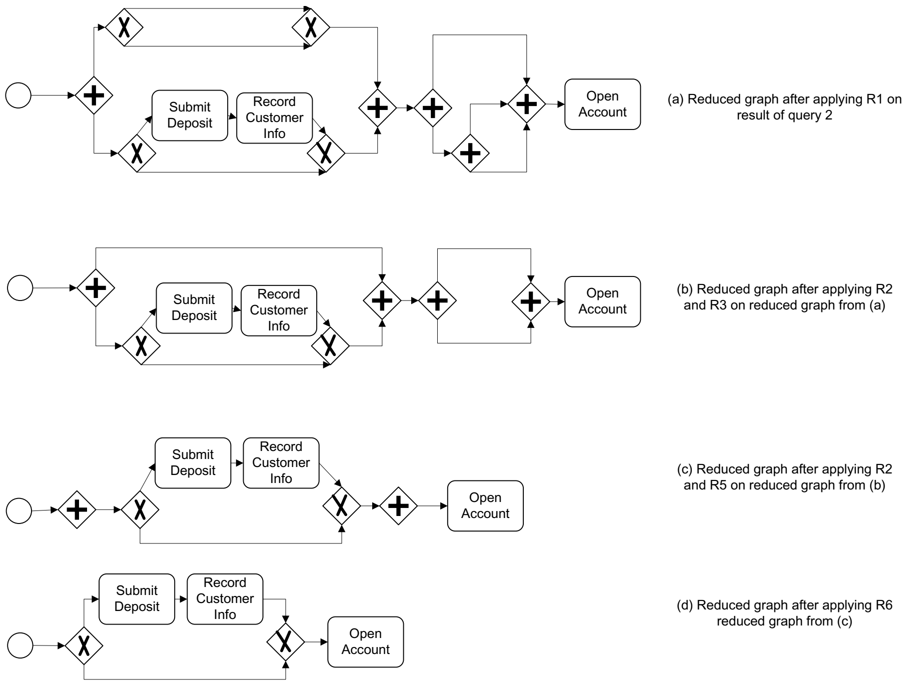

Submit Deposit
Record Customer Info
Open Account
(a) Reduced graph after applying R1 on result of query 2
(b) Reduced graph after applying R2 and R3 on reduced graph from (a)
(c) Reduced graph after applying R2 and R5 on reduced graph from (b)
(d) Reduced graph after applying R6 reduced graph from (c)

Fig. 11. Reduced graph for rule 2

introduces temporal quantifiers (always, eventually, next, until). The temporal operator *eventually* is of direct correspondence to the *leads to* concept. On the other hand, the translation of the *precedes* into a temporal expression in LTL would be complex. We used Past linear time temporal logic PLTL [17,29] as it has introduced the counterpart temporal quantifiers (always in past, once in the past, previous sate, since) to allow expressing formulae about the past states of a system. Although these quantifier did not increase the expressiveness of LTL, it made expressing formulae about the past exponentially succinct than in pure-future LTL [16]. Since the representation of these temporal quantifiers is not standardized, we mention the notation we use throughout this paper. For future states G(all future states), X(next state), F(eventually), U(Until). For past states H(all past states), Y(previous state), O(Once in the past), S(Since). It is straightforward to relate the concept of *precedes* to the temporal operator O and the concept of *leads to* to the operator F.

The generation of PLTL formulae from queries is straight forward. The following listing summarizes the translation into PLTL.

– A *leads to* B is translated to A $\rightarrow$F(B).
– A *precedes* B is translated to B $\rightarrow$O(A).

All generated formulae for different path constructs are conjuncted together and surrounded by the G operator to express the meaning of *in all possible execution scenarios*, be sure that the formulae are satisfied.

Query of rule 1 will generate the following temporal formula

```
  G((Obtain Customer Info →F(Open Account)) ∧ (Verify Customer ID
→F(Open Account)))
```

Model checking this formula against the reduced graph of Figure 10 will succeed, i.e. the process model complies with the rule. The query of rule 2 will generate the following temporal formula

```
G( (submit deposit →F(record customer info)) ∧
(open account →O(record customer info)))
```

Model checking this formula against the reduced graph of Figure 11 will fail, i.e. there are some execution scenarios that do not satisfy this formula.

Petri Net and State Space Generation

We need to generate, from the (reduced) graph, the finite state machine that will be, along with the PLTL expression, the input to model checker. In fact, we need to be sure that the state machine is finite, otherwise model checking is not feasible [3,8]. In order to generate the state machine and determine its finiteness, we have to give formal execution semantics to the different constructs of BPMN. We follow the approach introduced in [4]. In this approach a mapping of a subset of BPMN constructs (for example OR-join is not addressed) to Petri nets is given. We have implemented their algorithm in our tool, as will be shown later. The state machine is then obtained by the reachability graph of the Petri net. Finiteness of the state machine reduces to the boundedness of the net, so utilizing a Petri net tool to determine this property. Figure 12 is the generated Petri net for the reduced graph of Figure 11.

Output places of transitions corresponding to activities are of interest to indicate the execution of the activity — see labeled places of Figure 12. Further reductions on the level of a Petri net are possible provided that these places are not merged.

Table 1 shows the average running time (in milliseconds) of the model checker (MC) for checking rule 2 with and without reduction. Also, reduction time is

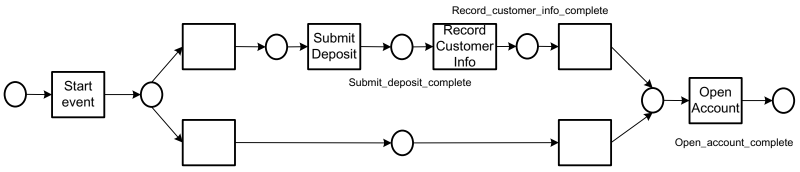


Fig. 12. Petri net for reduced process graph of Figure 11

Table 1. Comparison of average running time of model checker with and without using reduction

Process Model Id	No. Nodes	MC without reduction	MC with reduction
A	39	151 ms	110 ms
B	32	52 ms	31 ms
C	38	52 ms	48 ms
D	35	68 ms	52 ms

presented. The query of rule 2 matched 5 process models in the repository. One of them was excluded from model checking because it suffered from a deadlock.

Tool Support
We have implemented a prototypical tool chain for our approach. As process modelling environment we use Oryx[1], a web-based BPMN editor developed in our research group. We implemented a Petri net generator as an integrated component in BPMN-Q. LoLA [25][2] is used for checking boundedness and absence of deadlocks. LoLA is also used for producing the finite state machine. We implemented a PLTL generator returning the temporal logic formulae. As model checker we opted for NuSMV[3] due to its support for PLTL expressions.

5 Related Work

According to [22] checking for compliance can occur either *after-the-fact* or *before-the-fact*. Manually auditing processes is one way to check compliance in an *after-the-fact* fashion. An automated approach to detect violations from workflow logs using LTL checkers was introduced in [26].

Before-the -fact approaches can be further categorized as either (a) compliance-aware design or (b) post design verification. [22] is an example for compliance-aware design. Here, control objectives are modeled independently, that way addressing conflicting requirements between processes and regulations. The authors build their approach on a requirements modeling framework that later on propagates (forces) these requirements onto business processes. Another approach to guarantee compliance by design is given in [14], introducing PENELOPE as a declarative language to capture obligations and permissions imposed by business policies (sequencing and timing constraints between activities) and later on automatically generate business processes that are, by design, compliant with these policies. The same authors have discussed in [13] the importance of explicitly modeling business rules as an enabler of flexibility and generation of less complex business processes. A more recent approach to compliance by design is introduced in [21] which can be seen as an extension of [22] with a special focus on assisting the process designer to create compliant business processes. In

[1] See `http://oryx-editor.org`
[2] See `http://wwwteo.informatik.uni-rostock.de/ls_tpp/lola/`
[3] See `http://nusmv.irst.itc.it/`

[20] a more comprehensive framework where a categorization of control objective along with corresponding compliance patterns were discussed. They assume that process models are by default are not compliant. Later on, they are adapted and enriched with controls to be compliant with support of monitoring of violation at runtime.

We categorize our work as *before-the-fact* and *post design*. Other work in this category is briefly discussed. The Formal Contract Language (FCL) was introduced in [15] to formally measure the compliance between a business contract and a business process. In [12] an approach to check compliance of business processes and the resolution of violations was introduced. The paper defines Semantic Process Networks (SPN), where each activity is further annotated with effect predicates. Rules are then verified against this network. In [18], a formal approach based on model checking was given to check for compliance of processes defined in BPEL against constraints defined in the Business Property Specification Language (BPSL) that are translated to LTL. The approach is close to ours. However, we are able to express constraints in PLTL rather than LTL only, which gives our approach more expressiveness over the other. Similar work that verifies BPEL processes is in [28] where authors propose their own language PROPOLS to capture patterns to be checked against a business process. The approach depends on transforming PROPOLS expressions into FSAs and BPEL into LTS/FSA and check the language inclusion between the two FSAs. Work in [9,10,11] defines a set of visual patterns using the Process Pattern Specification Language (PPSL). These patterns are used to express constraints against UML Activity Diagrams. PPSL patterns are then translated into PLTL formulae. The approach is the closest to ours from the point of expressiveness i.e. it supports reasoning with PLTL, yet we offer a small set of constructs to express the same set of concepts. Similar work on verification of properties against UML Activity Diagrams has been accomplished earlier by Erik et al in [5,7,6], where they offered their own formalization of ADs and used model checking to verify properties against them.

Using queries for generating PLTL formulae, the retrieval of sub graphs to be tested and the application of reduction rules for simplifying the state space are unique properties of our approach.

6 Conclusion

In this paper we have presented an approach for compliance checking for BPMN process models using BPMN-Q queries. As centerpiece of the analysis, a model checker is used. The approach is not limited to BPMN process models, it can be applied to any process modelling language with formal execution semantics. The usage of BPMN-Q to express compliance rules was not limited to the graphical representation of the rules. BPMN-Q as a query language helps identify the set of process models that are subject to compliance checking, a task that is too time-consuming if done manually. The usage of reduction was to simplify the state space for the model checker, especially for large process models — which is the case in real world process models. For small process models, model checking can be applied directly without the overhead of reduction. This approach is effective under the assumption that business process models really reflect the way business is carried out. Although we assume that relevant (candidate) process models have to contain all activities mentioned in a rule, we still see our approach effective since the hard task is to check the ordering between activities. If a process model contains a non-empty subset of activities in a rule, we can conclude it non-compliant without further checking.

In the current version of our approach, we are able to give yes/no answers for the compliance between rule(s) and process models. As a limitation of our approach, the detailed analysis provided by the model checker in the case of non-compliance cannot be taken advantage of. This is due to the fact that we applied reductions. This means that generated counter examples by the model checker do not reflect real execution scenarios in process models. An important assumption behind the use of reduction rules is the relaxation of usage of temporal operators next X, and previous Y in queries. It has been pointed in literature that the X operator is of little interest when verifying properties of process models [5,7].

Currently, most research in the area of compliance checking focuses on verification of control flow aspects. As future work, we intend to extend BPMN-Q with the capability of querying data objects and verification of their states as pre-conditions for activities.

References

1. Sarbanes-Oxley Act of 2002. Public Law 107-204 (116 Statute 745), United States Senate and House of Representatives in Congress (2002)
2. Awad, A.: BPMN-Q: A Language to Query Business Processes. In: EMISA, pp. 115–128 (2007)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
4. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology (IST) (2008)
5. Eshuis, H.: Semantics and Verification of UML Activity Diagrams for Workflow Modeling. PhD thesis, Centre for Telematics and Information Technology (CTIT) University of Twente (2002)
6. Eshuis, R.: Symbolic model checking of uml activity diagrams. ACM Trans. Softw. Eng. Methodol. 15(1), 1–38 (2006)
7. Eshuis, R., Wieringa, R.: Tool support for verifying uml activity diagrams. IEEE Transactions on Software Engineering 30(7), 437–447 (2004)
8. Esparza, J.: Decidability of model checking for infinite-state concurrent systems. Acta Informatica 34(2), 85–107 (1997)
9. Förster, A., Engels, G., Schattkowsky, T.: Activity diagram patterns for modeling quality constraints in business processes. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 2–16. Springer, Heidelberg (2005)
10. Förster, A., Engels, G., Schattkowsky, T., Straeten, R.V.D.: A pattern-driven development process for quality standard-conform business process models. In: IEEE Symposium on Visual Languages and Human-Centric Computing VL (2006)
11. Förster, A., Engels, G., Schattkowsky, T., Straeten, R.V.D.: Verification of business process quality constraints based on visual process patterns. In: TASE, pp. 197–208. IEEE Computer Society, Los Alamitos (2007)

12. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
13. Goedertier, S., Vanthienen, J.: Compliant and Flexible Business Processes with Business Rules. In: 7th Workshop on Business Process Modeling (2006)
14. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes from Obligations and Permissions. In: 2nd Workshop on Business Processes Design (BPD 2006), Proceedings, Business Process Management Workshops (2006)
15. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: EDOC 2006, Washington, DC, USA, pp. 221–232. IEEE Computer Society Press, Los Alamitos (2006)
16. Hornus, S., Schnoebelen, P.: On solving temporal logic queries. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 163–177. Springer, Heidelberg (2002)
17. Laroussinie, F., Schnoebelen, P.: A hierarchy of temporal logics with past. Theoretical Computer Science 148(2), 303–324 (1995)
18. Lui, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal 46(2), 335–362 (2007)
19. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Institute of Information Systems and New Media Vienna University of Economics and Business Administration (WU Wien) Austria (May 2007)
20. Namiri, K., Stojanovic, N.: Pattern-based design and validation of business process compliance. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 59–76. Springer, Heidelberg (2007)
21. Lu, S.S.R., Governatori, G.: Compliance aware business process design. In: 3rd International Workshop on Business Process Design (BPD 2007), in Conjunction with 5th International Conference on Business Process Management (2007)
22. Sadiq, S.W., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007)
23. Sadiq, W., Orlowska, M.E.: Applying graph reduction techniques for identifying structural conflicts in process models. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 195–209. Springer, Heidelberg (1999)
24. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. 25(2), 117–134 (2000)
25. Schmidt, K.: Lola a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, p. 465. Springer, Heidelberg (2000)
26. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process mining and verification of properties: An approach based on temporal logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
27. van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Verification of epcs: Using reduction rules and petri nets. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 372–386. Springer, Heidelberg (2005)
28. Yu, J., Manh, T.P., Han, J., Jin, Y., Han, Y., Wang, J.: Pattern based property specification and verification for service composition. In: Aberer, K., Peng, Z., Rundensteiner, E.A., Zhang, Y., Li, X. (eds.) WISE 2006. LNCS, vol. 4255, pp. 156–168. Springer, Heidelberg (2006)
29. Zuck, L.: Past Temporal Logic. PhD thesis, Weizmann Intitute, Rehovet, Israel (August 1986)

Automatic Extraction of Process Control Flow from I/O Operations

Pedro C. Diniz[1] and Diogo R. Ferreira[2]

[1] IST/INESC-ID, Technical University of Lisbon, Portugal
[2] IST/INOV, Technical University of Lisbon, Portugal
{pedro.diniz,diogo.ferreira}@tagus.ist.utl.pt

Abstract. Many end users will expect the output of process mining to be a model they can easily understand. On the other hand, knowing which objects were accessed in each operation can be a valuable input for process discovery. From these two trends it is possible to establish an analogy between process mining and the discovery of program structure. In this paper we present an approach for extracting process control-flow from a trace of read and write operations over a set of objects. The approach is divided in two independent phases. In the first phase, Fourier analysis is used to identify periodic behavior that can be represented with loop constructs. In the second phase, a match-and-merge technique is used to produce a control-flow graph capable of generating the input trace and thus representing the process that generated it. The combination of these techniques provides a structured and compact representation of the unknown process, with very good results in terms of conformance metrics.

Keywords: Process mining, Control-flow graphs, Fourier analysis.

1 Introduction

Since the publication of [1], Petri nets became the preferred formal framework for the analysis [2], modeling [3], verification [4], mining [5] and conformance checking [6] of business processes. However, most workflow and Business Process Management (BPM) systems typically make use of proprietary modeling languages [7]. Despite efforts such as the development of workflow patterns [8], a vast community of end users still makes use of informal languages and notations, some of which have their origin in flowcharting [9], and many of which are reminiscent of basic programming concepts such as decision and loop constructs.

There is a number of reasons for the continuing use of such languages. First, it is often the case that the goal is to discuss process models with domain experts rather actually pursuing process analysis or enactment [10]. Second, there is often a perceived notion that process modeling is a kind of programming [11], hence the resemblance between modeling and programming constructs. Third, recent developments in Web service technology and SOA[1] have also contributed

[1] SOA: Service-Oriented Architecture.

to shorten the distance between process modeling and programming, as business processes may be implemented as compositions of web services [12]. Fourth, business analysts may be encouraged to use graphical languages when there are mechanisms to automatically translate them to executable models; for example, it is possible to translate process models in BPMN[2] to executable descriptions in BPEL[3] [13].

This sort of top-down implementation of business processes is in some way available in every BPM system (see for example [14]). But if we look at bottom-up approaches, and in particular to the problem of discovering business processes from event logs, we realize that current process mining techniques are able to recover process models in a representation that might not always match what the end users may be familiar with. Some techniques generate dependency graphs [15,16], others use probabilistic models [17,18], and most of the current techniques are geared towards retrieving Petri net models [19]. To communicate with end users, it may be necessary to translate Petri nets into other kinds of models, including business process notations such as EPC[4] [20,21].

In this paper we present a technique for extracting process behavior directly as a structure of programming constructs. Although we make use of just a couple of basic constructs, namely decisions and loops, it is possible to extend the same technique to more intricate elements. The goal is to extract a control-flow graph whose structure and building blocks resemble those of a computer program written in an imperative programming language. The nature and scope of business processes are obviously quite different from those of software programs, but a representation in terms of programming constructs may help the end user relate more easily to the results of process mining over a given event log.

The analogy can be extended even further when we consider the content of the input log. Typically, process mining techniques require that each event in the input log is associated with a specific process instance [5]; in some scenarios this context may be unavailable. On the other hand, process mining techniques focus mainly on the control-flow perspective and only recently have begun to take into account information about data or objects being accessed, which makes it possible to identify data dependencies between events [22]. The availability of such information becomes critical when the process models must comply with known object life-cycles [23].

In this work we consider that the input event log is available as a trace of all `read` and `write` operations over a set of objects. Such read and write operations could have been recorded, for example, as accesses to document repository, or even to a version-control system. They can also be regarded as low-level events such as memory, disk, or database accesses. The objects that are accessed in these operations can be regarded either as workflow-relevant data [24] or as program variables. The log contains no information about activity, process instance, or

[2] BPMN: Business Process Modeling Notation.

[3] BPEL: Business Process Execution Language for Web Services.

[4] EPC: Event-Driven Process Chain.

business context; it is simply a trace of all I/O operations performed by an unknown process, whose structure is to be determined.

The remainder of this paper is structured as follows. In section 2 we provide an overview of the overall approach, comprising two phases: the first phase which finds the boundaries of loops and their control variables, and the second phase which merges the sub-traces of each loop into a well-structured control-flow graph (CFG). These algorithms are explained in detail in sections 3 and 4, respectively. In section 5 we present experimental results on the application of the described approach to a set of sample traces, and evaluate the results according to a set of conformance metrics defined in [6]. Finally, section 6 concludes the paper.

