LNCS 7481

Alistair Barros
Avigdor Gal
Ekkart Kindler (Eds.)

Business Process Management

10th International Conference, BPM 2012
Tallinn, Estonia, September 2012
Proceedings

Springer

Alistair Barros   Avigdor Gal
Ekkart Kindler (Eds.)

Business Process Management

10th International Conference, BPM 2012
Tallinn, Estonia, September 3-6, 2012
Proceedings

Springer

Volume Editors

Alistair Barros
Queensland University of Technology
Faculty of Science and Technology
Information Systems Discipline
Business Process Management Group
126 Margaret Street, Brisbane, QLD, 4000, Australia
E-mail: alistair.barros@qut.edu.au

Avigdor Gal
Technion - Israel Institute of Technology
Faculty of Industrial Engineering and Management
Technion City, 32000 Haifa, Israel
E-mail: avigal@ie.technion.ac.il

Ekkart Kindler
Technical University of Denmark
Informatics and Mathematical Modelling
Richard Petersens Plads, 2800 Kgs. Lyngby, Denmark
E-mail: eki@imm.dtu.dk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32884-8 e-ISBN 978-3-642-32885-5
DOI 10.1007/978-3-642-32885-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012944722

CR Subject Classification (1998): H.3.5, F.3.2, J.1, H.4.1, D.2, H.5.3, K.4.3

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

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

Printed on acid-free paper

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

Preface

BPM 2012 was the tenth conference in a series that provides a prestigious forum for researchers and practitioners in the field of business process management (BPM). The conference was organized by the University of Tartu, Estonia, and took place during September 3–6, 2012, in the capital city of Estonia, Tallinn.

In response to the call for papers we received 126 submissions. Each paper was evaluated by three or four Program Committee members and one senior Program Committee member. We accepted 17 regular papers (13.5% acceptance rate) and seven short papers with an overall acceptance rate of 19%.

The call for papers of this year encouraged, in addition to the traditional research areas of BPM, contributions that apply or combine established and new techniques to the specific challenges of BPM. These include model-driven architecture, Web engineering, service-oriented architectures, complex event processing, and cloud computing. The conference's scientific program consisted of eight research sessions: "Process Quality," "Conformance and Compliance," "BPM Applications," "Process Model Analysis," "BPM and the Cloud," "Requirements and Performance," "Process Mining," and "Refactoring and Optimization."

The conference hosted three invited keynote presentations with a main theme of past and future. Alejandro Buchmann, professor for databases and distributed systems in the Computer Science Department of Technische Universität Darmstadt, outlined the architecture of future BPM systems using events. Wil van der Aalst, professor of information systems at Eindhoven University of Technology (TU/e), reflected on a decade of BPM conferences. Finally, Stephen A. White, the "father" of BPMN and a BPM architect at IBM, talked about the future of BPMN.

We thank the authors and presenters, whose papers made this program strong. We appreciate the senior Program Committee members, the Program Committee members, and the external reviewers for their thorough review and discussion of the submitted papers. We thank the BPM Steering Committee for their valuable guidance.

We thank the BPM 2012 team. Marlon Dumas, General Chair, and his Organizing Committee, Raimundas Matulevicius, Laura Kalda, and Georg Singer. Also, Stephen A. White (Industry Chair), Marcello La Rosa and Pnina Soffer (Workshop Co-chairs), Niels Lohmann and Simon Moser (Demo Co-chairs), Florian Daniel and Volker Gruhn (Tutorial and Panel Co-chairs), Michael zur Muehlen, Cesare Pautasso, and Jianmin Wang (Publicity Co-chairs), and Boualem Benatallah, Manfred Reichert, and Karsten Wolf (Doctoral Consortium Co-chairs).

We thank the conference sponsors: Bizagi (platinum), IBM and HP (gold), and Signavio (silver) and our publisher, Springer, for their continuous support of BPM. Finally, the use of EasyChair is appreciated, making our lives so much easier.

September 2012

Avigdor Gal
Ekkart Kindler
Alistair Barros

Organization

BPM 2012 was organized in Tallin, Estonia, by the University of Tartu, Estonia.

Steering Committee

Wil van der Aalst (Chair)	Eindhoven University of Technology, The Netherlands
Boualem Benatallah	University of New South Wales, Sydney, Australia
Fabio Casati	University of Trento, Italy
Peter Dadam	University of Ulm, Germany
Jörg Desel	Fernuniversität in Hagen, Germany
Schahram Dustdar	Vienna University of Technology, Austria
Arthur ter Hofstede	Queensland University of Technology, Australia
Barbara Pernici	Politecnico di Milano, Italy
Mathias Weske	HPI at the University of Potsdam, Germany

Executive Committee

General Chair

Marlon Dumas	University of Tartu, Estonia

Program Chairs

Alistair Barros	Queensland University of Technology, Australia
Avigdor Gal	Technion - Israel Institute of Technology
Ekkart Kindler	Technical University of Denmark

Industry Chair

Stephen A. White	IBM, USA

Workshop Chairs

Marcello La Rosa	Queensland University of Technology, Australia
Pnina Soffer	University of Haifa, Israel

Demo Chairs

Niels Lohmann	University of Rostock, Germany
Simon Moser	IBM Research and Development Lab Böblingen, Germany

Tutorial and Panel Chairs

Florian Daniel	University of Trento, Italy
Volker Gruhn	University of Duisburg-Essen, Germany

Publicity Chairs

Michael zur Muehlen	Stevens Institute of Technology, USA
Cesare Pautasso	University of Lugano, Switzerland
Jianmin Wang	Tsinghua University, China

Doctoral Consortium Chairs

Boualem Benatallah	University of New South Wales, Australia
Manfred Reichert	University of Ulm, Germany
Karsten Wolf	University of Rostock, Germany

Organizing Committee

Raimundas Matulevicius	University of Tartu, Estonia (Chair)
Laura Kalda	University of Tartu, Estonia
Georg Singer	University of Tartu, Estonia

Senior Program Committee

Wil van der Aalst	Eindhoven University of Technology
Boualem Benatallah	University of New South Wales
Peter Dadam	University of Ulm
Jörg Desel	Fernuniversität in Hagen
Schahram Dustdar	TU Wien
Arthur ter Hofstede	Queensland University of Technology
Stefan Jablonski	University of Bayreuth
Jana Koehler	Hochschule Luzern
Frank Leymann	University of Stuttgart
Jan Mendling	WU Vienna
Manfred Reichert	University of Ulm
Hajo Reijers	Eindhoven University of Technology
Michael Rosemann	Queensland University of Technology
Mathias Weske	HPI at the University of Potsdam

Program Committee

Karl Aberer	EPFL
Ana Karla Alves de Medeiros	Independent Consultant, Den Bosch
Ahmed Awad	University of Potsdam
Soeren Balko	AGT
Christoph Bussler	Xtime, Inc.