2 Overview

In the following sections we describe an approach for automatically uncovering a possible control-flow representation from a sequential input trace of I/O operations. In this context, the input trace is regarded as a numbered sequence of basic `read` and `write` operations on a set of variables. Each variable, $v_i \in V$ represents an object in the specific domain of objects for the process (*e.g.*, a document, or a database table or row) that is read or updated (written).

The approach is structured into two major phases as depicted in figure 1. In a first phase, we rely on Fourier analysis of the input trace to detect operations that occur in repeatable patterns or time slots in the trace thus exhibiting some periodic behavior. This analysis determines if periodic behavior is present and which variable(s) are possible *loop control* variables, *i.e.* variables that control the execution of loop constructs in the process structure that generated the observed trace.

At the end of the first phase, the approach has identified the runs of existing loops and separated them into sub-traces of the original input trace. In a second and completely separate phase, the algorithm creates a CFG by matching and merging identical operations in the different sub-traces, while respecting the sequential relation between the operations in each sub-trace and in the resulting CFG.

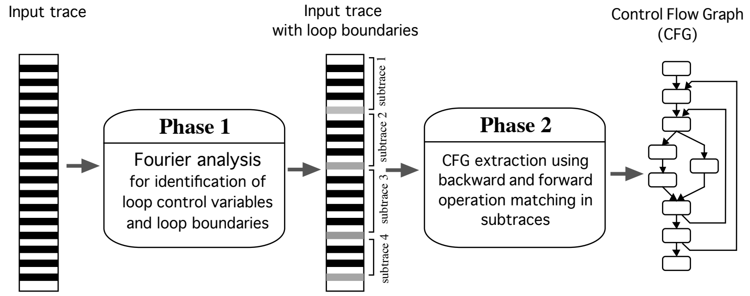

Input trace
Input trace with loop boundaries
Control Flow Graph (CFG)
Phase 1
Fourier analysis for identification of loop control variables and loop boundaries
subtrace 1
subtrace 2
subtrace 3
subtrace 4
Phase 2
CFG extraction using backward and forward operation matching in subtraces

Fig. 1. Overall approach

3 The Loop-Finding Algorithm

The first phase of the algorithm aims at finding loop constructs. We first describe the basic algorithm to uncover a single loop construct and then describe how to use this approach to find nested loop structures.

3.1 Fourier Transformation

To find a loop construct in a given section of a large trace the algorithm makes use of a digital signal processing technique – the Discrete Fourier Transform (DFT). Fourier techniques [25] translate periodic time-domain signals to the frequency domain. A periodic time-domain signal such as a sine function is represented in the frequency domain by a single magnitude and phase coefficient as depicted in figure 2(a). The Fourier transform of a generic periodic time-domain signal is a series of magnitude and phase coefficients, each corresponding to one of the harmonic frequencies of the given signal. This frequency-domain representation is called the spectrum of the input signal; a high-frequency composition on the spectrum reveals a fast changing signal and a low-frequency composition reveals a slow changing signal. For discrete periodic signals the same Fourier decomposition is possible and a periodic discrete signal with periodicity of T will have a frequency-domain representation exhibiting peaks separated by $1/T$ as illustrated in figure 2(b).

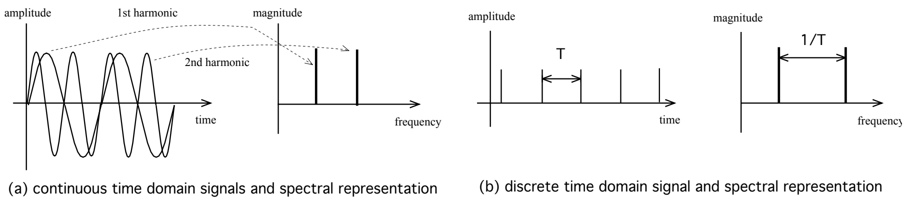

amplitude
1st harmonic
magnitude
2nd harmonic
time
frequency
(a) continuous time domain signals and spectral representation
amplitude
T
magnitude
1/T
time
frequency
(b) discrete time domain signal and spectral representation

Fig. 2. Illustrative examples of Fourier Transformations (FT/DFT)

3.2 Algorithm Rationale and Description

The basis for the loop-finding algorithm relies on the observation that if a specific operation over a variable has a predominately periodic behavior, its *signature* signal must have a frequency-domain representation exhibiting clearly spaced peaks reflecting the periodicity of the time-domain signature signal. As such, one is able to uncover the periodicity associated with a specific variable and operation by examining its spectral representation.

For a given variable v and operation op (either `read` or `write`) the algorithm computes its signature signal $sig_{v,op}(k)$ as a time-domain discrete signal. This discrete signal is composed of N samples $0 \leq k \leq N-1$, one for each time sample

$$S_{v,op}(n) = \frac{1}{N}\sum_{k=0}^{N-1} sig_{v,op}(k)e^{-jk2\pi n/N}$$

$$\|S_{v,op}(n)\| = \sqrt{S_{v,op}^{real}(n)^2 + S_{v,op}^{imag}(n)^2}$$

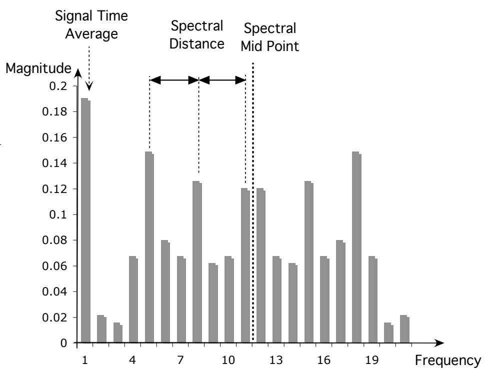


Fig. 3. DFT calculation expressions (left) and spectral example (right) where $sig_{v,op}$ is the discrete signature signal for variable v and operation op and $S_{v,op}(n)$ is its spectral representation for N spectral frequencies n/N for $0 \leq n \leq N-1$

in the input trace. The signature signal $sig_{v,op}(k)$ has value 1 for positions in the trace with an operation *op* over the v variable, and 0 otherwise. Figure 4(b) depicts a sample signature for the variable `i` and the `write` operation. The algorithm computes the Fourier transform of the signature signal for the variable v. It derives its spectral representation as a set of N complex coefficients, $S_{v,op}(k)$ with $0 \leq k \leq N-1$ and computes the corresponding magnitude $\|S_{v,op}(n)\|$ of the coefficients given by the norm-2 of its real and imaginary components as depicted by the equation in figure 3 (left).

Once the spectral representation is computed, the algorithm examines the peaks in the spectrum and determines the distances between peaks to discover the various base frequencies of the original time-domain signal. To accomplish this step, the algorithm selects the middle point in the spectral representation (*i.e.,* the coefficient at $N/2$) and measures the distance from that middle point to the next peak[5]. The inverse of this distance d is the frequency with which the operation *op* on variable v occurs in the time-domain signal.

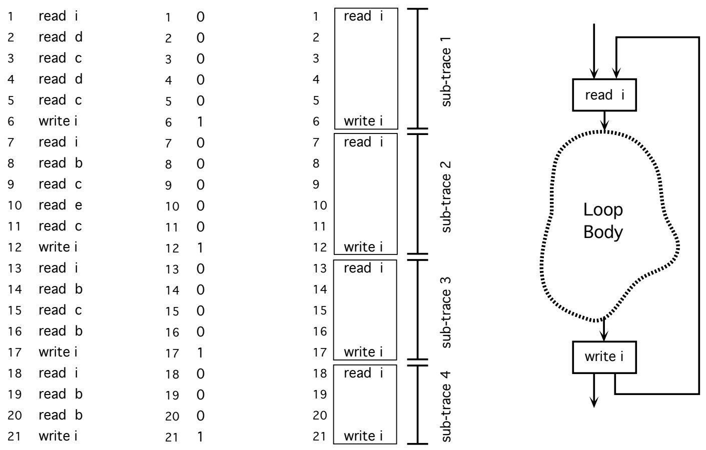
In figure 3 (right) we illustrate an example of a spectral representation of a periodic signal corresponding to an input trace with 21 operations. Excluding the zero-frequency component (which corresponds to the average or DC component of the signal) the spectral information for the signal exhibits distinct spikes separated by 3 time units in the frequency axis. In the time domain, this frequency mode corresponds to a repetition interval of approximately $(N-1)/d = (21-1)/3$ time units, *i.e.* between 6 and 7 events.

[5] A practical complication arises regarding the ability to clearly identify the peaks in the Fourier representation. The algorithm uses a thresholding step where all values below a specific percentage τ of the maximum value are eliminated. To select the value of threshold τ, the algorithm applies the Fourier analysis with values of $\tau = 1.0$ down to $\tau = 0.1$ and stops when it uncovers a feasible time-domain period.

Once the period(s) for each variable in the input trace has been identified, the algorithm selects as the loop-controlling variable the variable with the shortest period but with the highest number of occurrences in the trace[6]. Given this control variable the algorithm then scans the input trace selecting the occurrences of the variable that are located at approximate intervals corresponding to the identified period. As there can be some slight variations to the location of the occurrences of this control variable in the trace, the algorithm uses a simple windowing scheme to sync-up their occurrences in the trace.[7]

3.3 Example

Figure 4(a) presents an example of an input trace with a total of 21 `read` and `write` operations over a set of variables. For the variable `i` and operation `write` the algorithm uses Fourier analysis as depicted in figure 3 to uncover a period of approximately 6 time units, the period of `i` as the control variable for a loop construct. The algorithm then performs a linear scan over the input trace and splits the input trace into 4 sub-traces corresponding to the ranges of operations depicted in figure 4(c). The algorithm also detects a common initial operation `read i` in addition to the common final operation `write i` thus defining the entry and exit nodes of the CFG loop structure depicted in figure 4(d).



1	read i	1	0
2	read d	2	0
3	read c	3	0
4	read d	4	0
5	read c	5	0
6	write i	6	1
7	read i	7	0
8	read b	8	0
9	read c	9	0
10	read e	10	0
11	read c	11	0
12	write i	12	1
13	read i	13	0
14	read b	14	0
15	read c	15	0
16	read b	16	0
17	write i	17	1
18	read i	18	0
19	read b	19	0
20	read b	20	0
21	write i	21	1

(a) Initial Trace (b) Signature signal sig(i,write) (c) Uncovered sub-traces for the write operation for variable i with loop boundaries (d) Control-Flow-Graph (CFG) considering loop constructs only and based on the sub-traces in (c)

Fig. 4. Sample illustrative trace with 21 operations

[6] So that a spurious variable that occurs often but randomly is not selected.

[7] Spurious occurrences of an operation at time stamps that are not lined up with the recognized interval are not critical, as the second phase of the algorithm absorbs these instances by taking them into account as different paths in the loop control flow.

3.4 Handling Nested Loops

The approach described above enables the same algorithm to discover nested loops. The algorithm works inside-out by first finding shorter loops in the trace, corresponding to inner-loops. These loops have control variables with high spectral frequencies, as they occur more frequently than the control variables of outer loops. In order to recognize successive outer loops, the algorithm collapses the operations within an inner loop as a single aggregate macro operation and then performs a new Fourier analysis on the collapsed trace, attempting to uncover the next innermost loop. This procedure is repeated until the structure of nested loops is found and the trace collapses to a small segment where no more loops can be detected.

4 The Control-Flow Algorithm

In this second phase, the algorithm uses the sub-traces extracted during the first phase to create a well-structured CFG that can generate the entire input trace.

4.1 Algorithm Outline

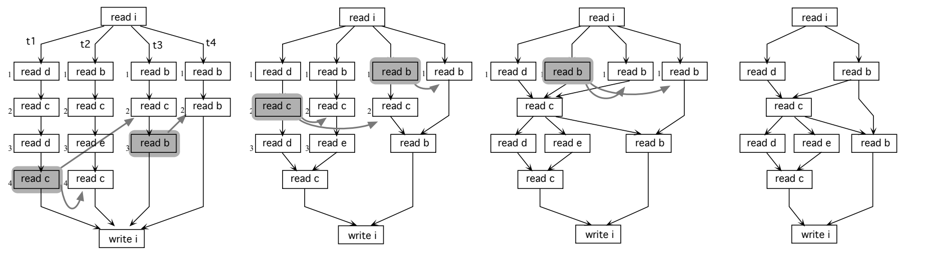
The algorithm is structured into two iterative, greedy refinements phases. It terminates when it cannot refine the CFG any further. The algorithm begins by building a CFG with all the sub-traces as possible (disjoint) paths. These paths are connected to the same *source* and *sink* nodes, which represent the entry and exit nodes of a loop construct. In a first pass, the algorithm works bottom-up against the flow of the sub-traces, creating nodes in a new CFG that correspond to the operations in the sub-traces, and whenever possible it merges two identical nodes that share a common descendant node in the graph.[8] In a second pass the algorithm works top-down along the flow, this time in the CFG resulting from the first pass, and merges any two identical nodes with a common ancestor node. At the end of these two match-and-merge phases the resulting CFG is augmented by a *back edge* which connects two nodes, respectively the *head* and *tail* of a loop, thus reflecting the iterative structure of the input trace.

There is no reason not to invert the order of these two passes as both ways will allow the algorithm to derive valid but possibly different CFGs. This may produce a slight difference in the placement of decision points, without a noticeable impact in the metrics presented ahead in section 5.

4.2 Backward Match-and-Merge

In this pass the algorithm attempts to merge nodes from each sub-trace with nodes from other traces as close as possible in the linear sequence within each trace. The merged nodes become a single node shared between sub-traces as

[8] It should be noted that at the beginning of this first pass the sink node is a common descendant of all paths.



(a) Initial set-up phase CFG and matching between sub-traces. (b) CFG after merging of nodes in traces t1 and t2 and traces t3 and t4. (c) CFG after merging of nodes in traces t1, t2 and t3. (d) Final CFG after merging of nodes in traces t2, t3 and t4.

Fig. 5. Backward Match-and-merge operation with 4 illustrative traces

depicted in figure 5(b). The current greedy strategy for matching and merging of nodes gives preference to matches that are chronologically closer, as that increases the likelihood of merging more nodes across the sub-traces. For example, although there is a match in figure 5(a) between the operation `read c` in the sub-trace t_1 and the same operation in the sub-trace t_3, the algorithm does not merge these two operations as there is a match from an operation `read b` in sub-trace t_3 that occurs later along the control flow than that first match. As a result, the first `read c` is only matched with the same operation in sub-trace t_2 and the operation in `read b` is matched with the same operation in sub-trace t_4. Such greedy matching process continues as depicted in figure 5(b) and 5(c) until there are no more possible matches, resulting in the CFG depicted in figure 5(d).

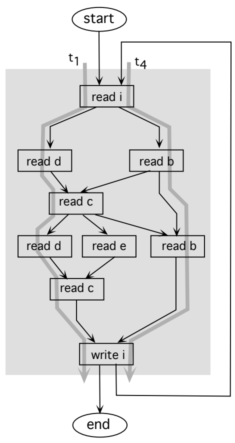
4.3 Forward Match-and-Merge

After having constructed a CFG in the first backward pass, the algorithm performs a forward match-and-merge to attempt to merge nodes that might have been left unmatched in the first phase[9]. This forward pass is also a greedy process where the algorithm iteratively merges consecutive adjacent nodes along the control flow until no further merges are possible or the end of sub-traces is reached. For the CFG derived at the end of the first step as depicted in figure 5(d), there are no opportunities for this particular transformation. The algorithm merges no nodes in this second pass and the final CFG with loop control-flow is shown in figure 6(a). As an illustrative example we depict in this final CFG two execution paths corresponding to the sub-traces t_1 and t_4 in the input sub-traces.

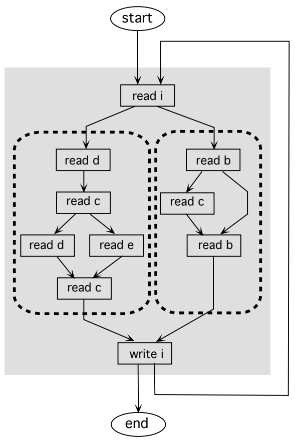
4.4 Structured Control-Flow

While the algorithm described above is effective in finding common subsequences among all traces preserving the sequential execution of the control flow, it can generate control-flow structures that in the lexical sense are not perfectly nested. These imperfectly nested control-flow structures cannot be described

[9] A typical scenario occurs when two or more sub-traces get *out-of-sync* and the corresponding nodes are then left unmatched.



(a) Complete CFG with loop construct and sample execution paths through the loop body.



(b) Complete CFG with loop construct using structured aggregation of nodes during CFG construction.

```
begin
  while (Pred1) do
     read i;
     if (Pred2) then
           read d;
           read c;
           if (Pred3) then
              read d;
           else
              read e;
           end if
           read c;
     else
        read b;
        if (Pred4) then
           read c;
        endif
        read b;
     end if
      write i;
  end while
end
```

(c) High-level programming constructs for CFG derived in (b) using generic conditional and loop predicates.

Fig. 6. Complete CFG with loop as derived by the algorithm

using high-level programming constructs without the use of goto or other arbitrary control-flow transfer primitives [26]. More importantly, in the context of process flows, it substantially complicates conformance checking between processes.

To overcome this problem we developed a variant of the control-flow algorithm that always generates structured CFGs at the expense of the loss of some sharing of nodes between sub-traces. The revised algorithm keeps the matching of nodes confined to a subset of traces that have been already matched in a previous matching step. The algorithm thus partitions the input traces into disjoint subset of traces which are then refined as the matching progresses. As a result, the matching and thus the merging is always done between traces that have a common descendant towards the sink node. The resulting CFG is thus less compact than the CFG generated by the first control-flow algorithm.

Figure 6 illustrates the CFGs resulting from the two variants of the algorithm for the same 4 illustrative sub-traces used in figure 5. Figure 6(a) presents the CFG resulting from the application of the first algorithm, whereas figure 6(b) presents the CFG obtained using the variant of the algorithm that generates structured CFGs. For this second CFG we depict the corresponding high-level program description that can generate the various traces in figure 6(c).

4.5 Algorithmic Complexity

Given that both the backward and the forward passes traverse and advance through each sub-trace at least one operation at a time in each iteration, the algorithm eventually terminates. At each iteration the algorithm performs in the worst case $O(n^2)$ when matching operations between the k sub-traces. On the other hand, each sub-trace is $O(n)$ long and this length is linearly related to the length of the input trace. Considering book-keeping and the manipulation of auxiliary data structures as a constant, the worst-case time complexity becomes $O(n^3)$ where n is the number of operations in each sub-trace.

Despite this worst-case complexity behavior, we do anticipate the algorithm to perform well given that at each step more than one node can be processed.[10]

5 Results

We have implemented the above algorithms in approximately $4,000$ lines of C code. In this section we describe a series of experiments with sample traces to evaluate the ability of the algorithm to extract a CFG for each input trace. In all these experiments we have used the revised version of the algorithm in order to derive structured CFGs. We then evaluated each of the derived CFGs according to a set of fitness and appropriateness metrics defined in [6].

5.1 Sample Traces

To support our experiments we generated a set of 4 traces with `read` and `write` operations over a set of scalar variables such as `i` or `a`.[11] Each trace is generated by a different C program that outputs the various operations reflecting its own execution.[12]

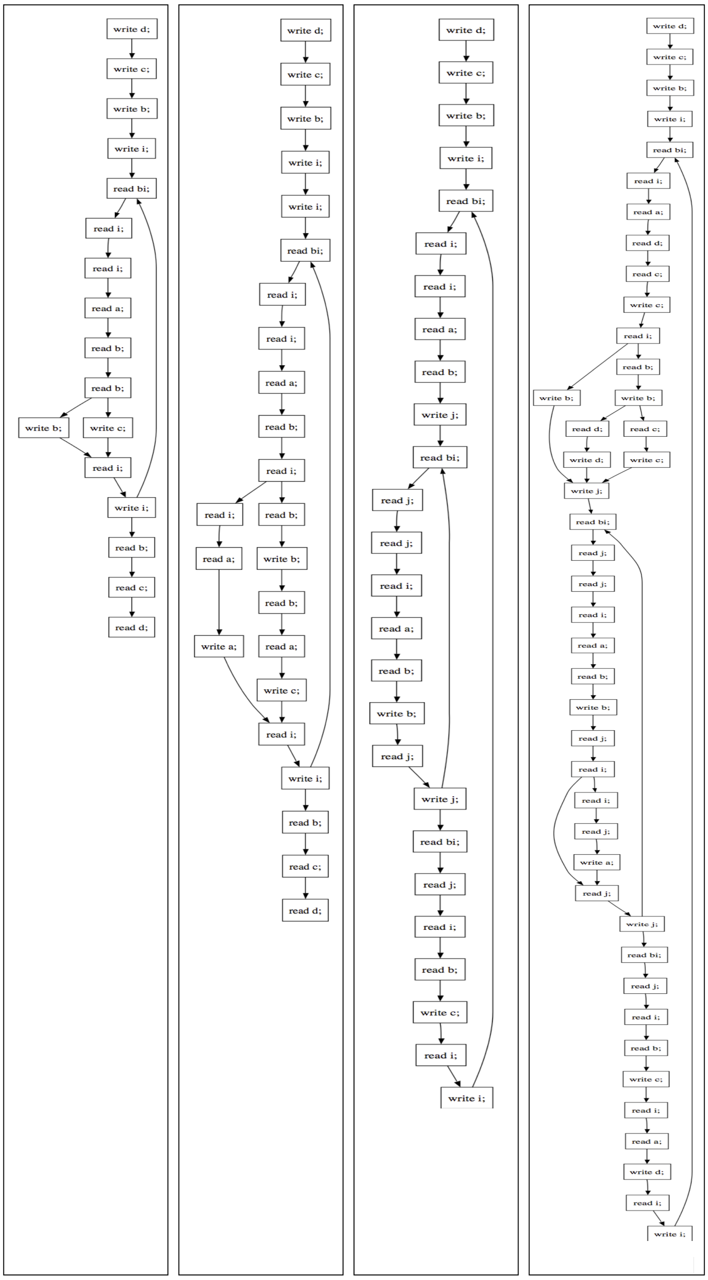
Table 1 presents the experimental results for the various traces. In the left section of the table we have a series of results characterizing the traces used, whereas on the right section we have results regarding metrics as discussed in the next section.

In table 1 we characterize each trace by its length or number of operations, and also by the number of variables that each trace contains. These are shown in columns 2 and 3, respectively. Column 4 presents the control variables extracted by the Fourier analysis phase of our algorithm, indicating for each variable the uncovered period (as an interval of two values). In column 5 we report the number of sub-traces identified for each control variable, and in column 6 we indicate the nested structure of the discovered CFG. Figure 7 shows the CFGs discovered for the 4 sample traces.

[10] Common string-matching algorithms such as the longest common sub-sequence (LCS) or the shortest common super-sequence (SCS) problems are impractical. Both LCS [27] and SCS [28] problems require $O(n^2)$ solutions for two sequences of length n, but are NP-hard for k strings. An incremental 2-at-a-time approach would lead to a time complexity of $O(n^k)$.

[11] The various input traces and CFG outputs are available at the following location: http://www.dei.ist.utl.pt/∼ped/submissions/BPM08

[12] For example, in the execution of loop constructs the trace generator will output the operations required to evaluate the loop control predicate. Updating the loop control variable, say `i = i + 1;` involves a read operation followed by a write operation.



(a) CFG for input trace 1 (b) CFG for input trace 2 (c) CFG for input trace 3 (d) CFG for input trace 4

Fig. 7. Complete CFGs for the 4 sample traces

Table 1. Characteristics of the sample traces used in experiments, and metrics for the derived CFG solutions

Trace	Characteristics					Metrics						
	number operations	number variables	control var. and period	number sub-traces	nesting structure	c_E	c_T	f	a_B	a'_B	a_S	a'_S
#1	52	7	i:[8,9]	5	single loop	1.000	1.000	1.000	0.988	0.924	0.588	1.000
#2	109	6	i:[12,13]	8	single loop	1.000	1.000	1.000	0.994	0.849	0.458	1.000
#3	1034	6	i:[101,102] j:[8,9]	10 101	doubly nested	1.000	1.000	1.000	0.998	1.000	0.407	1.000
#4	1108	7	i:[121,122] j:[8,9]	10 82	doubly nested	1.000	1.000	1.000	0.996	0.953	0.311	1.000

5.2 Metrics

The algorithms described above produce a CFG that is in effect one possible explanation for the behavior observed in the input trace. To assess the interest and potential of this approach, and to be able to compare it with other process mining algorithms, it is necessary to determine the extent to which the produced CFG is actually a good explanation for the original behavior. This assessment can be done by applying a set of metrics to compare the behavior allowed in the model with the behavior observed in the input trace. Such analysis can be done, for example, by checking if the process model would allow all events in the trace to occur in the same order. This is the underlying rationale for the *fitness* metric [6]. Other metrics – such as *behavioral appropriateness* and *structural appropriateness* [6] – are also useful to check that the model is a compact and non-redundant representation of the intended process.

Even though these metrics have been defined for Petri net models, it is possible to apply them to CFGs with small adaptations. The only metric that requires a significant adaptation is the *fitness* metric, as it is based on replaying the trace in the model. While the execution semantics of Petri nets are formally well-established (via tokens, places and transitions), those of CFGs may vary; for example, splits and joins could be given different interpretations. In the CFGs we use above, there are only OR-splits and OR-joins so the semantics become quite simple. We therefore redefined the *fitness* metric as the percentage of trace events that can be successfully replayed in the CFG in the same order.

Table 1 shows the results obtained for each trace and metric. Table 2 summarizes the purpose of each metric and provides a brief explanation of the typical results obtained in several runs of the algorithm for different input traces. A detailed description of these metrics can be found in [6].

Table 2. Applying the metrics defined in [6]

Metric	**Results**
Log coverage: $c_E = \frac{\lvert\{e \in E \mid l_E(e) \in L_T\}\rvert}{\lvert E \rvert}$	This metric checks if each event in the input trace is represented as a node label in the CFG. Since they all are, the metric is always 100%.
Model coverage: $c_T = \frac{\lvert\{t \in T_V \mid l_T(t) \in L_E\}\rvert}{\lvert T_V \rvert}$	This metric checks if each label in the CFG appears as an event in the trace. Since they all do, the metric is always 100%.
Fitness: $f = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right)$	This metric checks whether events can be replayed in the model in the same order as they appear in the trace. It is originally defined in terms of consumed (c) and produced tokens (p), as well as missing tokens (m) during replay and remaining tokens (r) after replay. Since we are using CFGs rather than Petri nets, this metric has been redefined as the percentage of trace operations that can be replayed in the same order in the CFG. As the trace is always a possible path in the resulting CFG, the metric is always 100%.
Simple behavioral appropriateness: $a_B = \frac{\lvert T_V \rvert - x}{\lvert T_V \rvert - 1}$	This metric uses the number of *visible tasks* $\lvert T_V \rvert$ (which in our case is the number of CFG nodes) and the mean number (x) of possible transitions at each step when replaying the trace. As the produced CFGs usually have a few decision points, at some steps there is more than one option for the next step, and hence the metric is usually close but not equal to 100%. Depending on trace length and the number of decision points, it is usually above 99%.
Advanced behavioral appropriateness: $a'_B = \frac{\lvert S^l_F \cap S^m_F \rvert}{2 \cdot \lvert S^m_F \rvert} + \frac{\lvert S^l_P \cap S^m_P \rvert}{2 \cdot \lvert S^m_P \rvert}$	This metric computes the *follows* and *precedes* relations both for the trace (S^l_F and S^l_P) and for the CFG (S^m_F and S^m_P). These relations basically say whether any given pair of operations *always*, *sometimes* or *never* follow each other. Based on these relations, the metric checks whether the CFG allows for more variability in behavior than that observed in the trace. Due to the presence of decision points, the CFG usually does allow for more variability, hence the metric is typically in the range 85%-95%.
Simple structural appropriateness: $a_S = \frac{\lvert L \rvert}{\lvert N \rvert}$	This metric checks the amount of duplicate nodes in the CFG (*i.e.* nodes with the same label) by dividing the number of distinct labels $\lvert L \rvert$ by the number of nodes $\lvert N \rvert$. As the match-and-merge procedure avoids merging nodes that would cause the model to become unstructured, it is usually the case that a number of duplicate nodes remain in the CFG. In relatively complex CFGs, the metric can be as low as 30%. It should be noted, however, that the true model (*i.e.* the program that generated the input trace) may have duplicate operations, so this simple metric is not a reliable measure of accuracy in our experiments.
Advanced structural appropriateness: $a'_S = \frac{\lvert T \rvert - (\lvert T_{DA} \rvert + \lvert T_{IR} \rvert)}{\lvert T \rvert}$	This metric penalizes the model when there are *redundant invisible tasks* (T_{IR}) or *alternative duplicate tasks* (T_{DA}). An *invisible task* is an operation that is represented in the model but is not present in the trace; in our CFGs there are no invisible tasks. Two CFG nodes are said to be *alternative duplicate tasks* if they have the same label but there is no possibility that they could occur in the same trace. The match-and-merge steps in our algorithm eliminate these alternative duplicate nodes, hence the value for this metric is always 100%.

6 Conclusion

The analogy between process modeling and programming suggests that it should be possible to approach process mining from the perspective of discovering program structure. This is useful not only because process modeling and execution languages make use of building blocks that resemble programming constructs, but also because the way a program operates on variables can be seen as being analogous to the way workflow participants manipulate documents and other data objects.

In this paper we described a combination of techniques to extract process behavior directly as a control-flow graph from a trace of all read and write operations over a set of objects. The approach is divided in two main phases where, in the first phase, signal processing techniques are used to detect periodic behavior that can be potentially represented with loop constructs. This is an interesting challenge since the problem of delimiting repeating behavior is not addressed by current process mining techniques, and the DFT proves to be an effective tool for this purpose. It also enables the algorithm to uncover nested loops, which can be abstracted as sub-processes. Furthermore, as a signal processing technique it is inherently more robust to noise than discrete algorithmic approaches.

In the second phase, the algorithm proceeds with a set of unsupervised match-and-merge steps to produce a structured graph by consolidating the behavior of different sub-traces, and by reducing their differences to a set of decision points. This technique can be used as a process mining algorithm by itself, if several traces of the same process are provided as input. In this case, the algorithm would be equivalent to beginning with M3 in [6] and proceed by merging nodes until an appropriate model is found. The match-and-merge procedure described in section 4.4 ensures that the outcome is a structured model.

Overall, the combination of these techniques is able to produce compact and accurate models of control-flow behavior, and exhibits very good results in terms of conformance metrics. Although only loops and decision constructs have been addressed in the present work, we are currently studying techniques to support other behavioral patterns, in particular parallel constructs.

Acknowledgment

The second author would like to acknowledge the support of *Fundação para a Ciência e a Tecnologia* (FCT) through project no. PTDC/CCI/70512/2006.

References

1. van der Aalst, W.: The application of petri nets to workflow management. Journal of Circuits. Systems and Computers 8(1), 21–26 (1998)
2. van der Aalst, W.: Woflan: a petri-net-based workflow analyzer. Systems Analysis Modelling Simulation 35(3), 345–357 (1999)
3. van der Aalst, W.: Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. Information and Management 37(2), 67–75 (2000)
4. van der Aalst, W., ter Hofstede, A.: Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. Information Systems 25(1), 43–69 (2000)
5. van der Aalst, W., Weijters, A., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
6. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
7. van der Aalst, W., ter Hofstede, A.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
8. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
9. Rosemann, M., Recker, J., Indulska, M., Green, P.: A study of the evolution of the representational capabilities of process modeling grammars. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 447–461. Springer, Heidelberg (2006)
10. van der Aalst, W., ter Hofstede, A., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
11. Yang, G.: Process library. Data and Knowledge Engineering 50(1), 35–62 (2004)
12. Emig, C., Weisser, J., Abeck, S.: Development of SOA-based software systems – an evolutionary programming approach. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006), Washington, DC, p. 182. IEEE Computer Society, Los Alamitos (2006)
13. Ouyang, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: Pattern-based translation of BPMN process models to BPEL web services. International Journal of Web Services Research 5(1), 42–61 (2008)
14. Kloppmann, M., Knig, D., Leymann, F., Pfau, G., Roller, D.: Business process choreography in websphere: Combining the power of bpel and J2EE. IBM Systems Journal 43(2), 270–296 (2004)
15. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
16. Greco, G., Guzzo, A., Pontieri, L.: Mining hierarchies of models: From abstract views to concrete specifications. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 32–47. Springer, Heidelberg (2005)
17. Herbst, J., Karagiannis, D.: Integrating machine learning and workflow management to support acquisition and adaption of workflow models. In: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, pp. 745–752. IEEE, Los Alamitos (1998)

18. Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 360–374. Springer, Heidelberg (2007)
19. van der Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
20. van Dongen, B., van der Aalst, W.: Multi-phase process mining: Building instance graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)
21. Verbeek, H., van Dongen, B.: Translating labelled P/T nets into EPCs for sake of communication. BETA Working Paper Series WP 194, Eindhoven University of Technology (2007)
22. Rozinat, A., Mans, R., van der Aalst, W.: Mining CPN models: Discovering process models with data from event logs. In: Proceedings of the Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006), Denmark, University of Aarhus, pp. 57–76 (2006)
23. Kuester, J., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
24. Hollingsworth, D.: The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition (1995)
25. Oppenheimer, A., Shaffer, R.: Digital Signal Processing. Prentice-Hall, Englewood Cliffs (1975)
26. Aho, A., Sethi, R., Ullman, J.: Compilers: Principles, Techniques and Tools. Addison-Wesley, Inc., Reading (1986)
27. Hirschberg, D.S.: Algorithms for the longest common subsequence problem. Journal of the ACM 24(4), 664–675 (1977)
28. Fraser, C., Irving, R.: Approximation algorithms for the shortest common supersequence. Nordic Journal of Computing 2(3), 303–325 (1995)

A Region-Based Algorithm for Discovering Petri Nets from Event Logs

J. Carmona[1], J. Cortadella[1], and M. Kishinevsky[2]

[1] Universitat Politècnica de Catalunya, Spain
[2] Intel Corporation, USA

Abstract. The paper presents a new method for the synthesis of Petri nets from event logs in the area of Process Mining. The method derives a bounded Petri net that over-approximates the behavior of an event log. The most important property is that it produces a net with the smallest behavior that still contains the behavior of the event log. The methods described in this paper have been implemented in a tool and tested on a set of examples.

1 Introduction

The discovery of formal models from event logs in information systems is known as *process mining*. Since the nineties, the area of process mining has been focused in providing formal support to business information systems [16]. In the industrial domain, ranging from hospitals and banks to sensor networks or CAD for VLSI, process mining can be applied to succinctly summarize the behavior observed in large event logs [14]. Nowadays, several approaches can be used to mine formal models, most of them included in the ProM framework [15].

The *synthesis problem* [7] is related to process mining: it consists in building a Petri net that has a behavior equivalent to a given transition system. The problem was first addressed by Ehrenfeucht and Rozenberg [8] introducing *regions* to model the sets of states that characterize marked places. Process mining differs from synthesis in the knowledge assumption: while in synthesis one assumes a complete description of the system, only a partial description of the system is assumed in process mining. Therefore, bisimulation is no longer a goal to achieve in process mining. Instead, obtaining approximations that succinctly represent the log under consideration are more valuable [19].

In the area of synthesis, some approaches have been studied to take the theory of regions into practice. In [3] polynomial algorithms for the synthesis of bounded nets were presented. This approach has been recently adapted for the problem of process mining in [4]. In [6], the theory of regions was applied for the synthesis of safe Petri nets with bisimilar behavior. Recently, the theory from [6] has been extended to bounded Petri nets [5]. In this paper we adapt the theory from [5] to the problem of process mining.

The work presented in this paper aims at constructing (mining) a Petri net that *covers* the behavior observed in the event log, i.e. traces in the event log

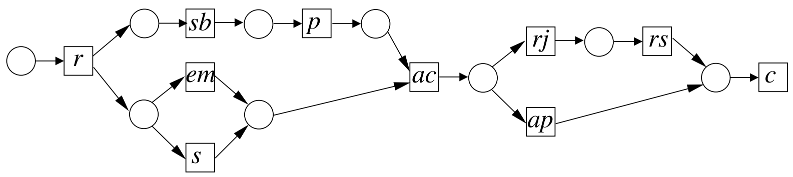


Fig. 1. Petri net mining to avoid overfitting

will be feasible in the Petri net. Moreover, the Petri net may accept traces not observed in the log. Additionally, a minimality property is demonstrated on the mined Petri net: no other net exists that both covers the log and accepts less traces than the mined Petri net. This capability of minimal over-approximation represents the main theoretical contribution of this paper. The methods presented in the paper can mine a particular k-bounded Petri net, for a given bound k. We have implemented the theory of this paper in a tool, and some preliminary results from logs are reported. The approach taken in this paper is a formal one and differs from the more heuristic methods in the literature. Although the methods presented might have a high complexity for large logs, they can be combined with recent iterative approaches [18] to alleviate their complexity.

This paper shares common goals with the previously presented paper [4]. In [4], two process mining strategies on region of languages are presented, having the same minimality goal as the one that we have in this paper. However the strategy is different: integer linear models are solved in order to find a set of special places called *feasible places* that guarantee the inclusion of the traces from the event log. The more places added, the more traces are forbidden in the resulting net. If the net contains all the possible feasible places, then the minimality property can be demonstrated. However, the set of feasible places might be infinite. In our case, given a maximal bound k for the mining of a k-bounded Petri net, minimal regions of the transition system are enough to demonstrate the minimality property on this bound.