Fabio Casati	University of Trento
Francisco Curbera	IBM Research
Gero Decker	Signavio
Alin Deutsch	University of California San Diego
Remco Dijkman	Eindhoven University of Technology
Boudewijn van Dongen	Eindhoven University of Technology
Johann Eder	University of Klagenfurt
Gregor Engels	University of Paderborn
Dirk Fahland	Eindhoven University of Technology
Hans-Georg Fill	University of Vienna
Luciano García-Bañuelos	University of Tartu
Giuseppe De Giacomo	Sapienza Università di Roma
Holger Giese	Hasso Plattner Institute
Claude Godart	LORIA, Université de Lorraine
Thomas Hildebrandt	IT University of Copenhagen
Marta Indulska	The University of Queensland
Leonid Kalinichenko	Russian Academy of Science
Gerti Kappel	Vienna University of Technology
Dimka Karastoyanova	University of Stuttgart
Marite Kirikova	Riga Technical University
Agnes Koschmider	Karlsruhe Institute of Technology (KIT)
John Krogstie	IDI, NTNU
Jochen Kuester	IBM Research
Akhil Kumar	Penn State University
Niels Lohmann	Universität Rostock
Peter Loos	IWi at DFKI, Saarland University
Heiko Ludwig	IBM Research
Daniel Moody	Ozemantics
Hamid Motahari	HP Labs
Bela Mutschler	University of Applied Sciences Ravensburg-Weingarten
Prabir Nandi	IBM Research
Alex Norta	University of Helsinki
Markus Nüttgens	Universität Hamburg
Andreas Oberweis	Universität Karlsruhe
Hervé Panetto	CRAN, University of Lorraine, CNRS
Theresa Pardo	Center for Technology in Government, University at Albany, SUNY
Oscar Pastor Lopez	Universitat Politecnica de Valencia
Cesare Pautasso	University of Lugano
Viara Popova	University of Tartu
Frank Puhlmann	inubit AG
Jan Recker	Queensland University of Technology
Stefanie Rinderle-Ma	University of Vienna
Domenico Saccà	University of Calabria
Shazia Sadiq	The University of Queensland

Erich Schikuta	University of Vienna
Heiko Schuldt	University of Basel
Guttorm Sindre	NTNU
Pnina Soffer	University of Haifa
Christian Stahl	Eindhoven University of Technology
Mark Strembeck	Vienna University of Economics and BA, Institute of Information Systems, New Media Lab
Harald Störrle	Danmarks Tekniske Universitet
Stefan Tai	Karlsruhe Institute of Technology
Samir Tata	Institut TELECOM; TELECOM SudParis; CNRS UMR Samovar
Farouk Toumani	Limos, Blaise Pascal University, Clermont-Ferrand
Alberto Trombetta	Insubria University
Aphrodite Tsalgatidou	National and Kapodistrian University of Athens
Hagen Völzer	IBM Research - Zurich
Jianmin Wang	Tsinghua University
Barbara Weber	University of Innsbruck
Jens Weber	University of Victoria
Matthias Weidlich	Technion - Israel Institute of Technology
Petia Wohed	DSV, SU/KTH
Karsten Wolf	Universität Rostock
Andreas Wombacher	University of Twente
Christian Zirpins	Seeburger AG

Additional Reviewers

Aghaee, Saeed
Athanasopoulos, George
Aubry, Alexis
Barba Rodríguez, Irene
Baumgrass, Anne
Beyhl, Thomas
Bonetta, Daniele
Breitenbücher, Uwe
Brosch, Petra
Buijs, Joos C.A.M.
Chinosi, Michele
Conforti, Raffaele
D'Alessandro, Pietro
De Masellis, Riccardo
Di Ciccio, Claudio
Dunkl, Reinhold
Elliger, Felix
Fahland, Dirk
Fazal-Baqaie, Masud
Felli, Paolo
Fischer, Robin
Folino, Francesco
Furfaro, Angelo
Gerth, Christian
Gierds, Christian
Gouyon, David
Greco, Gianluigi
Grieger, Marvin
Guzzo, Antonella
Hebig, Regina
Heinis, Thomas
Hildebrandt, Stephan
Hipp, Markus
Hofreiter, Birgit
Hoisl, Bernhard
Huemer, Christian
Iera, Antonio
Jalali, Amin
Jin, Tao
Kabicher-Fuchs, Sonja
Khalaf, Rania
Kopp, Oliver
Kouki, Pigi
Krause, Christian
Kuhlenkamp, Joern
Lakshmanan, Geetika
Lambers, Leen
Leitner, Maria

Lezoche, Mario
Loures, Eduardo
Ly, Linh Thao
Mangler, Juergen
Marrella, Andrea
Matijacic, Michel
Mayerhofer, Tanja
Michelberger, Bernd
Montali, Marco
Mueller-Wickop, Niels
Musliu, Nysret
Nagel, Benjamin
Narendula, Rammohan
Patrizi, Fabio
Penicina, Ludmila
Perrin, Olivier
Peternier, Achille
Pinggera, Jakob
Pontieri, Luigi
Ramezani, Elham
Reiter, Michael
Russo, Alessandro
Schefer-Wenzl, Sigrid
Schleicher, Daniel
Schultz, Martin
Schumm, David
Schuster, Nelly
Schönböck, Johannes
Seidl, Martina
Soltenborn, Christian
Torres, Victoria
Wanek, Helmut
Wen, Lijie
Werner, Michael
Westergaard, Michael
Widl, Magdalena
Yahia, Esma
Yerva, Surender
Zhu, Xinwei

Table of Contents

Process Mining

A Decade of Business Process Management Conferences: Personal Reflections on a Developing Discipline

Wil M.P. van der Aalst

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands
www.vdaalst.com

Abstract. The Business Process Management (BPM) conference series celebrates its tenth anniversary. This is a nice opportunity to reflect on a decade of BPM research. This paper describes the history of the conference series, enumerates twenty typical BPM use cases, and identifies six key BPM concerns: process modeling languages, process enactment infrastructures, process model analysis, process mining, process flexibility, and process reuse. Although BPM matured as a research discipline, there are still various important open problems. Moreover, despite the broad interest in BPM, the adoption of state-of-the-art results by software vendors, consultants, and end-users leaves much to be desired. Hence, the BPM discipline should not shy away from the key challenges and set clear targets for the next decade.

1 History of the BPM Conference Series

The first International Conference on Business Process Management (BPM 2003) took place in Eindhoven in the last week of June 2003, ten years ago. Therefore, I would like to take the opportunity to describe the origin of the conference series and look back on a decade of BPM research.

The direct trigger to organize the first BPM conference was generated by Grzegorz Rozenberg in his capacity as chair of the Petri Nets Steering Committee. He invited us to organize the 24th International Conference on Application and Theory of Petri Nets (Petri Nets 2003) in Eindhoven. Moreover, he stimulated me to organize a co-located event. This is how the idea for "BPM 2003" was born. The subtitle of BPM 2003 – "On the Application of Formal Methods to Process-Aware Information Systems" – illustrates the relation with Petri Nets conference. Together with Arthur ter Hofstede and Mathias Weske, I served as PC chair of BPM 2003 [11]. I was also PC chair of Petri Nets 2003 (together with Eike Best) [5]. Kees van Hee, Hajo Reijers, Eric Verbeek, and many others from the Technische Universiteit Eindhoven (TU/e) were involved in the organization of both conferences. BPM 2003 was remarkably successful considering it was organized for the first time: 77 papers were submitted of which 25 papers were accepted. Moreover, various BPM vendors and consultants participated. Carl Adam Petri gave a keynote, received a prestigious Royal medal ("Commandeur in de Orde van de Nederlandse Leeuw"), and it was interesting to see him talking with BPM vendors about workflows. Afterwards, we decided to continue organizing BPM conferences given the growing interest in the topic and enthusiasm of the BPM 2003 participants. We decided to disconnect the BPM conference from the Petri Nets conference (to clearly show that BPM is not linked to a specific formalism). Apparently, these were wise decisions; in subsequent years the BPM conference series evolved into one of the premier information systems conferences. The following list shows the ten BPM conferences organized thus far:

- BPM 2003 (Van der Aalst, Ter Hofstede, Weske, Reijers, Van Hee, et al.), Eindhoven, The Netherlands [11],
- BPM 2004 (Weske, Desel, Pernici, et al.), Potsdam, Germany [19],
- BPM 2005 (Godart, Perrin, Van der Aalst, Benatallah, Casati, Curbera, et al.), Nancy, France [4],
- BPM 2006 (Dustdar, Fiadeiro, Sheth, Rosenberg, et al.), Vienna, Austria [23],
- BPM 2007 (Rosemann, Dumas, Alonso, Dadam, Ter Hofstede, et al.), Brisbane, Australia [14],
- BPM 2008 (Pernici, Casati, Dumas, Reichert, et al.), Milan, Italy [22],
- BPM 2009 (Reichert, Dadam, Reijers, Eder, Dayal, et al.), Ulm, Germany [18],
- BPM 2010 (Zur Muehlen, Hull, Mendling, Tai, et al.), Hoboken, USA [31],
- BPM 2011 (Toumani, Rinderle-Ma, Wolf, Hacid, Schneider, et al.), Clermont-Ferrand, France [36], and
- BPM 2012 (Dumas, Kindler, Gal, Barros, et al.), Tallinn, Estonia [15].

Since 2005 the conference features co-located workshops. The main proceedings are published in Springer's Lecture Notes in Computer Science (LNCS) and the workshop proceedings are published in Springer's Lecture Notes in Business Information Processing (LNBIP). From 2003 to 2009, selected papers were invited for special issues of Data & Knowledge Engineering (DKE). Since 2010, each year the best papers are invited for a special issue of Information Systems (IS).

Although BPM 2003 was the first real BPM conference, there were some informal predecessor workshops. Together with Giorgio De Michelis and Skip Ellis, I organized the "Workflow Management: Net-based Concepts, Models, Techniques and Tools" (WFM'98) workshop [12]. This workshop was co-located with Petri Nets 1998 in Lisbon, Portugal. Together with Jörg Desel and Roland Kaschek, I also organized the "Software Architectures for Business Process Management" (SABPM'99) workshop [6]. This workshop was one of the pre-conference workshops of CAiSE 1999 in Heidelberg, Germany. Based on these events, I started to work with Jörg Desel and Andreas Oberweis on an edited book during my sabbatical at the University of Karlsruhe. In 2000, the book "Business Process Management: Models, Techniques, and Empirical Studies" [7] appeared. This LNCS volume can be seen as a direct predecessor of BPM 2003 given the topic and people involved. Therefore, I will include it in my later analysis.

2 Pre-BPM Era

Business Process Management (BPM) has various roots in both computer science and management science. Therefore, it is difficult to pinpoint the starting point of BPM.

However, it is obvious that BPM existed long before the term became popular. Therefore, I reflect on the origins of BPM by summarizing major developments before the conference in 2003.

Since the industrial revolution, productivity has been increasing because of technical innovations, improvements in the organization of work, and the use of information technology. Adam Smith (1723-1790) showed the advantages of the division of labor. Frederick Taylor (1856-1915) introduced the initial principles of scientific management. Henry Ford (1863-1947) introduced the production line for the mass production of "black T-Fords". It is easy to see that these ideas are used in today's BPM systems.

Around 1950 computers and digital communication infrastructures started to influence business processes. This resulted in dramatic changes in the organization of work and enabled new ways of doing business. Today, innovations in computing and communication are still the main drivers behind change in business processes. So, business processes have become more complex, heavily rely on information systems, and may span multiple organizations. *Therefore, process modeling has become of the utmost importance.* Process models assist in managing complexity by providing insights and by documenting procedures. Information systems need to be configured and driven by precise instructions. Cross-organizational processes can only function properly if there is a common agreement on the required interactions. As a result, process models are widely used in todays organizations.