Example. In [14], a small log is presented to motivate the *overfitting* produced by synthesis tools. The log contains the following activities: *r=register*, *s=ship*, *sb=send_bill*, *p=payment*, *ac=accounting*, *ap=approved*, *c=close*, *em=express_mail*, *rj=rejected*, and *rs=resolve*. Now assume that the event log contains the traces *(r, s, sb,p, ac, ap, c)*, *(r,sb,em, p, ac, ap, c)*, *(r, sb, p, em, ac, rj, rs, c)*, *(r, em, sb, p, ac, ap, c)*, *(r, sb, s, p, ac, rj, rs, c)*, *(r, sb, p, s, ac, ap, c)* and *(r, sb, p, em, ac, ap, c)*. From this log, a TS can be obtained [13] and a PN as the one shown in Figure 1 will be synthesized by a tool like `petrify` [6]. If the log is slightly changed (for instance, trace *(r, sb, s, p, ac, rj, rs, c)* is replaced by *(r, sb, s, p, ac, ap, c)*, the synthesis tool will adapt the PN to account for the changes, deriving a different PN. This means that synthesis algorithms are very sensitive to variations in the logs. However, the techniques presented in this paper, as it happens also with traditional mining approaches like the α-algorithm [16], are less sensitive to variations in event logs, and will derive the same PN over the modified log.

The two models used in this paper are *Petri nets* and *transition systems*. We will assume that a transition system represents an event log obtained from observing a real system from which an event-based representation (e.g. a Petri net) approximating its behavior must be obtained. The derivation of the transition system from an event log is an important step, that may have big impact in the final mined Petri net, as it is demonstrated in [13]. A two-step approach is presented in [13], emphasizing that the first step (generation of the transition system) is crucial for the balance between underfitting and overfitting. If the desired abstraction is attained in the first step, i.e. the transition system represents an abstraction of the event log, the second step is expected to reproduce exactly this abstraction, via synthesis. The methods presented in this paper extend the possibilities of this two-step approach, given that the second step might also introduce further abstraction in a controlled manner. The approaches based on regions of languages perform the mining process in only one step, provided that logs can be directly interpreted as languages [4].

2 Preliminaries: Theory of Regions

2.1 Finite Transition Systems and Petri Nets

Definition 1 (Transition system). *A* transition system *(*TS*) is a tuple* (S, E, A, s_{in})*, where* S *is a set of* states*,* E *is an alphabet of* actions*, such that* $S \cap E = \emptyset$*,* $A \subseteq S \times E \times S$ *is a set of* (labelled) transitions*, and* s_{in} *is the* initial state*.*

Let $\mathsf{TS} = (S, E, A, s_{in})$ be a transition system. We consider connected TSs that satisfy the following axioms:

- S and E are finite sets.
- Every event has an occurrence: $\forall e \in E : \exists (s, e, s') \in A$;
- Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

A TS is called *deterministic* if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. The relation between TSs will be studied in this paper. The *language* of a TS, $L(\mathsf{TS})$, is the set of traces feasible from the initial state. When, $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$, we will denote TS_2 as an over-approximation of TS_1. The notion of simulation between two TSs is related to this concept:

Definition 2 (Simulation [2]). *Let* $\mathsf{TS}_1 = (S_1, E, A_1, s_{in_1})$ *and* $\mathsf{TS}_2 = (S_2, E, A_2, s_{in_2})$ *be two* TS*s with the same set of events. A* simulation *of* TS_1 *by* TS_2 *is a relation* π *between* S_1 *and* S_2 *such that*

- *for every* $s_1 \in S_1$*, there exists* $s_2 \in S_2$ *such that* $s_1 \pi s_2$*.*
- *for every* $(s_1, e, s_1') \in A_1$ *and for every* $s_2 \in S_2$ *such that* $s_1 \pi s_2$*, there exists* $(s_2, e, s_2') \in A_2$ *such that* $s_1' \pi s_2'$*.*

When TS_1 is simulated by TS_2 with relations π, and viceversa with relation π^{-1}, TS_1 and TS_2 are *bisimilar* [2].

Definition 3 (Petri net [12]). *A Petri net (*PN*) is a tuple* (P, T, F, M_0) *where* P *and* T *represent finite sets of places and transitions, respectively, and* $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation. The initial marking* $M_0 \subseteq P$ *defines the initial state of the system*[1].

The sets of input and output transitions of place p are denoted by $\bullet p$ and $p\bullet$, respectively. The set of all markings of N reachable from the initial marking m_0 is called its Reachability Set. The *Reachability Graph* of PN ($\mathsf{RG}(\mathsf{PN})$) is a transition system in which the set of states is the Reachability Set, the events are the transitions of the net and a transition (m_1, t, m_2) exists if and only if $m_1 \xrightarrow{t} m_2$. We use $L(\mathsf{PN})$ as a shortcut for $L(\mathsf{RG}(\mathsf{PN}))$.

2.2 Regions

We now review the classical theory of regions for the synthesis of Petri nets [8, 7,6]. Let S' be a subset of the states of a TS, $S' \subseteq S$. If $s \notin S'$ and $s' \in S'$, then we say that transition $s \xrightarrow{a} s'$ *enters* S'. If $s \in S'$ and $s' \notin S'$, then transition $s \xrightarrow{a} s'$ *exits* S'. Otherwise, transition $s \xrightarrow{a} s'$ *does not cross* S'.

Definition 4. *Let* $\mathsf{TS} = (S, E, A, s_{in})$ *be a* TS*. Let* $S' \subseteq S$ *be a subset of states and* $e \in E$ *be an event. The following conditions (in the form of predicates) are defined for* S' *and* e*:*

$$\begin{aligned}\texttt{nocross}(e, S') &\equiv \exists (s_1, e, s_2) \in A : s_1 \in S' \Leftrightarrow s_2 \in S' \\ \texttt{enter}(e, S') &\equiv \exists (s_1, e, s_2) \in A : s_1 \notin S' \wedge s_2 \in S' \\ \texttt{exit}(e, S') &\equiv \exists (s_1, e, s_2) \in A : s_1 \in S' \wedge s_2 \notin S'\end{aligned}$$

The notion of a *region* is central for the synthesis of PNs. Intuitively, each region is a set of states that corresponds to a place in the synthesized PN, so that every state in the region models the marking of the place.

Definition 5 (region). *A set of states* $r \subseteq S$ *in* $\mathsf{TS} = (S, E, A, s_{in})$ *is called a* region *if the following two conditions are satisfied for each event* $e \in E$*:*

– *(i)* $\texttt{enter}(e, r) \;\Rightarrow\; \neg\texttt{nocross}(e, r) \;\wedge\; \neg\texttt{exit}(e, r)$
– *(ii)* $\texttt{exit}(e, r) \;\Rightarrow\; \neg\texttt{nocross}(e, r) \;\wedge\; \neg\texttt{enter}(e, r)$

A region is a subset of states in which *all* transitions labeled with the same event e have exactly the same "entry/exit" relation. This relation will become the predecessor/successor relation in the Petri net. The event may always be either an *enter* event for the region (case (i) in the previous definition), or

[1] Although this paper deals with bounded Petri nets, for the sake of clarity we restrict the theory of current and next sections to the simpler class of safe (1-bounded) Petri nets. Section 4 discusses how to generalize the method for bounded Petri nets.

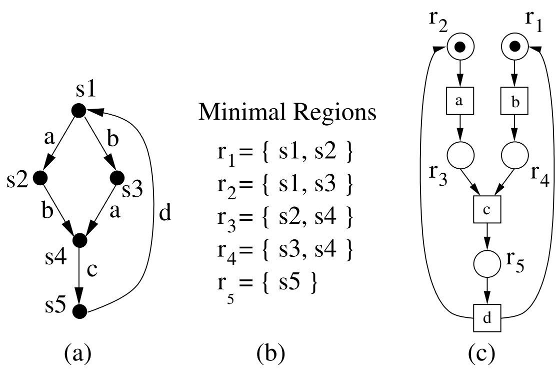


Fig. 2. (a) Transition system, (b) minimal regions, (c) synthesis applying Algorithm of Figure 3

always be an *exit* event (case (ii)), or never "cross" the region's boundaries, i.e. each transition labeled with e is *internal* or *external* to the region, where the antecedents of neither (i) nor (ii) hold. The transition corresponding to the event will be successor, predecessor or unrelated with the corresponding place respectively.

Examples of regions are reported in Figure 2: from the TS of Figure 2(a), some regions are enumerated in Figure 2(b). For instance, for region r_2, event a is an exit event, event d is an entry event while the rest of events do not cross the region.

Definition 6 (Minimal region). *Let r and r' be regions of a TS. A region r' is said to be a* subregion *of r if $r' \subset r$. A region r is a* minimal *region if there is no other region r' which is a subregion of r.*

Going back to the example of Figure 2, in Figure 2(b) we report the set of minimal regions. The union of disjoint regions is a region, so for instance the union of the regions r_1 and r_4 is the set $\{s1, s2, s3, s4\}$ which is also a (non-minimal) region.

Each TS has two *trivial regions*: the set of all states, S, and the empty set. Further on we will always consider only non-trivial regions. The set of non-trivial regions of TS will be denoted by R_{TS}. Given a set $S' \subseteq S$ and a region r, $r\mid_{S'}$ represents the *projection* of the region r into the set S', i.e. $r\mid_{S'} = r \cap S'$.

A region r is a *pre-region* of event e if there is a transition labeled with e which exits r. A region r is a *post-region* of event e if there is a transition labeled with e which enters r. The sets of all pre-regions and post-regions of e are denoted with ${}^\circ e$ and e°, respectively. By definition it follows that if $r \in {}^\circ e$, then all transitions labeled with e exit r. Similarly, if $r \in e^\circ$, then all transitions labeled with e enter r.

Algorithm: PN **synthesis**

- For each event $e \in E$ generate a transition labeled with e in the PN;
- For each minimal region $r_i \in R_{\mathsf{TS}}$ generate a place r_i;
- Place r_i contains a token in the initial marking iff the corresponding region r_i contains the initial state of the TS s_{in};
- The flow relation is as follows: $e \in r_i\bullet$ iff r_i is a pre-region of e and $e \in \bullet r_i$ iff r_i is a post-region of e, i.e.,

$$F_{\mathsf{TS}} \overset{def}{=} \{(r,e) | r \in R_{\mathsf{TS}} \ \wedge \ e \in E \ \wedge \ r \in {}^{\circ}e\} \\ \cup \{(e,r) | r \in R_{\mathsf{TS}} \ \wedge \ e \in E \ \wedge \ r \in e^{\circ}\}$$

Fig. 3. Algorithm for Petri net synthesis from [11]

2.3 Generation of Minimal Regions

The computation of the minimal regions is crucial for the synthesis methods in [6,5]. It is based on the notion of *excitation region* [10].

Definition 7 (Excitation region[2]). *The excitation region of an event e,* $\mathsf{ER}(e)$*, is the set of states in which e is enabled, i.e.*

$$\mathsf{ER}(e) = \{s \mid \exists s' : (s, e, s') \in A\}$$

Minimal regions can be generated from the ERs of the events in a TS in the following way: starting from the ER of each event, set expansion is performed on those events that violate the region condition (a pseudocode of the expansion algorithm is given in Figure 10 from [6]). This exploration can be done efficiently by considering only sets that are not supersets of regions already computed [6], because only minimal regions are required. Each time a region is obtained (according to Definition 5), it is added to the set of regions. Finally, from the set of regions computed, non-minimal regions are removed.

2.4 Region-Based Synthesis

The procedure given by [11] to synthesize a PN, $N_{\mathsf{TS}} = (R_{\mathsf{TS}}, E, F_{\mathsf{TS}}, R_{s_{in}})$, from an *elementary transition system*[3], $\mathsf{TS} = (S, E, A, s_{in})$, is illustrated in Figure 3. Notice that only minimal regions are required in the algorithm [7].

An example of the application of the algorithm is shown in Figure 2. The initial TS and the set of minimal regions is reported in Figures 2(a) and (b), respectively. The synthesized PN is shown in Figure 2(c).

[2] Excitation regions are not regions in the terms of Definition 5. The term is used due to historical reasons.

[3] Elementary transition systems are a proper subclass of the TS considered in this paper, were additional conditions to the ones presented in Section 2.1 are required.

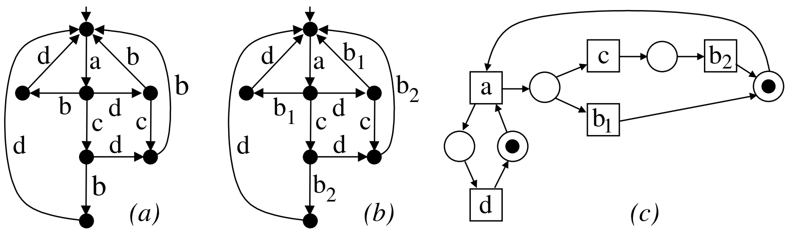


Fig. 4. (a) TS, (b) ECTS by label-splitting, (c) synthesized PN

2.5 Excitation-Closed Transition Systems

Definition 8 (Excitation-closed TS). *A transition system* TS $= (S, E, A, s_{in})$ *is called* excitation-closed *(*ECTS*) if it satisfies the following two axioms:*

- *Excitation closure: For each event a:* $\bigcap_{r \in {}^\circ a} r = \mathsf{ER}(a)$
- *Event effectiveness: For each event a:* ${}^\circ a \neq \emptyset$

The synthesis algorithm in Figure 3 applied to an ECTS derives a Petri net with reachability graph *bisimilar* to the initial TS [6]. Note that the *state separation property* of elementary transition systems, which enforces every pair of states to be distinguished by the set of regions is not required. The set of regions needed by the algorithm to preserve bisimilarity can be constrained to minimal pre-regions.

When the TS is not excitation closed, then it must be transformed to enforce that property. One possible strategy is to represent every event by multiple transitions with the same label. This technique is called *label splitting*. Figure 4 illustrates this technique. The initial TS, shown in Figure 4(a) is transformed by splitting the event b into the events b_1 and b_2, as shown in Figure 4(b), resulting in an ECTS. The synthesized PN, with two transitions for event b is shown in Figure 4(c). Hence in PN synthesis label splitting might be crucial for the existence of a PN with bisimilar behavior. However, sometimes label splitting might degrade the resulting PN structure significantly, deriving intricate causality relations that are not helpful for visualization. This phenomenon is discussed in [5].

The following sections introduce PN mining, a version of PN synthesis where the excitation closure is dropped.

3 Algorithm for Petri Net Mining

The goal of Petri net mining is to generate a PN that over-approximates all observed behaviors in the TS, i.e. $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$, and where $L(\mathsf{PN}) \setminus L(\mathsf{TS})$ is small [4]. Additionally, obtaining a succinct PN with nice causality relations is desirable. For this purpose, the classical synthesis conditions must be adapted to allow the discovery of behaviors not present in the input TS. In this section we show a simple yet powerful approach for relaxing the region-based synthesis conditions from [6,5] to obtain over-approximations of the TS. Formally, given a $\mathsf{TS}=(S, E, A, s_{in})$, the theory of regions can be adapted for mining a Petri net $\mathsf{PN}=(P, T, F, M_0)$ with the following characteristics:

1. $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$,
2. $T = E$, i.e. there is no label splitting, and
3. *Minimal language containment* (`MLC`) property:

$$\forall \mathsf{PN}' = (P', T', F', M_0') \textit{ s.t. } T' = E : L(\mathsf{TS}) \subseteq L(\mathsf{PN}') \Rightarrow L(\mathsf{PN}) \subseteq L(\mathsf{PN}')$$

Therefore the obtained Petri net represents the minimal over-approximation of the input TS that can be synthesized without label splitting. The remainder of this section will show how to relax the region-based synthesis to derive such Petri net.

3.1 Mining Over-approximations of a TS

Bisimilarity or language equivalence are very restricting equivalence relations, not very useful for the area of Petri net mining where over-approximations of the initial event log are more valuable [19,4]. In [6,5], bisimulation between the TS and the synthesized PN holds due to the excitation closure condition. Let us assume in this section that the excitation closure condition is dropped, i.e. the set of minimal pre-regions of some events may properly include the excitation region of the event. With this simple relaxation, the PN obtained by the Algorithm of Figure 3 will satisfy the `MLC` property (Theorem 2).

Theorem 1. *Let* $\mathsf{TS}=(S, E, A, s_{in})$ *be a transition system, and* $\mathsf{PN}=(P, T, F, M_0)$ *be the synthesized net with the set of minimal regions of* TS*, using Algorithm of Figure 3. Then* $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$*.*

Proof. The proof corresponds to the sufficiency direction from Theorem 3.4 in [6]. The theorem guarantees bisimilarity between an ECTS and the reachability graph of the synthesized Petri net from the set of minimal regions. From the two simulations necessary for having bisimulation in that theorem, only one is preserved if the excitation closure condition is dropped. This remaining simulation is the one between the TS and the reachability graph of the PN. Hence every trace in the TS can be simulated by the PN when minimal regions are used, even if the TS is not excitation closed. This suffices to prove the theorem. □

Moreover, as the following results show, regions are preserved under language containment or simulation.

Lemma 1. *Let* $\mathsf{TS}=(S, E, A, s_{in})$*,* $\mathsf{TS}' = (S', E', A', s_{in})$ *be two transition systems such that* $S \subseteq S'$*,* $E \subseteq E'$*,* $T \subseteq T'$*. If* $r \in R_{\mathsf{TS}'}$ *then* $r\,|_S \in R_{\mathsf{TS}}$*.*

Proof. If predicates *(i)*,*(ii)* in Definition 5 hold in r for transitions in A', then they also hold for the transitions in A when r is restricted to S, given that $A \subseteq A'$ and $S \subseteq S'$, i.e. by removing arcs, no new violations of the region conditions can be created (see Definition 5). □

We now prove a similar lemma on the correspondence of regions between simulated TSs.

Lemma 2. *Let* $\mathsf{TS}=(S,E,A,s_{in})$*,* $\mathsf{TS}' = (S',E,A',s'_{in})$ *be such that there exists a simulation relation of* TS *by* TS' *with relation* π*. If* $r \in R_{\mathsf{TS}'}$*, then* $\pi^{-1}(r) \in R_{\mathsf{TS}}$*, and the* `nocross`/`enter`/`exit` *predicates for every event at* r *are preserved in* $\pi^{-1}(r)$*.*

Proof. The proof for this lemma is similar to the one used in Lemma 1, but on simulated states: for every transition $(s,e,s') \in A$ there exists a transition $(\pi(s),e,\pi(s')) \in A'$. Therefore, the predicates *(i)*,*(ii)* in Definition 5 hold in TS for the set $\pi^{-1}(r)$. $\square$

In general, language containment between two TSs does not imply simulation [9]. However, if the TS corresponding to the superset language is deterministic then language containment guarantees the existence of a simulation:

Lemma 3. *Let* $\mathsf{TS}_1 = (S_1,E_1,A_1,s_{in1})$ *and* $\mathsf{TS}_2 = (S_2,E_2,A_2,s_{in2})$ *be two* TS*s such that* TS_2 *is deterministic, and* $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$*. Then* TS_2 *is a simulation of* TS_1*.*

Proof. The relation $\pi \subseteq (S_1 \times S_2)$ defined as follows:

$$s_1 \pi s_2 \Leftrightarrow \exists\ \sigma : s_{in1} \xrightarrow{\sigma} s_1 \wedge s_{in2} \xrightarrow{\sigma} s_2$$

represents a simulation of TS_1 by TS_2: the first item of Definition 2 holds since $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$. If the contrary is assumed, i.e. $\exists s_1 \in S_1 : \not\exists s_2 \in S_2 : s_1 \pi s_2$ then the trace leading to s_1 in TS_1 is not feasible in TS_2, which contradicts the assumption. The second item holds because the first item and the determinism of TS_2: for every $s_1 \in S_1$, TS_2 deterministic implies that there is only one state possible $s_2 \in S_2$ such that $s_1 \pi s_2$. But now if e is enabled in s_1 and not enabled in s_2 will imply that the trace σe, with $s_{in1} \xrightarrow{\sigma} s_1$, is not feasible in TS_2, reaching a contradiction to $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$. $\square$

And now we can proof the `MLC` property on the mined Petri net from a TS:

Theorem 2. *Let* $\mathsf{PN}=(P,T,F,M_0)$ *be the synthesized net with the set of minimal regions of* $\mathsf{TS}=(S,E,A,s_{in})$*, using Algorithm of Figure 3. Then* PN *satisfies the* `MLC` *property.*

Proof. By contradiction. Let $\mathsf{TS}' = (S',E',A',s'_{in})$ be the reachability graph corresponding to a $\mathsf{PN}' = (P',T',F',M'_0)$ such that $E' = T$, $L(\mathsf{TS}) \subseteq L(\mathsf{TS}')$ and $L(\mathsf{PN}) \not\subseteq L(\mathsf{TS}')$. The following facts can be observed:

– TS' and $RG(\mathsf{PN})$ are deterministic because $E = E' = T$ and therefore they correspond to the reachability graph of Petri nets with a different label for each transition.
– Since TS' is deterministic and $L(\mathsf{TS}) \subseteq L(\mathsf{TS}')$, then there is a simulation π of TS by TS' (Lemma 3).