In the last century many process modeling techniques have been proposed. In fact, the well-known Turing machine described by Alan Turing (1912-1954) can be viewed as a process model. It was instrumental in showing that many questions in computer science are undecidable. Moreover, it added a data component (the tape) to earlier transition systems. Petri nets play an even more prominent role in BPM as they are graphical and able to model concurrency. In fact, most of the contemporary BPM notations and systems use a token-based semantics adopted from Petri nets. Petri nets were proposed by Carl Adam Petri (1926-2010) in 1962. This was the first formalism able to model concurrency. Concurrency is very important as in business processes many things happen in parallel. Many cases may be handled at the same time and even within a case there may be various activities enabled or running concurrently. Therefore, a BPM system should support concurrency natively.

Since the seventies there has been consensus on the modeling of data (cf. the Relational Model by Codd [17] and the Entity-Relationship Model by Chen [16]). Conversely, process modeling is best characterized by the term "divergence". There is little consensus on the fundamental concepts. Despite the availability of established formal languages (e.g., Petri nets and process calculi) industry has been pushing ad-hoc/domain-specific languages. As a result there is a plethora of systems and languages available today (BPMN, BPEL, UML, EPCs, etc.).

Figure 1 sketches the emergence of BPM systems and their role in the overall information system architecture. Initially, information systems were developed from scratch, i.e., everything had to be programmed, even storing and retrieving data. Soon people realized that many information systems had similar requirements with respect to data management. Therefore, this generic functionality was subcontracted to a database system. Later, generic functionality related to user interaction (forms, buttons, graphs, etc.)

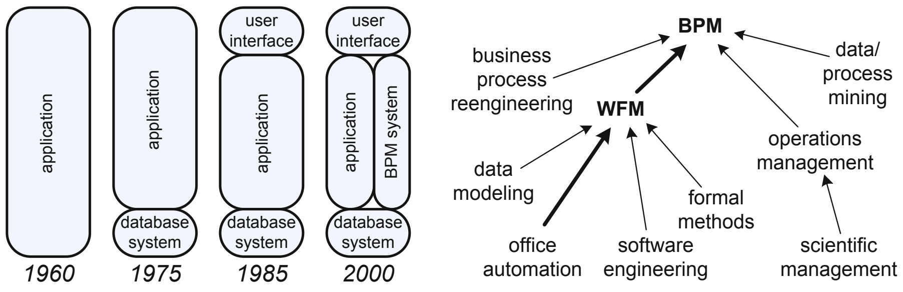


Fig. 1. Historic view on information systems development illustrating that BPM systems can be used to push processes out of the application (left, adapted from [1])) and an overview of some disciplines that contributed to the development of the BPM discipline (right)

was subcontracted to tools that can automatically generate user interfaces. This trend continued in different areas. BPM systems can be seen in this context: a BPM system takes care of process-related aspects. Therefore, the application can focus on supporting individual/specific tasks. In the mid-1990s many *Workflow Management* (WFM) systems became available. These systems focused on automating workflows with little support for analysis, flexibility, and management. BPM systems provide much broader support, e.g., by supporting simulation, business process intelligence, case management, etc. However, compared to the database market, the BPM market is much more diverse and there is no consensus on notations and minimal capabilities. This is not a surprise as process management is much more complex than data management.

A good starting point from a scientific perspective is the early work on *office information systems*. In the seventies, people like Skip Ellis, Anatol Holt, and Michael Zisman already worked on so-called office information systems, which were driven by explicit process models [2, 24, 26, 25, 29, 30, 34, 40, 42, 41]. Ellis et al. [24, 26, 25] developed office automation prototypes such as Officetalk-Zero, Officetalk-D and Officetalk-P at Xerox PARC in the late 1970s. These systems used Information Control Nets (ICN), a variant of Petri nets, to model processes. Office metaphors such as inbox, outbox and forms were used to interact with users. The prototype office automation system SCOOP (System for Computerizing of Office Processes) developed by Michael Zisman also used Petri nets to represent business processes [40, 42, 41]. It is interesting to see that pioneers in office information systems already used Petri-net variants to model office procedures. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the eighties. In the nineties, there was a clear revival of the ideas already present in the early office automation prototypes [8]. This is illustrated by the many commercial WFM systems developed in this period.

In the mid-nineties there was the expectation that WFM systems would get a role comparable to Database Management (DBM) systems. Most information systems subcontract their data management to DBM systems and there are just a few widely used products. However, despite the availability of BPM/WFM systems, process management is not subcontracted to such systems at a scale comparable to DBM systems. The application of "pure" BPM/WFM systems is still limited to specific industries such as banking and insurance. However, BPM/WFM technology is often hidden inside other systems. For example, ERP systems like SAP and Oracle provide workflow engines. Many other platforms include workflow-like functionality. For example, integration and application infrastructure software such as IBM's WebSphere provides extensive process support. In hindsight, it is easy to see why process management cannot be subcontracted to a standard BPM/WFM system at a scale comparable to DBM systems. As illustrated by the varying support of workflow patterns [9, 37], process management is much more complex than data management. *BPM is multifaceted, complex, and difficult to demarcate.* Given the variety in requirements and close connection to business concerns, it is often impossible to use generic BPM/WFM solutions. Therefore, BPM functionality is often embedded in other systems and BPM techniques are frequently used in a context with conventional information systems.

The first BPM conference in 2003 marked the transition from WFM to BPM [10]. Since then the BPM discipline matured. Today, the relevance of BPM is acknowledged by practitioners (users, managers, analysts, consultants, and software developers) and academics. This is illustrated by the availability of BPM systems, conferences, and books such as [3, 8, 13, 21, 28, 32–35, 39].

3 BPM Use Cases

One of the goals of this paper is to reflect on 10 years of BPM research by analyzing the proceedings of past BPM conferences (BPM 2003 - BPM 2011) and the edited book [7] that can be viewed as a predecessor of the first BPM conference (see Section 1). In total 289 papers were analyzed by tagging each paper with the *use cases* and *key concerns* described in the remainder.