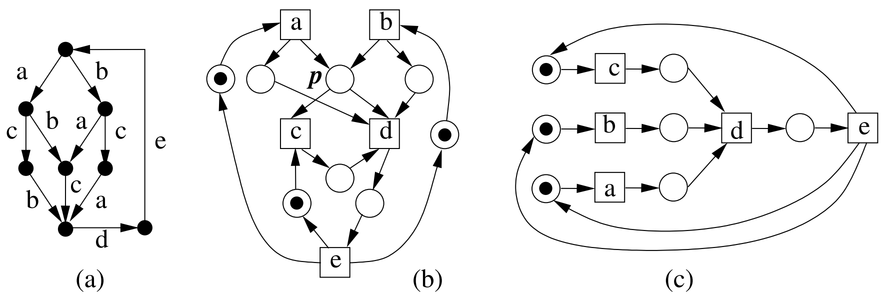


Fig. 5. (a) Initial TS, (b) Mined Petri net , (c) Mined marked graph

– $\forall r' \in R_{\mathsf{TS}'}$, $r = \pi^{-1}(r) \in R_{\mathsf{TS}}$, and the `nocross`/`enter`/`exit` predicates of the events is the same in r' and r (Lemma 2).
– Each non-minimal region can be described as the union of disjoint minimal regions [6], and therefore we can focus only on minimal regions.
– Each minimal region $r \in R_{\mathsf{TS}}$ is a region in $R_{RG(\mathsf{PN})}$, since PN has been obtained with Algorithm of Figure 3 from the set of minimal regions in TS. Moreover, since $RG(\mathsf{PN})$ is deterministic and $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$ (Theorem 1), then there is a simulation of TS by $RG(\mathsf{PN})$ (Lemma 3). Now using Lemma 2, together with the fact that r is a region both in R_{TS} and $R_{RG(\mathsf{PN})}$, the `nocross`/`enter`/`exit` predicates of events in TS hold also in $RG(\mathsf{PN})$.
– Hence, the previous items show that for a region in TS' there is a corresponding region in $RG(\mathsf{PN})$ with the same `nocross`/`enter`/`exit` predicates on the events. In Petri net terms, this fact means that the flow relation of PN' is included in the flow relation of PN. Additionally, the simulations connecting both transition systems preserve the initial states (see Lemma 3). This contradicts the assumption that $L(\mathsf{PN}) \not\subseteq L(\mathsf{TS}')$. □

3.2 Related Issues and Further Extensions

Now we discuss the features of the mining strategy and possible extensions.

Visualization capabilities. By removing the excitation closure condition, one guarantees that there is a 1-to-1 correspondence between the events in the log and the transitions in the Petri net. This is important in terms of visualization. Moreover, the set of minimal regions can include *redundant* regions: a region r is redundant if the language of the Petri net without r is preserved. Therefore redundant regions can be safely removed from the net. The theory in [6, 5] proposes methods to detect redundant places, based in the preservation of the excitation closure. Those methods have been adapted and included in the mining approach presented in this paper.

Mining of Petri net subclasses. As it has been done in synthesis (see [6], Section 4.4), the approach presented in this paper might be adapted to mine

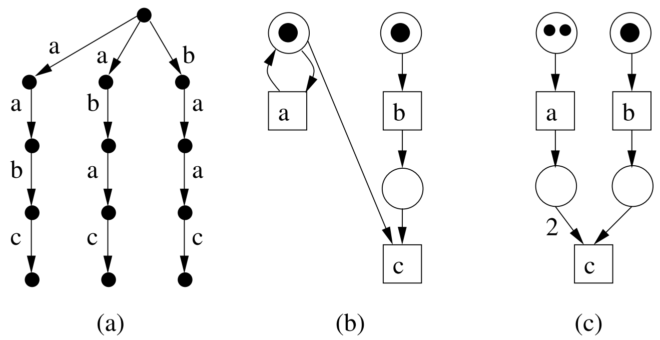


Fig. 6. (a) Transition system, (b) Mined safe Petri net, (c) Mined 3-bounded Petri net

Petri net subclasses. The basic idea is to restrict the generation of regions in order to generate regions satisfying structural Petri net conditions. Let us use the example in Figure 5 to illustrate this. In Figure 5(b) we report the mined two-bounded Petri net from the TS of Figure 5(a) (next Section shows how to generalize the mining method to the bounded case). Now imagine that we are interested in the mining of *marked graphs*, i.e. Petri nets where places have at most one predecessor (successor) transition. Notice that place p in Figure 5(b) does not satisfy this condition. If the mining of a marked graph is applied, the Petri net shown in Figure 5(c) is obtained.

Critical events. The methods presented can be extended to select those events that might be *critical* in terms of representation: for those events, the avoidance of over-approximation might be imposed by requiring excitation closure on them. The application of label-splitting can be guided to attain this goal.

4 Mining Bounded Petri Nets

In the literature for the mining of Petri nets from event logs, it is widely accepted the use of ordinary and safe Petri nets for the discovery of process models (a remarkable exception is presented in [4]). Due to the recent results for the synthesis of bounded and weighted Petri nets [5], this limitation can be waved, and therefore a more succinct and accurate model of the log can be obtained using the techniques developed in this paper. Moreover, the possibility to search for unsafe regions might be crucial in order not to over-approximate the event log too much. To illustrate this fact, see the example in Figure 6. The mining of a safe Petri net from the TS of Figure 6(a) is shown in Figure 6(b), whereas Figure 6(c) reports the mining of a 3-bounded net. The language accepted by the PN from Figure 6(b) is $(a* \parallel b)c$ which might be an over-conservative approximation[4], while the net in Figure 6(c) accepts $(a^3 \parallel b)c$, which although being also an over-approximation, it is a more accurate one. This section introduces

[4] The expression $e_1 \parallel e_2$ denotes the set of possible interleavings between e_1 and e_2.

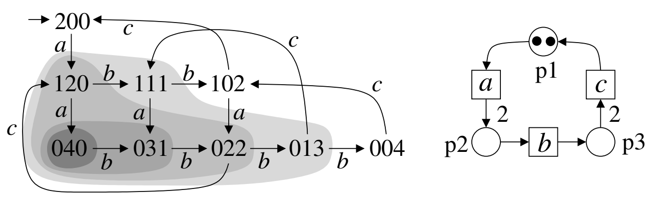


Fig. 7. A transition system and an equivalent bounded Petri net

informally how the theory of the previous sections can be generalized to mine bounded systems.

In [5], an extension of the region-based synthesis of Petri nets has been presented to support bounded nets. The methods assume that a k is initially given for the search of a k-bounded Petri net. Let us use the example in Figure 7 to summarize the theory. In the bounded case, the basic idea is that regions are represented by multisets (i.e., a state might have multiplicity greater than one). Figure 7 depicts a TS with 9 states and 3 events. After synthesis, the Petri net at the right is obtained. Each state has a 3-digit label that corresponds to the marking of places p_1, p_2 and p_3 of the Petri net, respectively. The shadowed states represent the general region that characterizes place p_2. Each grey tone represents a different multiplicity of the state (4 for the darkest and 1 for the lightest). Each event has a gradient with respect to the region (+2 for a, -1 for b and 0 for c). The gradient indicates how the event changes the multiplicity of the state after firing. For the same example, the equivalent *safe* Petri net has 5 places and 10 transitions.

In summary, the generalization of the theory of Sections 2 and 3 is based on the idea that regions are no longer sets but multisets, and the predicates for region conditions must take into account the gradient of each event on the multisets. The excitation closure notion is defined on the support (states with multiplicity greater or equal than one) of the multiset. Finally, the algorithm for synthesis of bounded Petri nets is generalized to account for bounded markings and weighted arcs. The interested reader may refer to [5] for details.

The theory in [5] has been incorporated in the Petri net mining approach presented in this paper. Hence the mining of Petri nets can be guided to find bounded Petri nets. An example of k-bounded mining is shown in Section 5.

5 Examples, Experiments and Tool

The theory described in this paper has been incorporated in `Genet`, a tool for the synthesis and mining of concurrent systems [5]. Most of the examples have been obtained from [1].

Mining of safe Petri nets

Some examples have been presented along the paper. An additional example is shown in Figure 8. The language accepted by the PN of Figure 8(b) is

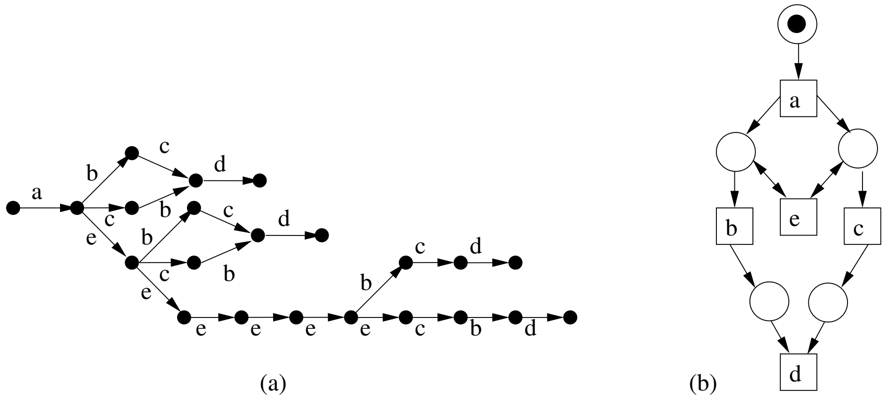


Fig. 8. (a) Initial TS, (b) Mined Petri net

$ae*(b \parallel c)d$, which properly includes the language of the TS of Figure 8(a). Remarkably, applying the α-algorithm [17] to this event log results in the same PN.

Mining of bounded Petri nets

An example of the power of k-bounded PN mining is shown in Figure 9. The system modeled represents 5 processess sharing 3 resources. Every process requires one resource, but there is one process that requires two resources. We assume that the TS used for this example can be constructed from a set of simulations. The 3-bounded PN from the corresponding transition system contains 20 states and 74 arcs. The synthesis of a safe PN from the transition system applies many label splittings in order to enforce the excitation closure, deriving in a PN with 15 places, 34 transitions and 128 arcs. Clearly, neither the initial TS nor the synthesized PN are of any help to realize the control flow of this example. However, the mined 3-bounded PN is a succint representation of the log.

Experiments

The mining of some examples is summarized in Table 1. Following the two-step mining approach from [13], we have obtained the transition systems from each log with the *FSM Miner* plugin available in ProM. For each log, columns report the number of states of the initial log $|S|$, number of states of the minimal bisimilar transition system $|[S]|$ (that gives an idea of the amount of redundancy present in the initial log) and number of events $|E|$. Next, the number of places $|P|$ and transitions $|T|$ of the PN obtained by synthesis is reported. For each version of the mining algorithm (safe and 2-bounded), the number of places of the mined PN and number of states of the corresponding minimal bisimilar reachability graph are reported. The CPU time for the mining of all examples but the last one has taken less than two seconds. The mining of pn_ex_10, 2-bounded version, took one minute. Finally the same information is provided for two well-known mining algorithms in ProM: the *Parikh Language-based Region* and the *Heuristics* [20] miners. The number of unconnected transitions ($|T_U|$)

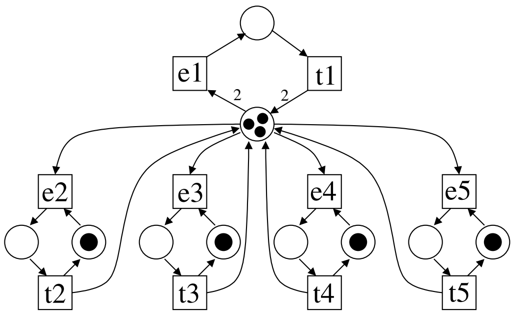


Fig. 9. Mined 3-bounded PN for a system of five processes sharing three resources

Table 1. PN mining applied to event logs from [1]

				synthesis		mining									
				`petrify` safe		`Genet` safe		`Genet` 2-bounded		ProM Parikh			ProM Heuristics		
benchmark	$\|S\|$	$\|[S]\|$	$\|E\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|[S]\|$	$\|P\|$	$\|[S]\|$	$\|P\|$	$\|T_U\|$	$\|[S]\|$	$\|P\|$	$\|T_I\|$	$\|[S]\|$
groupedFollowsa7	18	10	7	7	7	6	11	7	11	7	0	10	7	1	8
groupedFollowsal1	15	15	7	8	9	10	16	12	15	7	0	7	14	10	22
groupedFollowsal2	25	25	11	15	11	15	25	15	25	11	0	13	15	3	25
herbstFig6p21	16	16	7	11	13	7	22	11	16	1	6	2	18	15	∞
herbstFig6p34	32	32	12	16	13	16	34	18	32	8	2	12	19	12	∞
herbstFig6p41	20	18	14	16	14	16	18	16	18	17	0	18	14	0	18
staffware_15	31	24	19	20	20	18	22	19	31	18	0	21	19	0	19
pn_ex_10	233	210	11	64	218	13	281	16	145	8	2	14	41	25	∞

derived by the Parikh miner and the number of invisible transitions introduced by the Heuristic miner is also reported ($|T_I|$).

The numbers in Table 1 suggest some remarks. If the synthesis is compared with the mining in the case of safe PNs, it should be noticed that even for those small examples the number of transitions is reduced, due to the absence of label splitting (see row for pn_ex_10). The number of places is also reduced in general. It should also be noticed that 2-bounded mining represents the log more accurately, and thus more places are needed with respect to the mining of safe nets. Sometimes the mined PN accepts more traces but the corresponding minimal bisimilar transition system has less states, e.g. pn_ex_10: after over-approximating the initial TS, several states become bisimilar and can be minimized.

The Parikh miner tends to derive very aggressive abstractions, as it is demonstrated in the pn_ex_10 and herbstFig6p21 logs. Sometimes the Petri nets obtained with this miner contain isolated transitions, because the miner could not find places connecting them to the net. The Heuristics miner is based on the frequency of patterns in the log. The miner derives a heuristic net that can be afterwards converted to Petri net with ProM. Some of the Petri nets obtained with this conversion turned out to be unbounded (denoted with symbol ∞ in the table), and contain a significant amount of invisible transitions. This miner is however very robust to noise in the log. In conclusion, different miners can achieve different mining goals, widening the application of Process mining into several directions.

6 Conclusions

A strategy for adapting the theory of regions for the area of Process mining has been presented. The main contribution is to allow the generation of over-approximations of the event log by means of a bounded Petri net, not necessarily safe. An important result is presented that guarantees the minimal language containment property on the mined PN. The theory has been incorporated in a synthesis tool.

Acknowledgements

We would like to thank W. van der Aalst, E. Verbeek and B. van Dongen for helpful discussions and guidance, and anonymous referees for their help in improving the final version of the paper. This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

1. Process mining, www.processmining.org
2. Arnold, A.: Finite Transition Systems. Prentice-Hall, Englewood Cliffs (1994)
3. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–383. Springer, Heidelberg (1995)
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Proc. 5th Int. Conf. on Business Process Management, September 2007, pp. 375–383 (2007)
5. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. In: 29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (June 2008)
6. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Transactions on Computers 47(8), 859–882 (1998)
7. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. 33(4), 297–315 (1996)
8. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica 27, 315–368 (1990)
9. Engelfriet, J.: Determinacy - (observation equivalence = trace equivalence). Theor. Comput. Sci. 36, 21–25 (1985)

10. Kishinevsky, M., Kondratyev, A., Taubin, A., Varshavsky, V.: Concurrent Hardware: The Theory and Practice of Self-Timed Design. John Wiley and Sons, London (1993)
11. Nielsen, M., Rozenberg, G., Thiagarajan, P.: Elementary transition systems. Theoretical Computer Science 96, 3–33 (1992)
12. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Bonn, Institut für Instrumentelle Mathematik (technical report Schriften des IIM Nr. 3) (1962)
13. van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Günther, C.: Process mining: A two-step approach to balance between underfitting and overfitting. Technical Report BPM-08-01, BPM Center (2008)
14. van der Aalst, W.M.P., Günther, C.W.: Finding structure in unstructured processes: The case for process mining. In: Basten, T., Juhás, G., Shukla, S.K. (eds.) ACSD, pp. 3–12. IEEE Computer Society, Los Alamitos (2007)
15. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W.E., Weijters, A.J.M.M.: ProM 4.0: Comprehensive support for real process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
16. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
17. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
18. van Dongen, B., Busi, N., Pinna, G., van der Aalst, W.: An iterative algorithm for applying the theory of regions in process mining. Technical Report Beta rapport 195, Department of Technology Management, Eindhoven University of Technology (2007)
19. Verbeek, H., Pretorius, A., van der Aalst, W.M.P., van Wijk, J.J.: On Petri-net synthesis and attribute-based visualization. In: Proc. Workshop on Petri Nets and Software Engineering (PNSE 2007), June 2007, pp. 127–141 (2007)
20. Weijters, A., van der Aalst, W., de Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology (2006)

BESERIAL: Behavioural Service Interface Analyser

Ali Aït-Bachir[1,⋆], Marlon Dumas[2], and Marie-Christine Fauvet[1]

[1] University of Grenoble, LIG (MRIM)
{Ali.Ait-Bachir,Marie-Christine.Fauvet}@imag.fr
[2] University of Tartu Estonia
marlon.dumas@ut.ee

Abstract. In a service-oriented architecture, software services interact by means of message exchanges that follow certain patterns documented in the form of behavioural interfaces. As any software artifact, a service interface evolves over time. When this happens, incompatibility problems may arise. We demonstrate a tool, namely *BESERIAL*, that can pinpoint incompatibilities between behavioural interfaces.