Before conducting this analysis, I identified 20 use cases as shown in Figure 2. For example, use case *design model* (DesM) refers to the creation of a process model from scratch and use case *discover model from event data* (DiscM) refers to the automated generation of a process model using process mining techniques. Models constructed through use case DesM are descriptive (D), normative (N), and/or executable (E). This is denoted by the "$D|N|E$" tag in Figure 2. A model discovered through process mining (DiscM) is typically not normative as it is based on observed behavior (cf. "$D|E$" tag). Models can also be obtained through *selection* (SelM), *merging* (MerM), or *composition* (CompM). Most papers were tagged with one or two use cases. The tagging was based on the most important use case(s) the paper aims to support. For example, the paper "Graph Matching Algorithms for Business Process Model Similarity Search" [20] presented at BPM 2009 was tagged with the use case *select model from collection* (SelM) since the paper presents an approach to rank process models in a repository based on some initial model.

Use cases *design configurable model* (DesCM), *merge models into configurable model* (MerCM), and *configure configurable model* (ConCM) deal with configurable models. There models – also referred to as reference models – correspond to families of concrete process models (i.e., variants of the same process). Use case *refine model*

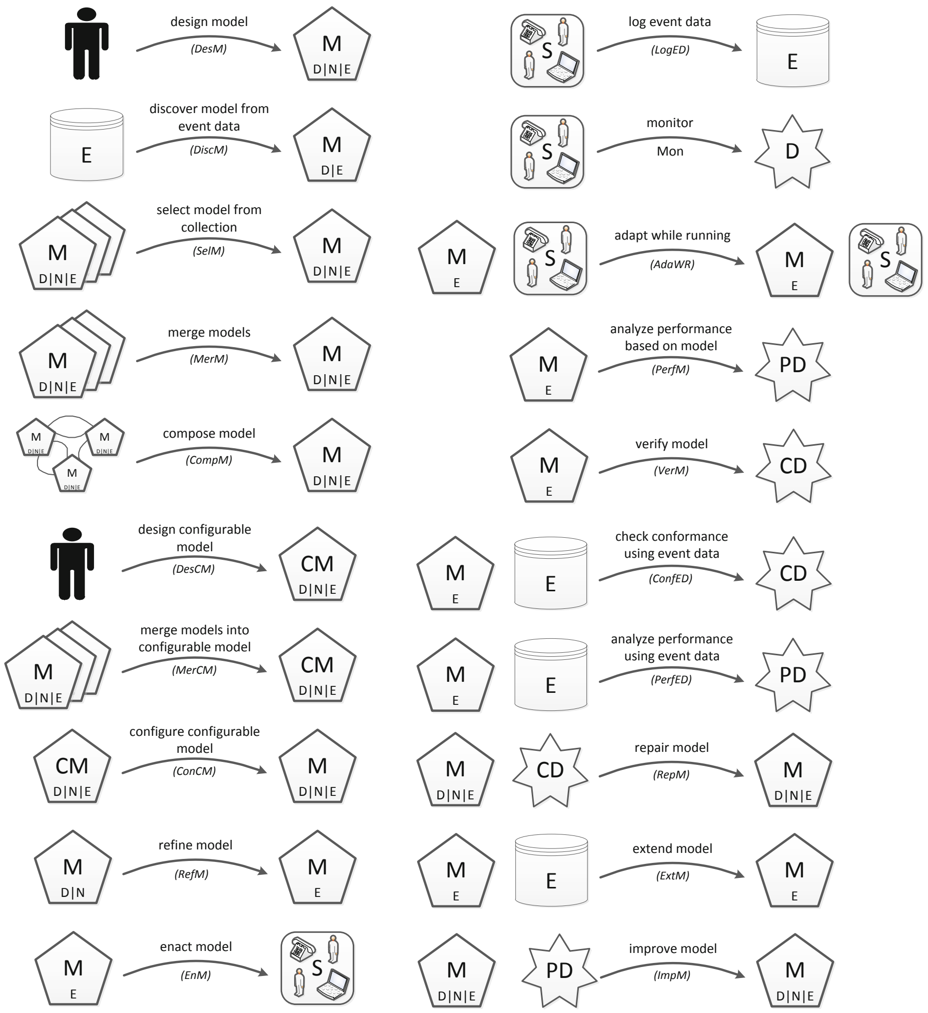


Fig. 2. Twenty BPM use cases: *M* = model, *E* = event data, *CM* = configurable model, *S* = information system, *D* = diagnostics, *CD* = conformance-related diagnostics, *PD* = performance-related diagnostics. Models can be tagged as descriptive (*D*), normative (*N*), or executable (*E*).

(RefM) refers to the transformation of a descriptive or normative model into an executable model. Executable models can be enacted by applying use case *enact model* (EnM).

Use cases can be chained together as shown in Figure 3. The last use case in the chain is *analyze performance using event data* (PerfED) which requires event data and an executable model. PerfED use may be used to uncover bottlenecks observed in reality. Use case *check conformance using event data* (ConfED) requires similar input but focuses on deviations rather than performance. Use case *analyze performance based on model* (PerfM) focuses on performance without using event data, e.g., model-based

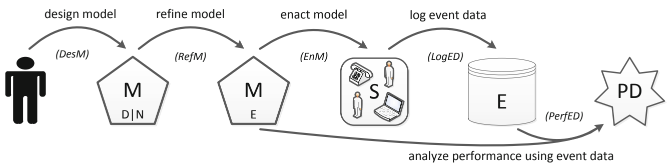


Fig. 3. Composite use case obtained by chaining five use cases: DesM, RefM, EnM, LogED, and PerfED

simulation used to analyze flow times, utilization, and bottlenecks. Use case *monitor* (Mon) refers runtime analysis without using any model (e.g., flow time analysis without looking "inside the process").

Use case *adapt while running* (AdaWR) refers to changing the system and model at runtime to provide flexibility, e.g., modifying a model and subsequently migrating process instances. The paper "Instantaneous Soundness Checking of Industrial Business Process Models" [27] also presented at BPM 2009 is a typical example of a paper tagged with use case *verify model* (VerM). In this paper 735 industrial business process models are checked for soundness (absence of deadlock and lack of synchronization) using three different approaches.

Use case *repair model* (RepM) changes a model based on conformance-related diagnostics, e.g., deviations are used to correct the model. Use case *improve model* (ImpM) is similar but focuses on performance-related diagnostics. For example, bottleneck analysis is used to redesign the process. Use case *extend model* (ExtM) refers to the process mining scenario where a model is enriched using information extracted from the event log [3].

Papers where typically tagged with one dominant use case, but sometimes more tags were used. In total, 367 tags were assigned (on average 1.18 use cases per paper). By simply counting the number of tags per use case and year, the relative frequency of each use case per year can be established. For example, for BPM 2009 four papers were tagged with use case *discover model from event data* (DiscM). The total number of tags assigned to the 23 BPM 2009 papers is 30. Hence, the relative frequency is $4/30 = 0.133$. Table 1 shows all relative frequencies including the one just mentioned. The table also shows the average relative frequency of each use case over all years. These averages are shown graphically in Figure 4.

Figure 4 shows that use cases *design model* (DesM) and *enact model* (EnM) are most frequent. This is not very surprising as these use cases are less specific than most other use cases. The third most frequent use case – *verify model* (VerM) – is more surprising (relative frequency of 0.144). An example paper having such a tag is [27] which was mentioned before. Over the last decade there has been considerable progress in this area and this is reflected by various papers presented at BPM. In this context it is remarkable that the use cases *monitor* (Mon) and *analyze performance using event data* (PerfED) have a much lower relative frequency (respectively 0.009 and 0.015). Given the practical needs of BPM one would expect more papers presenting techniques to diagnose and improve the performance of business processes.

Figure 5 shows changes of relative frequencies over time. The graph shows a slight increase in process-mining related topics. However, no clear trends are visible due to

Table 1. Relative importance of use cases in [7], [11], [19], [4], [23], [14], [22], [18], [31], [36], and [15]. Each of the 289 papers was tagged with, on average, 1.18 use cases (typically 1 or 2 use cases per paper). The table shows the relative frequency of each use case per year. The last row shows the average over 10 years. All rows add up to 1.

	design model	discover model from event data	select model from collection	merge models	compose model	design configurable model	merge models into configurable model	configure configurable model	refine model	enact model	log event data	monitor	adapt while running	analyze performance based on model	verify model	check conformance using event data	analyze performance using event data	repair model	extend model	improve model
year	DesM	DiscM	SelM	MerM	CompM	DesCM	MerCM	ConCM	RefM	EnM	LogED	Mon	AdaWR	PerfM	VerM	ConfED	PerfED	RepM	ExtM	ImpM
2000	0.406	0.000	0.000	0.000	0.031	0.000	0.000	0.000	0.000	0.188	0.000	0.000	0.063	0.125	0.188	0.000	0.000	0.000	0.000	0.000
2003	0.306	0.028	0.056	0.000	0.056	0.000	0.000	0.028	0.000	0.222	0.028	0.000	0.139	0.000	0.111	0.000	0.000	0.000	0.028	0.000
2004	0.348	0.130	0.000	0.000	0.043	0.000	0.000	0.000	0.000	0.217	0.000	0.000	0.043	0.087	0.087	0.000	0.000	0.000	0.000	0.043
2005	0.216	0.039	0.039	0.000	0.098	0.000	0.000	0.000	0.000	0.294	0.000	0.000	0.059	0.078	0.137	0.000	0.000	0.000	0.000	0.039
2006	0.094	0.038	0.057	0.019	0.132	0.000	0.000	0.000	0.019	0.245	0.019	0.000	0.075	0.019	0.226	0.019	0.019	0.000	0.000	0.019
2007	0.231	0.128	0.026	0.026	0.051	0.026	0.026	0.077	0.077	0.077	0.026	0.026	0.026	0.051	0.103	0.000	0.026	0.000	0.000	0.000
2008	0.227	0.045	0.023	0.000	0.045	0.000	0.023	0.000	0.023	0.182	0.045	0.000	0.023	0.091	0.136	0.000	0.045	0.023	0.068	0.000
2009	0.167	0.133	0.067	0.000	0.033	0.000	0.033	0.000	0.133	0.133	0.033	0.000	0.000	0.033	0.167	0.033	0.033	0.000	0.000	0.000
2010	0.167	0.133	0.100	0.000	0.000	0.033	0.000	0.033	0.033	0.300	0.000	0.000	0.000	0.000	0.100	0.067	0.000	0.000	0.033	0.000
2011	0.061	0.091	0.121	0.030	0.000	0.000	0.061	0.000	0.000	0.121	0.000	0.061	0.000	0.000	0.182	0.152	0.030	0.061	0.030	0.000
average	0.222	0.077	0.049	0.007	0.049	0.006	0.014	0.014	0.029	0.198	0.015	0.009	0.043	0.048	0.144	0.027	0.015	0.008	0.016	0.010

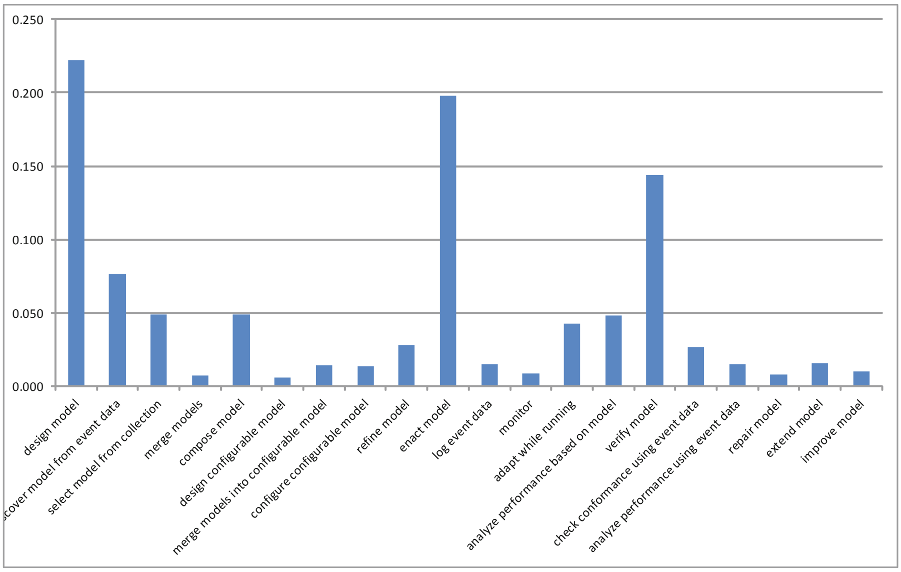


Fig. 4. Average relative importance of use cases (taken from Table [1])