1 Motivation

The interface of a software service establishes a contract between the service and its clients or peers. In its basic form, a service interface defines the operations provided by the service and the schema of the messages that the service can receive and send. This *structural interface* can be captured for example using WSDL. In the case of *conversational services* that provide several inter-related operations, a service interface may also capture the inter-dependencies between these operations. Such *behavioural interfaces* can be described for example using BPEL business protocols, or more simply using state machines as we consider in this paper [1].

As a service evolves, its interface is likely to undergo changes. These changes may lead to the situation where the interface provided by a service no longer matches the interfaces that its peers expect from it. This may result in incompatibilities between the service and the client applications and other services that interact with it.

This paper presents a tool, namely BESERIAL, which is able to automatically detect incompatibilities between behavioural service interfaces and to report them graphically. This feature enables designers to pinpoint the exact locations of these incompatibilities and to fix them. BESERIAL is able to detect elementary changes in the flow of service operations that lead to incompatibilities (e.g. adding an operation, deleting an operation and modifying an operation). Existing tools, such

⋆ This author is partially funded by the Web Intelligence project granted by the French Rhône-Alpes Region and a scholarship from Estonian Ministry of Education and Research.

as WS-Engineer[1], are able to detect if two behavioural interfaces are compatible. But, they will only detect one incompatibility at a time, whereas BESERIAL identifies several incompatibilities at once. Also BESERIAL allows one to compare a given interface with multiple other interfaces in order to identify which one most closely matches the given interface. This can be used for service selection. A screencast of the demo is available online at `http://www-clips.imag.fr/mrim/User/ali.ait-bachir/webServices/webServices.html`

2 BESERIAL Tool

In BESERIAL behavioural service interfaces are modeled by means of Finite State Machines (FSM)[2]. These FSMs are serialized in SCXML[3]. Figure 1 (*Process1*) depicts an FSM representing the behaviour of a given interface (the underlying service supports orders and deliveries of goods). In this work, we consider only the external behaviour of an interface (sent messages and received messages) and we abstract away from internal steps of the underlying business process. Accordingly, transitions are labelled by the types of messages to be sent (with prefix '!') or received (with prefix '?').

2.1 Incompatibility Detection

One of the main features of BESERIAL is to detect changes between a new version of an interface and a previous one. BESERIAL specifically detects changes that may cause the new version to be incompatible vis-a-vis of clients or peers that use the previous version. To that end, BESERIAL relies on a simulation algorithm incorporating an incompatibility diagnosis and recovery mechanism. The originality of BESERIAL is that the simulation algorithm does not stop at the first incompatibility encountered [1,3] but tries to search further to identify a series of incompatibilities leading up to one of the final states of the old version of the interface.

A screenshot of the tool is shown in Figure 1. The screenshot displays the result of comparing two versions of an interface FSM (*Process2* versus *Process1*). The operation that allows customers to cancel an order has been deleted as well as the operation that allows the supplier to send updated information about an order to the customer (*!OrderResponse*). Accordingly, we can see two state pairs ($updateOrderResonse, \_invoice$) and ($transfer, \_transfer$) linked by a dashed edge labelled *deletion*. The deleted operations are *!OrderResponse* and *?CancelOrder* shown by dotted arrows.

For validation purposes, we built a test collection consisting of 14 process scenarios from the xCBL[4] textual description of order management choreographies.

[1] `http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/`

[2] Transformations exist between other languages for describing behavioural service interfaces (e.g. BPEL) and FSMs – see for example the WS-Engineer and Tools4BPEL toolsets referenced above.

[3] `http://www.w3.org/TR/scxml/`

[4] XML Common Business Library (`http://www.xcbl.org/`).

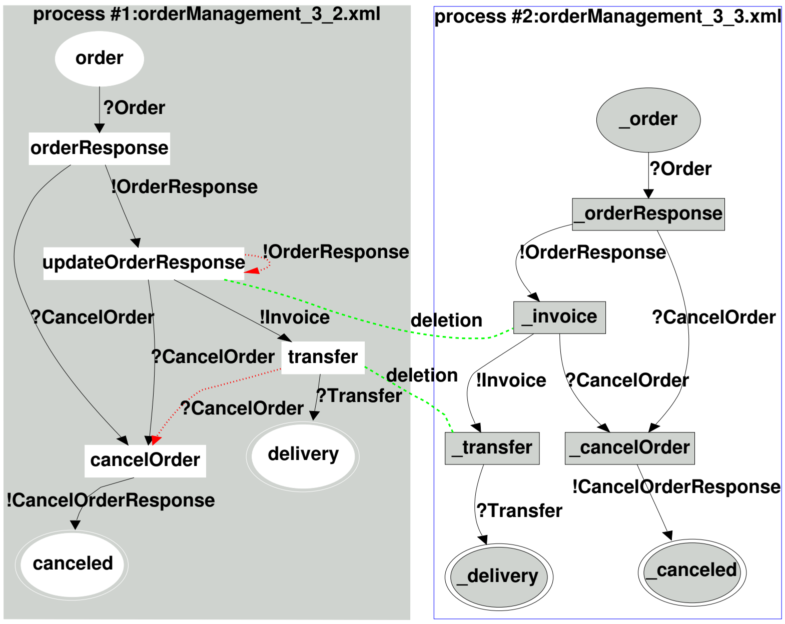


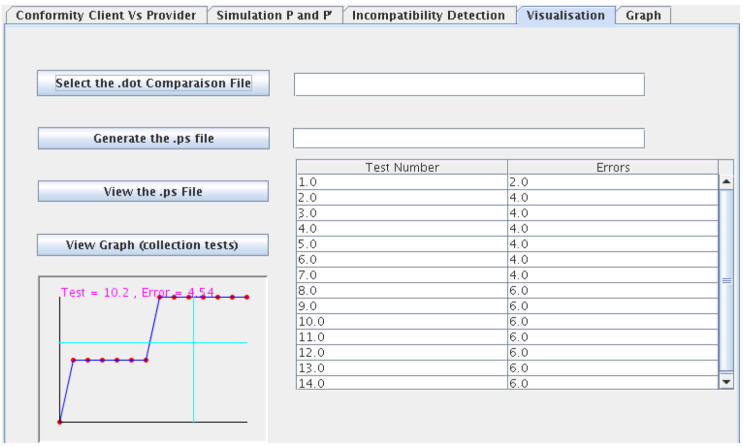
Fig. 1. Detected incompatibilities in two FSMs

These two-party choreographies describe possible document exchanges between trading partners in an Order Management business process.

2.2 BESERIAL in Action

One of the two features of BESERIAL is to detect changes between two behavioural interfaces that cause that one interface does not simulate the behaviour of another interface. A typical usage scenario is one where the compared interfaces correspond to consecutive versions of a service. The algorithm simulates the two FSMs by visiting state pairs (one state from each of the two interfaces). Given a state pair, the algorithm determines if an incompatibility exists and classifies it as addition, deletion or modification (i.e. replacement of one operation with another). If an *addition* is detected the algorithm moves along the transition of the added operation in the new version only. Conversely, if the change is a *deletion*, the algorithm will move along the transition of the deleted operation in the old version only. However, if a *modification* is detected, the algorithm progresses along both FSMs simultaneously. Detection results are written down in a text area showing the test and its outcomes (states, changes). Results can be viewed graphically, as in Figure 1, to better pinpoint the incompatibilities.

BESERIAL can also compare one interface to a collection of interfaces. The comparison results are sorted increasingly according to the number of detected incompatibilities. This functionality may be used to select which service interface is most closely compatible with the required interface.


Conformity Client Vs Provider
Simulation P and P'
Incompatibility Detection
Visualisation
Graph
Select the .dot Comparaison File
Generate the .ps file
View the .ps File
View Graph (collection tests)
Test = 10.2 , Error = 4.54

Test Number	Errors
1.0	2.0
2.0	4.0
3.0	4.0
4.0	4.0
5.0	4.0
6.0	4.0
7.0	4.0
8.0	6.0
9.0	6.0
10.0	6.0
11.0	6.0
12.0	6.0
13.0	6.0
14.0	6.0

Fig. 2. Test results of incompatibility detections in BESERIAL tool

In Figure 2, one given interface is compared to other interfaces and the detection result is rendered and sorted in the table area. Results can be sorted increasingly and a graph can be viewed. The graph shows which interface yields less incompatibilities with respect to the interface given as reference. In this example, the closest interface to the given one yields two incompatibilities and the worst result is six incompatibilities.

3 Future Work

In this paper we focused on elementary incompatibility detection. Ongoing work aims at extending BESERIAL towards two directions:

– Detecting complex incompatibilities combining elementary ones,
– Fixing detected incompatibilities.

As BESERIAL covers synchronous communications only, it also needs to be extended to address the asynchronous case along the lines of [2].

References

1. Bordeaux, L., Salan, G., Berardi, D., Mecella, M.: When are two web services compatible? In: Proc. 5th Int. on Technologies for E-Services, Canada, pp. 15–28. Springer, Heidelberg (2004)
2. Motahari-Nezhad, R.H., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: Proc. of the 16th WWW Int. Conf., Canada, pp. 993–1002. ACM, New York (2007)
3. Wu, J., Wu, Z.: Similarity-based web service matchmaking. In: Proc. of the Int. Conference on Services Computing, Florida, pp. 287–294 (2005)

Business Transformation Workbench: A Practitioner's Tool for Business Transformation

Juhnyoung Lee, Rama Akkiraju, Chun Hua Tian, Shun Jiang,
Sivaprashanth Danturthy, and Ponn Sundhararajan

IBM Corporation
Armonk, New York 10504, U.S.A.
{jyl,akkiraju}@us.ibm.com, {chtian,jshun}@cn.ibm.com,
{Sivaprashanth.D,psundhar}@in.ibm.com

Abstract. Business transformation is a key management initiative that attempts to align people, business process and technology of an enterprise more closely with its business strategy and vision. It is an essential part of the competitive business cycle. Existing consulting methods and tools do not address issues such as scalability of methodology, knowledge management, asset reuse, and governance well, to name a few. This paper presents Business Transformation Workbench, a practitioner's tool for business transformation addressing these problems. It implements a methodical approach that was devised to analyze business transformation opportunities and make business cases for transformation initiatives and thereby offering decision-support to the consultants. It provides an intuitive way to evaluate and understand various opportunities in staff and IT consolidation and process standardization. It embodies structured analytical models, both qualitative and quantitative, to enhance the consultants' practices. BT Workbench has been instantiated with data from finance management domain and applied to address a client situation as a case study. An alpha testing of the tool was conducted with about dozen practitioners. 90% of the consultants who tested the BT Workbench tool felt that the tool would help them do a better job during a client engagement.

1 Introduction

Business transformation is a key management initiative that attempts to align people, business process and technology of an enterprise closely with its business strategy and vision. Business transformation is often achieved by taking a holistic look at various dimensions of an enterprise such as business models, management practices, business processes, organizational structure and technology and optimizing them with best-practice or differentiated methods to reach a strategic end state. For example, business transformation in the enterprise finance area would, among others, optimize financial processes such as accounts receivables, eliminate non-value-added tasks, improve efficiency and productivity of people, and reduce errors by using technologies. Business transformation is considered an essential part of the competitive business cycle.

Consulting service companies in the business transformation area brand technology and consulting as their core product and service offerings. These offerings include models, methods and tools devised for facilitating business transformation. While the state-of-the-art business transformation consulting models and methods are useful, there are a number of general problems that need to be addressed to make them more effective. First, the current approaches are often limited in scalability because they demand subject matter experts to work with a variety of disconnected data, tools, templates and other assets. It is often cumbersome and difficult to streamline the data gathering and management manually. Data and documents often reside in multiple folders distributed among several machines. Consistency checking across data can only be done manually, and the process requires experts. Second, it is hard to capture a structured thinking process without a tool which enforces the process or method. Third, it is difficult to disseminate and reuse knowledge effectively, if it is not captured systematically. In addition, assets such as knowledge, models and methods are not necessarily managed. For example, more often than not, there is no version control in place, and updating the assets is hard to do consistently across the board. Finally, it is difficult to visualize multiple views with scattered documents of a process view, a metrics view, a component view, a resource view, etc., which, in turn, makes it hard to link up upstream and downstream analyses.

This paper presents a practitioner's tool for business transformation addressing these problems. Business Transformation Workbench is a productivity tool for business consultants. It is a tool to analyze business performance, to identify transformation opportunities and to assess the business value of specific transformation initiatives. Using this tool, consultants can examine which business functions and components are underperforming in comparison to industry benchmark measures and why. By investigating the organizational responsibilities and IT application portfolio in conjunction with business components, shortfalls such as duplications, over-extensions, gaps and deficiencies can be identified and reasoned. Specific solutions can be discovered to address the identified shortfalls. Financial benefits of implementing specific solutions can be analyzed further via conducting a business case analysis. The Business Transformation Workbench embodies best practices and methodologies in a tool, which also helps address scalability, data management, and governance, linkages to upstream and downstream activities, analyses around benchmarking, Component Business Model (CBM) based analysis, and business case preparation.

2 Business Transformation Workbench

BT Workbench provides an integrated view of various business models and data, including component business models, a business process model such as APQC (American Product Quality Council) Process Classification Framework (PCF) and SAP Business Process Hierarchy (BPH), a value driver model, an IT infrastructure map, an organization structure map, and a solution catalog, with the models linked to each other. It automates traditional component business model-based analyses in form of visual queries and inference.

For example, one can ask questions such as which metrics help measure the performance of a given business component? What are the IT systems that support the business functions represented by a business component? Which organizations implement the business functions represented by a business component? Which transformation solutions can address a given shortfall? These questions are answered in the tool via the explicit and the inferred linkages made among different models such as the component model, IT system model, organizational model, metrics model etc. This is also referred to as daisy chain analysis in the tool. The tool automates the component performance analysis by comparing the metrics that help measure the performance of a component with benchmark data. This is referred to as ‘heat map’ analysis in the tool. The underperforming components can be marked as shortfalls based on whether it is caused by a misaligned IT system or by an organization. This identification and marking of shortfalls is referred to as ‘shortfall assessment’ in the tool. Finally, the tool provides business benefit analyses in terms of value drivers and standard financial metrics for business case analysis such as NPV (Net Present Value), IRR (Internal Return Rate), ROI (Return on Investment), and payback time. BT Workbench provides normative and constructive business performance analysis models, so it can be easily configured for different types of clients, initiatives, and projects.

3 Component Business Modeling

Component Business Modeling is a novel business modeling technique from IBM. A component business model represents the entire business in a simple framework that fits on a single page. It is an evolution of traditional views of a business, such as business unit view, functional view, geographical view, and processes view. The component business model methodology helps identify basic building blocks of business, where each building block includes the people, processes and technology needed by this component to act as a standalone entity and deliver value to the organization.

After a comprehensive analysis of the composition of each business, a consultant can map these individual building blocks, or components, onto a single page. Each component business map is unique to each company. The columns are created after thorough analysis of a business’s functions and value chain. The rows are defined by actions. Figure 1 shows a visual representation of a CBM map for a business. The top row, ‘direct,’ shows all of those components in the business that set the overall strategy and direction for the organization. The middle row, ‘control,’ represents all of the components in the enterprise which translate those plans into actions, in addition to managing the day-to-day running of those activities. The bottom row, ‘execute,’ contains the business components that actually execute the detailed activities and plans of an organization. The component business map shows activities across lines of business, without the constrictions of geographies, internal silos or business units. This single page perspective provides a view of the business which is not constricted by barriers that could potentially hamper the ability to make meaningful business transformation. The component business model facilitates to identify which components of the business create differentiation and value. It also helps identify where the business has capability gaps that need to be addressed, as well as opportunities to improve efficiency and lower costs across the entire enterprise.

The Business Component Performance Analysis is an essential capability of CBM where the user discovers one or more ‘hot’ components that are associated with one or more business strategies and/or pain points. In the traditional CBM analysis, this step was conducted manually by the business consultants relying on his/her knowledge and expertise in the business domain. The BT Workbench automates the capability as visual queries, by taking metrics values into account with the analysis. First, the system allows the user to explore the value driver tree to identify one or more value drivers that may be associated with a certain business strategy/pain point. The discovery of 'hot' components that affect the business strategy can be accomplished. Then the system colors the identified hot components differently to distinguish ones that affect positively or negatively to the strategy. The BT Workbench system compares the industry benchmark and the as-is value of the operational metrics and performance indicators associated with the components to decide on their color.

4 Concluding Remarks

In this paper we presented Business Transformation Workbench, a consulting practitioner's tool for identifying and analyzing business transformation opportunities. It embodies structured analytical models (both qualitative and quantitative) to enhance the consultants' practices. The tool helps visualize the linkages of various enterprise models such as the business component model, the business process model, the value driver model, the organization model, the IT application model, and the solution model. Using this tool, consultants can examine which business functions and components are underperforming in comparison to industry benchmark measures and why. By investigating the organizational responsibilities and IT application portfolio in conjunction with business components, shortfalls such as duplications, over-extensions, gaps and deficiencies can be identified and reasoned. Specific solutions can be discovered to address the identified shortfalls. Financial benefits of implementing specific solutions can be analyzed further via conducting a business case analysis. BT Workbench has been instantiated with data from finance management domain and applied to address a client situation as a case study. An alpha testing of the tool was conducted with about dozen practitioners. The feedback has been very encouraging. 90% of the consultants who tested the BT Workbench tool felt that the tool would help them do a better job during a client engagement. The tool is currently being piloted with customer engagements in a large IT consulting organization.

The BT Workbench methodology and its software solution is part of an ongoing research initiative on business design and transformation at IBM Research and Global Business Service Divisions. With a methodology and a research prototype in place, we work with practitioners to validate them with real-world business transformation initiatives. In addition to the tool and methodology, in practice, the availability of useful and accurate content and information of business components, value drivers, processes and solutions is critical to meaningful analyses.

Oryx – An Open Modeling Platform for the BPM Community

Gero Decker, Hagen Overdick, and Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
{gero.decker,hagen.overdick,weske}@hpi.uni-potsdam.de

1 Introduction

In the academic business process management community, tooling plays an increasingly important role [8,6]. There are good reasons for this fact. Firstly, theoretical concepts benefit from exploration using prototypical implementation of the concepts. By experimentation based on real-world business processes, concepts can be evaluated and refined. Secondly, the practical applicability of the research work can be demonstrated, which is important to raise awareness of academic BPM research to practitioners.

In academic research groups, researchers tend to implement small-scale prototypes that can do exactly what the particular researcher is interested in. Typically each project is started from scratch. If results from collaborators are re-used, then re-use is done in a non-structured way, by copying and pasting program code. As a result, the wheel is re-invented many times, and valuable resources are wasted. Motivated by this observation, the business process technology research group at HPI has decided to develop an open and extensible framework for business process management, called Oryx (`http://oryx-editor.org`).

Oryx supports web based modeling of business processes using standard Firefox web browsers, so that no additional software installation at the client side is required. Users log on to the Oryx web site and authenticate by openID, an internet authentication standard. They start modeling processes, share them with collaborators, or make them available to the public.

More technically, in Oryx each model artefact is identified by a URL, so that models can be shared by passing references, rather than by exchanging model documents in email attachments. Since models are created using a browser and models are just "a bookmark away", contribution and sharing of process models is eased. Using a plugin mechanism and stencil technology, Oryx is extensible. Today there are stencil sets for different process modeling languages, including BPMN [1], EPC [3], Petri nets [4], and Workflow nets [7]. But the extensibility is not restricted to process languages. The plugin mechanism also facilitates the development of new functionality, for instance mappings to executable languages, thereby providing a business process management framework.

The rest of the paper is structured as follows. Section 2 highlights the most important requirements for the use case scenarios addressed. Section 3 outlines how these requirements are addressed in the Oryx framework by discussing its architecture. Finally, a conclusion is given in Section 4.

2 Oryx Use Cases

Most importantly, there must be editing functionality for graphical business process models. Different modeling languages are present in the BPM field. Probably most prominently, there are the Business Process Modeling Notation (BPMN [1]) and event-driven process chains (EPC [3]). Here, the corresponding stencil sets must be available and certain restrictions on the models that can be created must be enforced. E.g. it must not be possible to connect two events in an EPC or to have incoming sequence flow into a start event in BPMN.

Once process models have been created, it must be verified if the models are free of modeling errors, e.g. using one of the "soundness" notions. Such analysis requires that the diagrams are not just interpreted as mere collection of nodes and edges. Elements must be properly connected, for instance BPMN tasks with preceding or succeeding tasks, or tasks with their parent subprocess or pool. BPMN comes with a special challenge, namely attached intermediate events, where a node is directly connected to another node without edges in between.

Process models are subject to transformations. E.g. BPMN models must be transformed to Petri nets in order to carry out analysis. In other scenarios, high-level models serve as input for generating stubs for more technical models. As an example, BPMN models can be transformed to Business Process Execution Language (BPEL) processes. In order to ease integration with other systems, common interchange formats must be supported.

3 Oryx Overview

Figure 1 depicts the Oryx architecture. While the current release requires a specific Oryx backend, in theory any location on the Web will do. Oryx itself is a set of Javascript routines loaded into a web browser as part of a single document describing the whole model. Models are represented in RDF format.

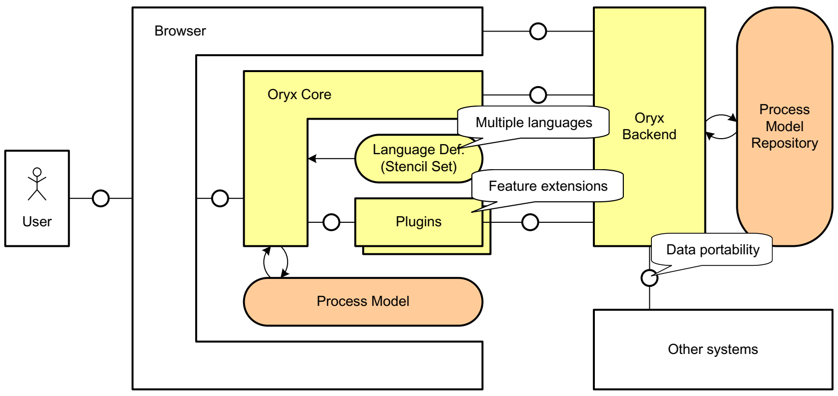


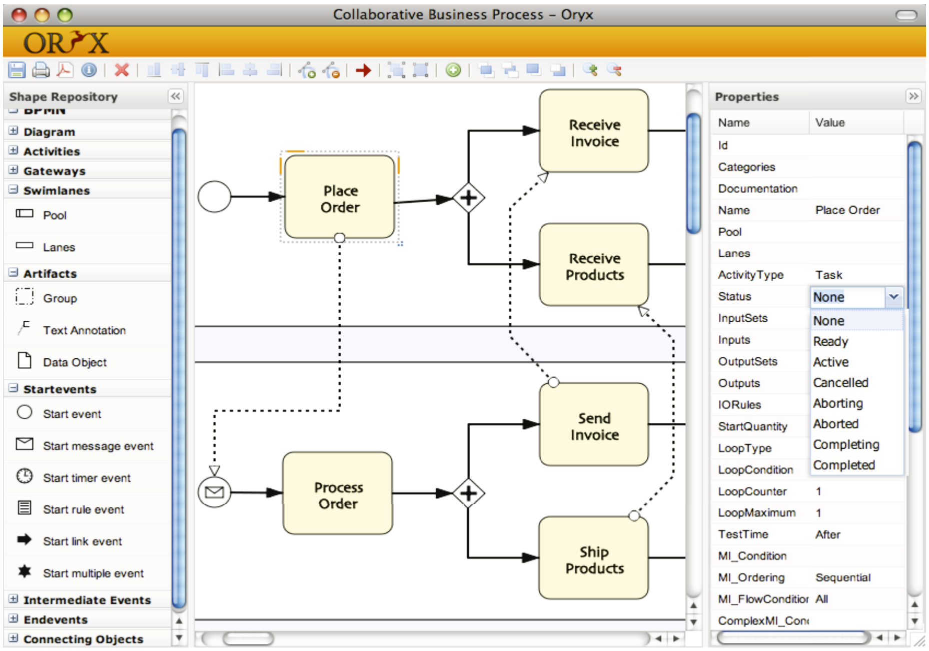
Fig. 1. Oryx architecture

The Oryx core provides generic handling of nodes and edges; how to create, read, and update them; as well as an infrastructure for stencil sets and plugins.

Language Support via Stencil Sets. Stencil sets drive the Oryx core, as they provide explicit typing, connection rules, visual appearance, and other features differentiating a model editor from a generic drawing tool. Hence, a stored Oryx model is structure first, directly based on the loaded stencil sets, visual diagram second. Oryx today has full support for BPMN 1.0 and 1.1. In addition, there is a stencil set for EPC and Petri nets.

Feature Extensions via Plugins. Plugins allow for both generic as well as notation-specific extensions. E.g. element selection and cut & paste are plugin features, as they are not needed for an Oryx viewer. More advanced plugins allow for complex model checking beyond the powers of the stencil set language. For instance, a BPMN to Petri net mapping is included as specified in [2] as well as BPMN to BPEL transformation [5], XPDL serialization for BPMN and soundness checking.

Data Portability beyond Oryx. The Oryx core, with the help of stencil sets and plugins, allows users to create, edit, and view visual models within a browser. Currently, Oryx does so by self-modifying the loaded page and sending it back to the server in whole. Being web-based Oryx reduces deployment and collaboration to distributing a single bookmark.



Name	Value
Id	
Categories	
Documentation	
Name	Place Order
Pool	
Lanes	
ActivityType	Task
Status	None
InputSets	
Inputs	
OutputSets	
Outputs	
IORules	
StartQuantity	
LoopType	
LoopCondition	
LoopCounter	1
LoopMaximum	1
TestTime	After
MI_Condition	
MI_Ordering	Sequential
MI_FlowConditior	All
ComplexMI_Con	

Fig. 2. Oryx for BPMN: Modeling by drag and drop of notational elements shown on the left hand side. Right hand side shows all BPMN 1.0 attributes of a selected model element, here the Place Order activity.

4 Conclusion and Outlook

Oryx is an extensible platform for process modeling on the web. Using its extension mechanism, it aims at providing a platform to be used by the BPM community. Researchers can use the platform to implement extensions and thereby to evaluate their particular research question.

Researchers often lack access to business process models. Companies are always reluctant to provide public access to their models, so that generally few business process models are available to the public. Oryx maintains a repository of business process models. Each user can define visibility of his process models to "public", so that it can be read and evaluated by anybody accessing Oryx. While this technical feature does not solve the organizational problem of companies, we believe that, with your help, the Oryx process model repository will grow over time to a valuable resource for the BPM community. We encourage our industrial partners to use the Oryx modeling platform and to share an anonymous version of their models for scientific usage.

We invite all interested parties to use Oryx in their own research work, and to contribute to this open source project.

Acknowledgments. The authors thank the Oryx team at HPI for their development work, including Martin Czuchra, Ole Eckermann, Nicolas Peters, Willi Tscheschner, Daniel Polak, Stefan Krumnow, Sven Wagner-Boysen, Björn Wagner and all other contributors to the Oryx project.

References

1. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG) (February 2006)
2. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology (IST) (2008)
3. Keller, G., Nüttgens, M., Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
4. Petri, C.A.: Communication with Automata (in German). PhD thesis, Universität Bonn, Institut für Instrumentelle Mathematik, Schriften IIM Nr.2 (1962)
5. Pfitzner, K., Decker, G., Kopp, O., Leymann, F.: Web Service Choreography Configurations for BPMN. In: WESOA 2007, Vienna, Austria. LNCS. Springer, Heidelberg (2007)
6. Reichert, M., Rinderle, S., Kreher, U., Acker, H., Lauer, M., Dadam, P.: ADEPT Next Generation Process Management Technology. In: CAiSE Forum (2006)
7. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
8. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The prom framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)

Transforming BPMN Diagrams into YAWL Nets

Gero Decker[1], Remco Dijkman[2], Marlon Dumas[3],
and Luciano García-Bañuelos[4,⋆]

[1] Hasso Plattner Institute, Germany
gero.decker@hpi.uni-potsdam.de
[2] Eindhoven University of Technology, The Netherlands
r.m.dijkman@tue.nl
[3] University of Tartu, Estonia
marlon.dumas@ut.ee
[4] Universidad Autónoma de Tlaxcala, Mexico
lgbanuelos@gmail.com

Abstract. While the Business Process Modeling Notation (BPMN) is the de facto standard for modeling business processes on a conceptual level, YAWL allows the specification of executable workflow models. A transformation between these two languages enables the integration of different levels of abstraction in process modeling. This paper discusses the transformation of BPMN diagrams to YAWL nets and presents a tool that carries out this transformation.

1 Introduction

Process modeling occurs at different levels of abstraction. First, models serve to communicate as-is business processes, pinpoint improvement options, conduct resource and cost analysis and to capture to-be processes. The Business Process Modeling Notation (BPMN [1]) is the de facto standard for process modeling at this level. On the other hand we find languages that are targeted at technically realizing business processes, used as input for process execution engines. The Business Process Execution Language (BPEL) is a standard for implementing process-oriented compositions of web services. YAWL [2] is an alternative to BPEL, with a strictly defined execution semantics, a first-class concept of "task", and sophisticated support for data mappings and task-to-resource allocation.

While the mapping from BPMN to BPEL has been studied in detail and is implemented by several tools, the mapping from BPMN to YAWL has not yet received attention. At first glance, this mapping may seem straightforward. Indeed, the conceptual mismatch between BPMN and YAWL is not as significant as the one between BPMN and BPEL, especially with regards to control-flow structures. However, mapping BPMN to YAWL turns out to be tricky in the details, revealing subtle differences between the two languages.

The transformation from BPMN to YAWL can be used as an instrument to implement process-oriented applications. It also opens the possibility of reusing

⋆ Funded by CUDI (e-Grov Project) and by ANUIES (ORCHESTRA Project).

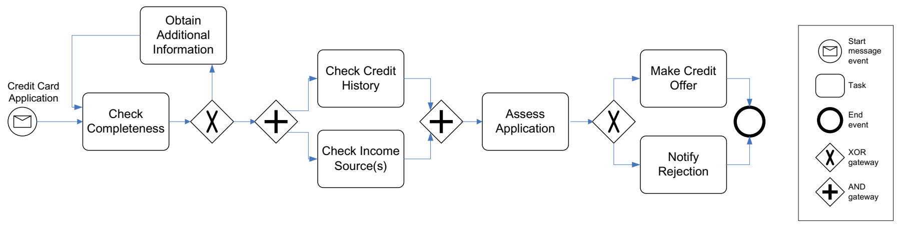
existing static analysis techniques available for YAWL. Like Petri nets, YAWL has a formally defined semantics which enables the analysis of YAWL nets to detect semantic errors such as deadlocks. At the same time, YAWL allows one to capture advanced process modeling constructs that can not always be captured in plain Petri nets, e.g. the OR-join or multi-instance activities.

The next section outlines the mapping from BPMN to YAWL. Section 3 then discusses the tool implementation and Section 4 concludes. The tool is available at: `http://is.tm.tue.nl/staff/rdijkman/bpmn.html`

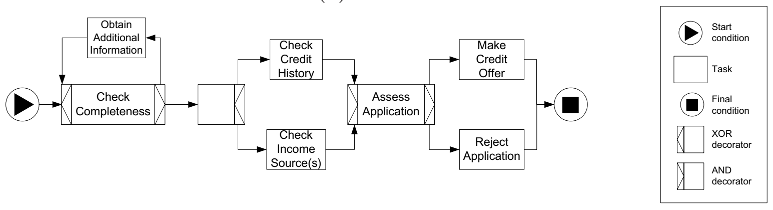
2 Overview of BPMN to YAWL Transformation

At their core, BPMN and YAWL share several common concepts. In particular, the concept of task in BPMN matches the concept of task in YAWL. Also, the concept of gateway in BPMN matches the concept of decorator in YAWL, and the concept of flow in BPMN matches the same concept in YAWL. As an illustration, Figure 1 shows a simple business process model in BPMN, and the corresponding YAWL net produced by the BPMN2YAWL tool. It can be seen from this simple example that the the join decorator of task "Check Completeness" matches the XOR-merge gateway in BPMN, meaning that the task can be reached from either of its incoming flows. Similarly, the split decorator of this same task matches the XOR-split gateway in the BPMN diagram.

At this level, one key difference between BPMN and YAWL is that YAWL does not provide the ability to directly chain several connectors together, such as an AND-split connector with a branch that leads directly to an XOR-split connector. The BPMN2YAWL tool deals with this mismatch by introducing empty YAWL tasks (i.e. YAWL tasks without decomposition) which serve purely for control routing. In the working example (Figure 1), the AND-split in the



(a) In BPMN



(b) In YAWL

Fig. 1. Simple process model in BPMN and in YAWL

BPMN diagram is mapped to an empty task with an AND decorator. The BPMN2YAWL tool ensures that such empty tasks are only created when it is necessary, so as to minimise the number of empty tasks.

Subprocess tasks in BPMN are mapped to composite tasks in YAWL. In the case where the subprocess in BPMN has an attached event (e.g. an error event), the mapping is more complicated. Figure 2 shows an exception in BPMN and the mapping onto YAWL. In the BPMN diagram, an error 'invalid policy' can occur within the 'insurance check' subprocess. This error is passed to the parent process, which then continues to 'notify customer'. In YAWL a BPMN error is mapped onto a task that sets a subprocess variable (capturing whether or not the error has occurred) to 'true'. The parent process reads this variable upon completion of the subprocess and proceeds according to the value of the variable.

The transformation covers data and resource aspects in addition to control-flow. Properties and assignments in BPMN are mapped to variables, input/output parameters and input/output transformations in YAWL. Lanes in BPMN are mapped to roles in YAWL. Pools are treated as separate business processes (and each one is mapped separately), while message flows are not covered by the mapping since their implementation depends on the communication infrastructure.

The transformation does not cover transactions and compensation handlers because these constructs do not have a direct correspondence in YAWL. Also, these constructs are underspecified in the current BPMN specification. Finally, the transformation does not cover complex gateways.

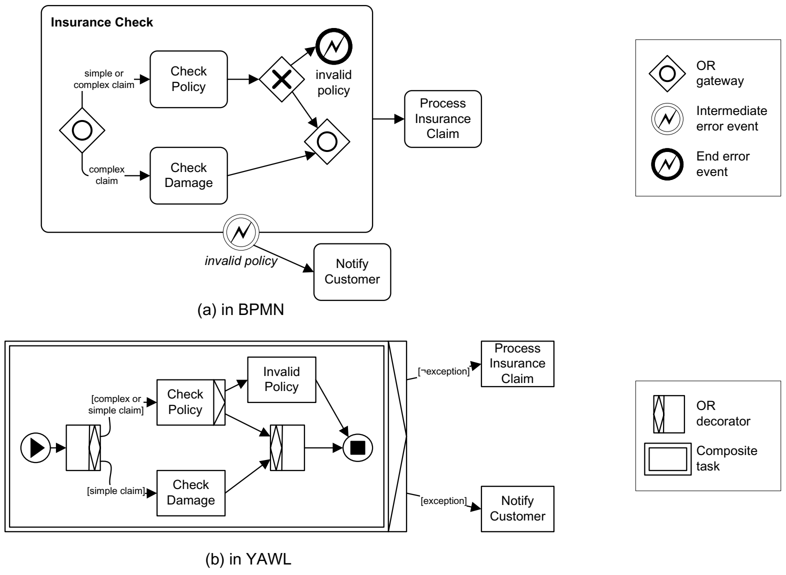


Fig. 2. Mapping attached error events from BPMN to YAWL

3 Tool Implementation

The BPMN2YAWL tool is implemented as an Eclipse plugin. The tool takes as input BPMN diagrams produced by the STP BPMN editor. The models produced by the STP BPMN editor are split in two files: one contains the XMI representation of the model, while the other contains layout information. Once installed, the BPMN2YAWL plugin provides a menu item that allows to transform the XMI file (.bpmn file). It then produces a YAWL engine file that does not contain layout information. This file can be imported into the YAWL editor which applies an automated layout algorithm.

The STP BPMN editor does not support certain features of BPMN. Specifically, it does not support the markers and properties for multi-instance activities and ad hoc activities. To overcome this limitation, the BPMN2YAWL tool is able to detect special types of text annotations: one for multi-instance activities and one for ad hoc activities. The text annotations for multi-instance activities include parameters for specifying minimum and maximum amount of instances to be started, and number of instances that need to complete before proceeding.

4 Outlook

Ongoing work aims at extending the BPMN2YAWL plugin in order to make the transformation reversible. After generating a model, the plugin will be able to propagate changes in the YAWL net into the BPMN diagram (and vice-versa) in order to maintain the models synchronized. For most constructs (e.g. tasks and gateways) the definition of this reversible transformation is straightforward. But when explicit conditions are introduced in the YAWL net, mapping these back to BPMN may prove challenging, or in some cases, impossible. We are investigating under which syntactic restrictions is it possible to preserve the reversibility of the transformation. The aim is that designers are only allowed to alter the YAWL net produced by BPMN2YAWL if the changes can be propagated back to the BPMN diagram. In tandem with this, we plan to incorporate features to visually report differences between process models in BPMN and in YAWL, so that when changes are made to either the source or the target model, the corresponding changes in the other model can be presented to the designer.

References

1. Business Process Modeling Notation, V1.1. Technical report, Object Management Group (OMG) (January 2008), http://www.omg.org/spec/BPMN/1.1/PDF/
2. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)

Goal-Oriented Autonomic Business Process Modeling and Execution: Engineering Change Management Demonstration

Dominic Greenwood

Whitestein Technologies AG, Zürich, Switzerland
dgr@whitestein.com

This demonstration paper describes the Living Systems Autonomic BPM (LS/ABPM) suite from Whitestein Technologies AG. This unique product consists of several integrated components for modelling, executing and administering business processes using a ground-breaking *goal-oriented* approach to BPM. A real and current customer case in the domain of Engineering Change Management (ECM), from Daimler AG, is used to explore the approach and features of the suite in the demonstration. Key improvements over conventional BPM techniques and technologies include business-goal oriented process modeling and extending process agility beyond the design stage by offering autonomic, self-optimizing process orchestration and execution.

The LS/ABPM suite includes a Process Modeler for goal-oriented process modeling, a Process Navigation Engine for process execution, a Management Console for process deployment and administration, and Process Task Libraries and Application Frameworks to easily build solutions for different vertical markets. The suite tools are all based on, and compliant with, standard technologies (i.e., BPMN and J2EE) allowing seamless enterprise integration.