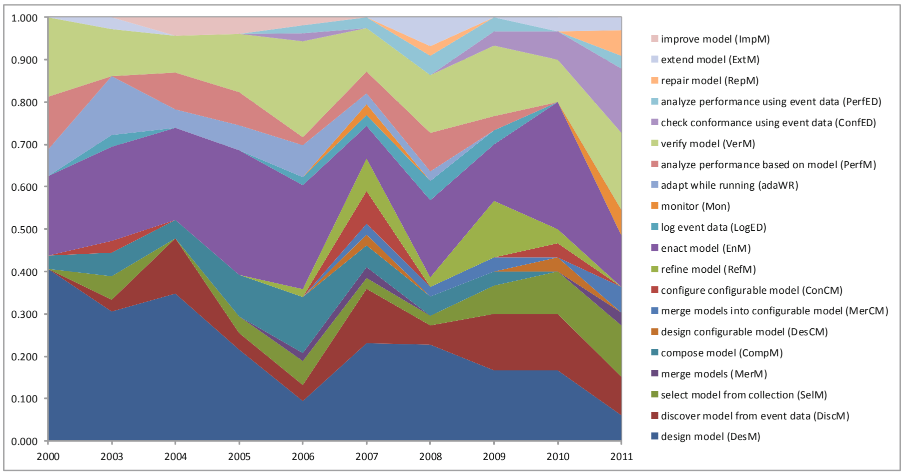


Fig. 5. Development of the relative importance of each use case plotted over time (derived from Table [1])

the many use cases and small numbers. Therefore, all BPM papers were also analyzed based on the six key concerns presented next.

4 BPM Key Concerns

To provide a trend analysis at a coarser level of granularity, I also identified six *key concerns* before analyzing the 289 papers:

Process modeling languages. The first concern is about the process modeling language to be used. Many papers propose a new notation or evaluate existing ones. There are often competing requirements, e.g., the language should be very expressive but simple at the same time [9]. Languages intended for automated process execution (e.g., BPEL) may be very different from languages mainly used for discussion and documentation (e.g., EPCs). Other languages may be tailored towards verification (e.g., WF-nets) or process mining (e.g., C-nets or hidden Markov chains).

Process enactment infrastructures. The second key concern is the creation of an infrastructure to execute, support, and monitor processes. Examples of topics dealing with this concern are the development of workflow engines, service-oriented computing, interoperability, cloud computing, enterprise application integration, work distribution systems, etc.

Process model analysis. The third concern refers to the analysis of processes based on models without using event data. Papers addressing this concern are about topics such as soundness verification, simulation, model checking, queueing networks, controllability, etc.

Process mining. The fourth concern refers to all analysis techniques that are driven by event data. For example, process discovery techniques that construct a model based on observed traces. Process mining is not limited to discovery and also includes conformance checking and extension [3]. Conformance checking can be used to check if reality, as recorded in an event log, conforms to the model and vice versa. Extension adds a new perspective to the process model by cross-correlating it with an event log.

Process flexibility. The fifth concern addresses the problem that existing WFM/BPM systems tend to be inflexible. Process flexibility can be seen as the ability to deal with both foreseen and unforeseen changes, by varying or adapting those parts of the business process that are affected by them, while retaining the essential format of those parts that are not impacted by the variations [38]. Papers on case handling, adaptive workflows, late-binding, declarative languages, etc. all aim at adding flexibility.

Process reuse. The sixth concern addresses the problem that (parts of) processes are often “reinvented” rather than reused. The challenge is to avoid duplicate modeling and implementation efforts. Configurable process models, reference models, process repositories, similarity search, etc. are typical approaches to promote reuse.

As for the use cases, the papers in [7], [11], [19], [4], [23], [14], [22], [18], [31], [36], and [15] were tagged with one, or sometimes more, key concerns. A total of 342 tags were assigned to the 289 papers (1.18 tag per paper on average). The tags were used to determine the relative frequencies listed in Table 2. For example, for BPM 2010 I tagged four papers with key concern *process reuse*. The total number of tags for BPM 2010 is 25. Hence, the relative frequency is $4/25 = 0.16$. The bottom row gives the average relative frequency of each concern over all 10 years. Both trends and averages are depicted graphically in Figure 6. As expected, the first three concerns are most frequent. The fourth and sixth concern (process mining and process reuse) are gaining importance, whereas the relative frequency of the process flexibility concern seems to decrease over time.

Table 2. Relative importance of the six key concerns over the years. All rows add up to 1. The last row shows the average relative frequency over all years.

year	process modeling languages	process enactment infrastructures	process model analysis	process mining	process flexibility	process reuse
2000	0.355	0.161	0.290	0.000	0.161	0.032
2003	0.325	0.200	0.250	0.050	0.075	0.100
2004	0.286	0.238	0.238	0.143	0.048	0.048
2005	0.288	0.231	0.212	0.058	0.096	0.115
2006	0.154	0.308	0.288	0.096	0.077	0.077
2007	0.387	0.097	0.194	0.194	0.065	0.065
2008	0.324	0.108	0.297	0.135	0.081	0.054
2009	0.148	0.111	0.370	0.222	0.037	0.111
2010	0.240	0.240	0.200	0.160	0.000	0.160
2011	0.143	0.171	0.200	0.314	0.000	0.171
average	0.265	0.187	0.254	0.137	0.064	0.093

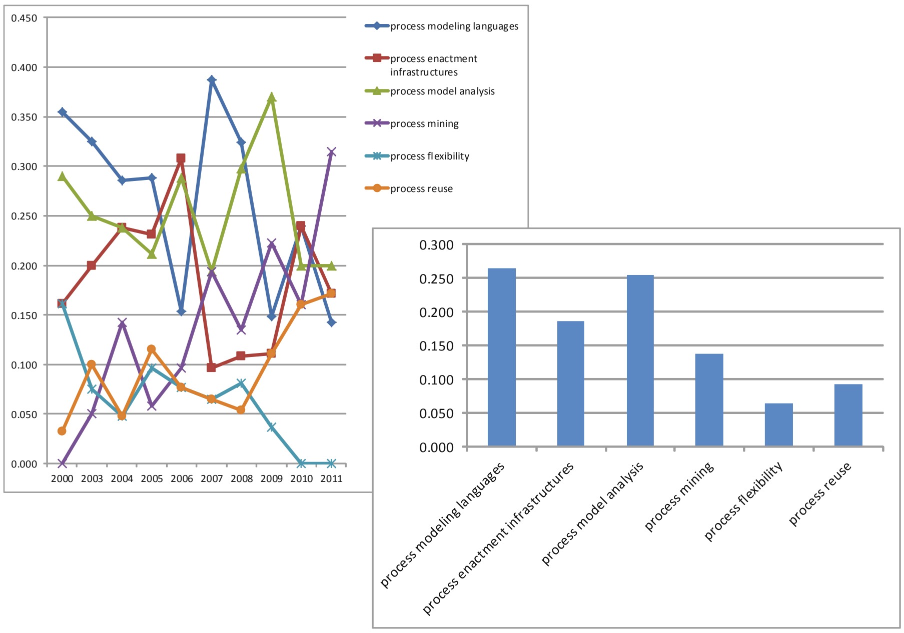


Fig. 6. Visualization of the results from Table 2 for each of the six key concerns: changes of relative importance over time (left) and average relative frequency (right)