LS/ABPM is the first BPMS to enable true goal-oriented process modeling. The Goal-Oriented BPMN (GO-BPMN)[1] notation is an extension of standard BPMN, with all model elements given a precise operational semantics allowing unambiguous model execution. Process designers use the LS/ABPM Process Modeler to capture a process' business-level purpose in terms of its goals (what/why) and the plans that are capable of achieving them (how). Owing to this loose coupling of ends and means, goal-oriented process models flexibly and concisely express the available paths across business targets. This is inherently different from conventional approaches that need to model explicit process variants, thus creating more rigidity and complexity. Both business analysts and IT specialists easily grasp and use GO-BPMN models, which are directly executable for smooth testing and deployment.

The LS/ABPM Process Navigation Engine uses the Living Systems Technology Suite (LS/TS) autonomic software technology middleware to directly map process model goals onto process instance goals using a form of Belief-Desire-Intention (BDI) execution logic [2]. The engine interprets a loaded model and dynamically selects the available plans best suited to attain goal-specified business objectives in real-time. It immediately considers changes in goals, plans and environment conditions, and autonomically adapts the executing process, constantly tracking its goals. The result is real-time process agility and responsiveness to changes in goals, plans and context-conditions as they happen during execution. This yields the greatest benefit to processes not following a strict predefined execution sequence, which is often the case for human-centric collaboration activities and increasingly also in service-oriented architectures.

The development of the LS/ABPM suite was motivated by two factors, first an observation that many enterprises have an extant desire to elevate business process modeling and management from purely the IT domain, toward providing intuitive access and use to business people who often prefer to think in terms of objectives, rather than solely the actions taken toward achieving objectives. Secondly, contemporary flexible methods of working, just-in-time organizational reaction times, distributed intra/inter-organization collaboration and constantly changing markets are creating business landscapes requiring a strong degree of real-time process agility, without sacrificing reliability or robustness. We therefore suggest that some enterprises are now discovering that a common limitation with conventional approaches to BPM is their inability to create business process models that are both meaningful to business people and capable of offering the real-time process flexibility and rapid process adaptation required to cope effectively with the pressing dynamic business conditions typifying many modern enterprises. As evidenced by our work with customers in the manufacturing domain, there is very often a need to alter executing process structures, sometimes in real-time, without perturbing the process as a whole which can continue to run as normal, accounting for updates on-the-fly. If a BPMS is not built to innately manage change in this manner the result can be reductions in both dependability and visibility, especially from a management perspective.

LS/ABPM Process Modeler

The LS/ABPM Process Modeler component of the suite provides business and IT users with a comprehensive set of tools and methodologies for goal-oriented [3] process model design, testing, and validation. Goals and plans to intuitively express business processes LS/ABPM leverages the concepts of every-day goals and plans for a more intuitive BPM experience: first define the goals a process must accomplish and then specify the possible plans that are capable of achieving these goals.

Graphical modeling language: LS/ABPM's business process specification relies on GO-BPMN to focus on business goals and business organization. GO-BPMN extends the widely used OMG-standard BPMN with semantics for modeling goals, plans and their relationships in addition to standard BPMN elements.

Separation of goals and plans: GO-BPMN models cleanly separate the (business) goals to be achieved and the plans to achieve them. Changes to any goal or plan in a GO-BPMN process model are made independently and don't have a ripple effect of consequences as they would in a sequential process model. Hence, changes can be made at any time, even during execution. The resulting adaptability and resilience to changing business conditions save time and reduce the costs associated with business process maintenance.

Separation of process and organizational model: The Process Modeler distinguishes between the goal/plan aspect of a process and the associated organization structures and constraints: what needs to be done is not mixed with who can or should do it. Clean separation between the execution of process logic and the organization of human participants helps adapt to organizational restructuring and isolates a process' business value from human resourc deployment issues.

Accessible process models for business domain experts: Thanks to the primary focus on business goals in lieu of procedures, the resulting process models are of highly descriptive nature. This intuitive method not only supports easier changes, but also enables domain experts to directly do the modeling. LS/ABPM substantially narrows the gap between business and technology.

Modular design for cooperation: Process models are modular allowing collaboration on large models, domain-specific modules, or libraries. Different people can work on reusable modules, later consolidating results into a whole model.

LS/ABPM Process Navigation Engine
The LS/ABPM Process Navigation Engine directly executes GO-BPMN process models. It pursues the defined business goals by creating a path that takes into account model changes and plans alternatives in real-time.

Direct execution of process models: LS/ABPM's GO-BPMN process models are directly executable, and the whole user interface can be automatically generated from the process model. Domain experts can test their models on their own computer for rapid process development and consistent process lifecycle management. Round-trip engineering is intrinsic in LS/ABPM, as the suite never needs to translate between a modeling notation (such as BPMN) and an execution language (such as BPEL).

Autonomic goal-oriented process performance: Each business goal connects to one or more plans, each defining a distinctive way to achieve the goal. The Process Navigation Engine selects and orchestrates the appropriate plans in real-time based on business rules and other domain-relevant context conditions. Sanity conditions can be defined and the system ensures them through continuous monitoring and prompt corrective action.

Agile process navigation and responsiveness: Agility in LS/ABPM is based on the autonomic, real-time composition and navigation of a goal-plan-context model, not on the rigid execution of explicit, situation-specific process model variants. This offers unprecedented adaptivity to dynamic business conditions.

Active coordination and cooperation between process models: The Engine performs active coordination and cooperation between multiple process models through message-driven synchronization between process controllers. Competing goals and plans do not lead to obstruction, but are autonomically resolved.

LS/ABPM Management Console
The LS/ABPM Management Console offers powerful tools for the deployment and steering of processes and other system administration tasks.

Continuous visibility of process execution and events: LS/ABPM provides detailed monitoring of running process instances. At any point, the achieved, running, and waiting goals of a process can be inspected, as well as the corresponding pending tasks.

Systematic control of executing business processes: Supervisors can control all elements (goals, plans, etc.) to fine-tune a process during execution. Other aspects under their control include visualization, data persistency, user management, role-based assignments of personnel, and security.

Operational Overview

Once a GO-BPMN models has been automatically validated by the Process Modeler it can be loaded into the Autonomic Process Navigation Engine for execution. In this respect the model itself is directly executable with no requirement to translate it via an intermediary representation such as BPEL. The engine is composed of two layers, the LS/ABPM process navigation engine and the J2EE compliant LS/TS autonomic middleware platform. In effect, the LS/ABPM process navigation engine is an LS/TS application consisting of a collection of goal-oriented agents with BDI logic engines acting as process instance controllers. An agent controller is assigned to each process instance, responsible for coordinating the process algebra and task structuring within goal-plan combinations, taking into account goal and plan preconditions. When a process model is created using the Process Modeler it is directly loaded into a new process controller agent, wherein process goals are mapped onto logical goals within the goal-oriented execution engine provided by LS/TS. The controller then executes the process instance by initiating the entire goal hierarchy and waiting for appropriate triggers to be sensed within the system environment to activate goals and move forward with process execution. Each process controller is at the heart of an autonomic feedback control loop that uses observations made of the *system* being affected by the process instance to effect decisions within the corresponding process instance relating to, for example, which goals should be activated and which plans selected to meet goal requirements. Such autonomic control allows process instances to be self-configured and self-optimized bringing about both process flexibility and resilience.

References

1. Greenwood, D., Rimassa, G.: Autonomic goal-oriented business process management. In: Proceedings of the Third International Conference on Autonomic and Autonomous Systems (ICAS 2007) (2007)
2. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco (1995)
3. Tibco: Goal-driven business process management: Creating agile business processes for an unpredictable environment. Tibco Whitepaper (2006)

COREPRO$_{Sim}$: A Tool for Modeling, Simulating and Adapting Data-Driven Process Structures*

Dominic Müller[1,2], Manfred Reichert[1], Joachim Herbst[2], Detlef Köntges[1,2], and Andreas Neubert[1]

[1] Institute of Databases and Information Systems, Ulm University, Germany
{dominic.mueller,manfred.reichert,andreas.neubert}@uni-ulm.de
[2] Dept. GR/EPD, Daimler AG Group Research & Advanced Engineering, Germany
{joachim.j.herbst,detlef.koentges}@daimler.com

Abstract. Industry is increasingly demanding IT support for large engineering process structures consisting of hundreds up to thousands of synchronized processes. In technical domains, such process structures are characterized by their strong relation to the assembly of a product (e.g., a car); i.e., resulting process structures are *data-driven.* The strong linkage between data and processes can be utilized for automatically creating process structures as well as for (dynamically) adapting them at a high level of abstraction. This paper presents the COREPRO$_{Sim}$ demonstrator which enables sophisticated support for modeling, coordinating and (dynamically) adapting data-driven process structures. COREPRO$_{Sim}$ substantiates the COREPRO approach which provides a new paradigm for the integration of complex data and process structures.

1 Introduction

In the engineering domain, the development of complex products (e.g., cars) necessitates the coordination of large *process structures.* Managing such structures, however, is a complex task which is only rudimentarily supported by current workflow technology [1]. Process structures often show a strong linkage with the assembly of the product; i.e., the processes to be coordinated can be explicitly assigned to the different product components. Further, synchronizations of these processes are correlated with the relations existing between the product components. We denote such process structures as *data-driven.* COREPRO utilizes information about product components and their relations for modeling, coordinating, and (dynamically) adapting process structures based on given (product) data structures. For example, the assembly of a (product) data structure can be used to automatically create the related process structure [2].

The adaptation of process structures constitutes a particular challenge. When adding or removing a car component (e.g., a navigation system), for example, the instantiated process structure has to be adapted accordingly; i.e., processes as well as synchronization relations between them have to be added or removed. When changing a (product) data structure during runtime, in addition, the

* This work has been funded by *Daimler AG Group Research* and has been conducted in the *COREPRO* project (http://www.uni-ulm.de/dbis)

running process structure must be adapted on-the-fly, but without leading to faulty synchronizations (e.g. deadlocks). To cope with this challenge, again, our approach takes benefit from the strong linkage between process structure and (product) data structure. Data structure changes can be translated into consistent adaptations of the corresponding process structure [3]. Thus, changes of data-driven process structures can be introduced by users at a high level of abstraction, which reduces complexity as well as cost of change significantly.

This paper sketches COREPRO$_{Sim}$ – a demonstrator enabling the modeling, enactment and (dynamic) adaptation of data-driven process structures. It enhances the COREPRO$_{Modeler}$, our first demonstrator supporting the manual modeling of data-driven process structures [4]. COREPRO$_{Sim}$ has been realized using the *Eclipse Graphical Editing Framework*. It implements an instantiation concept for automatically creating data-driven process structures and a runtime framework for simulating them [2,3]. Furthermore, COREPRO$_{Sim}$ translates (dynamic) changes of currently processed data structures into corresponding adaptations of the related process structures. Finally, consistency is checked to ensure that dynamic adaptations result again in a sound process structure.

2 COREPRO Framework and Demo Description

The COREPRO modeling framework considers the sequence of states a (data) object goes through during its lifetime. A car component, for example, passes states like *tested* and *released*. Generally, state transitions are triggered when executing the processes which modify the respective object (e.g., *test* and *release*). An *object life cycle* (OLC) then constitutes an integrated and user-friendly view on the states of a particular object and its manipulating processes (cf. Fig. 1b).

Based on a collection of OLCs and their synchronizations, large process structures can be built. OLC state transitions do not only depend on the processes associated with the respective object, but also on the states and state transitions of other objects. As example consider a car prototype, which will be only tested if all subsystems (e.g., engine, chassis and navigation system) are tested before. By connecting the states of different OLCs, a logical view on the data-driven process structure results (cf. Fig. 1d).

Five steps become necessary to create a data-driven process structure using COREPRO$_{Sim}$ (cf. Fig. 1). Step 1 deals with the specification of a domain-specific *data model*, which defines object and relation types, and therefore constitutes the schema for instantiating concrete (product) *data structures*. An object type represents a class of objects within the data model (cf. Fig. 1a), which can be instantiated multiples times (cf. Fig. 1c).

Step 2 (cf. Fig. 1b) is related to the modeling of OLCs. Internally, OLCs are mapped to *state transition systems* whose states correspond to object states and whose (internal) state transitions are associated with object-specific processes. Non-deterministic state transitions are realized by associating different internal state transitions with same source state and process, and by adding a process result as condition (e.g., `P2` with possible results `0` and `1` in Fig. 1b).

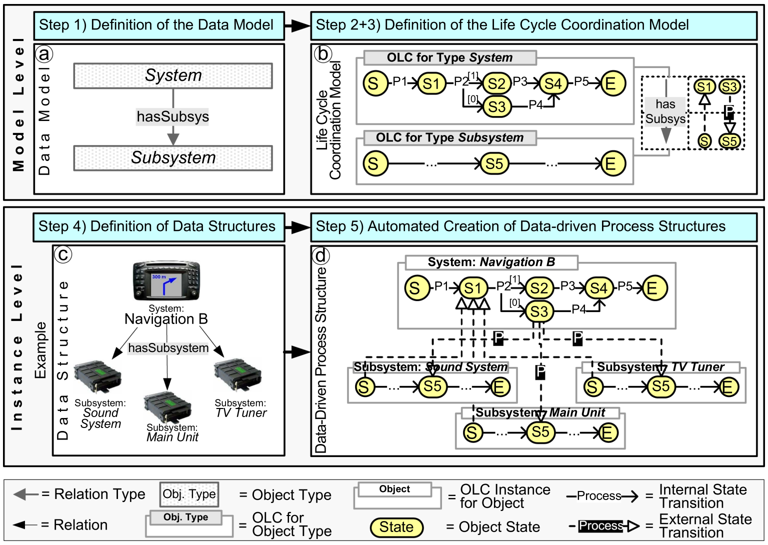

Model Level
Step 1) Definition of the Data Model
Data Model
System
hasSubsys
Subsystem
Step 2+3) Definition of the Life Cycle Coordination Model
Life Cycle Coordination Model
OLC for Type System
OLC for Type Subsystem
has Subsys
Instance Level
Step 4) Definition of Data Structures
Example Data Structure
System: Navigation B
hasSubsystem
Subsystem: Sound System
Subsystem: Main Unit
Subsystem: TV Tuner
Step 5) Automated Creation of Data-driven Process Structures
Data-Driven Process Structure
System: Navigation B
Subsystem: Sound System
Subsystem: TV Tuner
Subsystem: Main Unit
= Relation Type
= Relation
Obj. Type = Object Type
Obj. Type = OLC for Object Type
Object = OLC Instance for Object
State = Object State
Process = Internal State Transition
Process = External State Transition

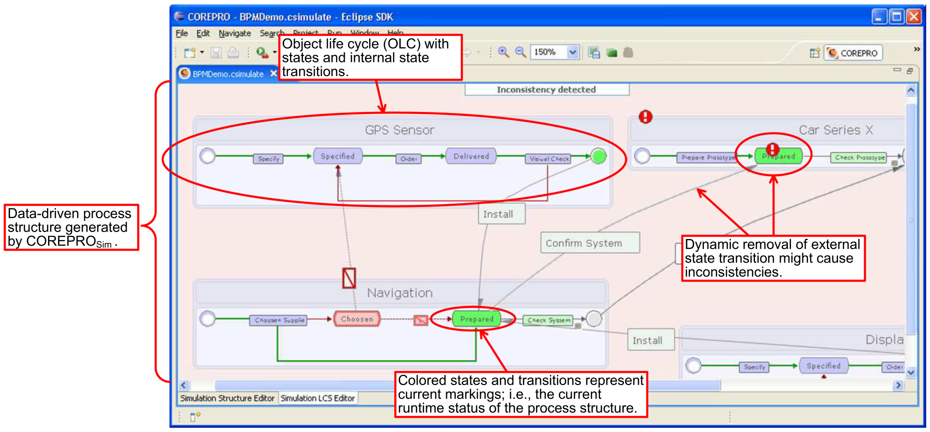
Fig. 1. Procedure for Creating Data-Driven Process Structures

Step 3 deals with the state dependencies existing between the OLCs of *different* object types. In COREPRO, an OLC dependency is expressed in terms of *external state transitions* between concurrently enacted OLCs (which together form the process structure). Like an internal OLC state transition, an external state transition can be associated with the enactment of a process. To benefit from the *strong linkage* between object relations and OLC dependencies, external state transitions are mapped to object relation types (cf. Fig. 1b).

Steps 4 + 5 are related to the instance level. COREPRO$_{Sim}$ allows to instantiate different data structures based on a given data model (cf. Fig. 1c) and to automatically create related process structures (cf. Fig. 1d). A data-driven process structure includes an OLC instance for every object from the data structure. Likewise, as specified in Step 3, for each relation in the data structure external state transitions are added to the process structure; e.g., for every `hasSubsystem` relation in the data structure from Fig. 1c, corresponding external state transitions are added (cf. Fig. 1d).

As result we obtain an executable process structure describing the dynamic aspects of the given data structure (cf. Fig. 1d). To ensure a correct dynamic behavior, COREPRO$_{Sim}$ allows for checking soundness of the process structure based on the concepts described in [2].

When simulating data-driven process structures, COREPRO$_{Sim}$ uses different *markings* to reflect the current runtime status (cf. Fig. 2). We annotate states as well as (internal and external) state transitions with respective markings. By analyzing *state markings*, for example, we can immediately figure out whether a particular state of a process structure has been already passed, is currently activated, has been skipped, or has not been reached yet. *Transition markings*,


Object life cycle (OLC) with states and internal state transitions.

Data-driven process structure generated by COREPRO$_{Sim}$.

Dynamic removal of external state transition might cause inconsistencies.

Colored states and transitions represent current markings; i.e., the current runtime status of the process structure.

Fig. 2. Simulation and Dynamic Change of Process Structure with COREPRO$_{Sim}$

in turn, indicate whether the associated process has been started, skipped or completed. The use of markings further eases consistency checking and runtime status adaptations in the context of dynamic changes of process structures.

To cope with flexibility requirements of engineering processes, we allow users (e.g., engineers) to perform dynamic process structure adaptations. In COREPRO$_{Sim}$, this is accomplished by automatically translating changes of the data structure (cf. Fig. 1c) into adaptations of the respective process structure [3]. Removing an object, for example, leads to the removal of the related OLC. Such dynamic adaptations must not violate soundness of the process structure. To ensure this, COREPRO$_{Sim}$ constrains changes to certain runtime states (i.e., markings) of the process structure (cf. Fig. 2).

In summary, COREPRO$_{Sim}$ constitutes a powerful demonstrator realizing the concepts developed in the COREPRO project [1,2,3]. It is currently applied in the context of a case study in the automotive industry. In future, we will extend the current prototype with extensive mechanisms for runtime exception handling and integrate existing data sources and applications.

References

1. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 368–377. Springer, Heidelberg (2006)
2. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–147. Springer, Heidelberg (2007)
3. Müller, D., Reichert, M., Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 48–63. Springer, Heidelberg (2008)
4. Müller, D., Reichert, M., Herbst, J., Poppa, F.: Data-driven design of engineering processes with COREPRO$_{Modeler}$. In: WETICE 2007, pp. 376–378. IEEE, Los Alamitos (2007)

Author Index