5 Reflections

Before discussing some insights obtained when tagging the 289 BPM papers, I first reflect on the analysis method used.

The tagging of a paper with use cases and key concerns is highly subjective. It is unlikely that another expert would come to the exact same tags for each paper. For example, to tag a paper one needs to decide what the key contribution of the paper is. Many papers are rather broad and difficult to classify. For example, papers on topics such as "Social BPM", "BPM maturity", and "BPM Security" cannot be tagged easily, because these topics seem orthogonal to the uses cases and key concerns. This explains why broad use cases like *design model* (DecM) and *enact model* (EnM) score relatively high.

The key concerns were identified before tagging the papers. In hindsight there seem to be at least three potentially missing concerns: *process integration*, *patterns*, and *collaboration*. Many papers are concerned with web services and other technologies (e.g., SaaS, PaaS, clouds, and grids) to integrate processes. These are now tagged as *process enactment infrastructures* (second concern). In the BPM proceedings there are various papers proposing new patterns collections or evaluating existing languages using the existing workflow patterns [9, 37]. These are now tagged as *process modeling languages* (first concern). Another recurring concern seems to collaboration, e.g., collaborative modeling or system development.

Given a process, different *perspectives* can be considered: the *control-flow* perspective ("What activities need to be executed and how are they ordered?"), the *organizational* perspective ("What are the organizational roles, which activities can be executed by a particular resource, and how is work distributed?"), the *case/data* perspective ("Which characteristics of a case influence a particular decision?"), and the *time* perspective ("What are the bottlenecks in my process?"), etc. The use cases and key concerns are neutral/orthogonal with respect to these perspectives. Although most papers focus on the control-flow perspective, there are several papers that focus on the organizational perspective, e.g., papers dealing with optimal resource allocations or role-based access control. It would have been useful to add additional tags to papers based on the perspectives considered.

Next, I reflect on the papers themselves. Comparing papers published in the early BPM proceedings with papers published in more recent BPM proceedings clearly shows that the BPM discipline progressed at a remarkable speed. The understanding of process modeling languages improved and analysis techniques have become much more powerful. Despite the good quality of most papers, some weaknesses can be noted when reflecting on the set of 289 BPM papers.

- Many papers introduce a new modeling language. The need for such new languages is often unclear, and, in many cases, the proposed language is never used again. A related problem is that many papers spend more time on presenting the context of the problem rather than the actual analysis and solution. For example, there are papers proposing a new verification technique for a language introduced in the same paper. Consequently, the results cannot be used or compared easily.
- Many papers cannot be linked to one of the 20 use cases of Section 3 in a straightforward manner. Authors seem to focus on originality rather than relevance and

show little concern for real-life use cases. One could argue that such papers propose solutions for rather exotic or even artificial problems.

– Many papers describe implementation efforts; however, frequently the software is not available for the reader. Moreover, regrettably, many of the research prototypes seem to "disappear" after publication. As a result, research efforts get lost.
– Many papers include case studies, e.g., to test a new technique or system, which is good. Unfortunately, most case studies seem rather artificial. Often the core contribution of the paper is not really evaluated or the case study is deliberately kept vague.

6 Outlook

The BPM discipline clearly matured over the last decade. Nevertheless, there are many exciting open problems and BPM will remain highly relevant. Whereas it used to be in vogue to present a technique which just exists on paper, more recent BPM papers tend to describe an implementation of the ideas (proof-of-concept). Moreover, the importance of empirical evaluation seems to increase. Consider for example a new fictive verification technique X. It used to be acceptable to just present the idea and some proof of X's correctness. Now it is expected that X is implemented and some experimental results need to be provided in the paper. In the future authors will be required to include an empirical evaluation using large collections of process models and really compare results using benchmark examples. This is good development. Researchers should focus on the hard BPM problems rather than trying to come up with original (fake or artificial) problems. Hence, it would be good to organize competitions centering around challenging BPM problems that need to be solved urgently.

The "Big Data" wave provides new prospects for BPM research. Organizations are recording large amounts of event data and start understanding the potential value of such data. This is great opportunity to promote "evidence-based BPM", e.g., research ideas can be empirically evaluated using real data. This will increase the credibility of BPM research and help convincing practitioners to adopt new ideas.

This paper lists 20 use cases. As mentioned, for some papers it is unclear to see which use case the authors are trying to support. It would be interesting to require BPM authors to tag their paper with the use cases addressed by their work. This could serve as a reality check for the authors and help to structure the field (e.g., to find reviewers and related work). Such ideas would of course require further development of the use cases presented in this paper (including requests for input, open discussions, and consensus building).

Although BPM research is extremely relevant and its results are useable by many organizations, there are no strong industrial counterparts willing to invest in foundational BPM research. Established consultancy and software firms tend to be rather conservative and end-user organizations are not aware of advances in BPM research. Topics are hyped, but the actual realization of ideas often leaves much to be desired. As a result "BPM is everywhere and nowhere". Compare this to other domains (e.g., the high-tech industry) where there are fewer players, but these are able to invest in R&D and cannot survive by selling only buzzwords. Given these circumstances, it is important to develop *shared* open-source software as a means to influence practitioners. Unfortunately, many prototypes are developed from scratch and “fade onto oblivion” when the corresponding research project ends. Therefore, there is a need to maintain larger open-source software platforms shared between different groups. However, this is not easy. For example, open-source tools like ProM and YAWL have more than 400,000 lines of code. It is a challenge to sustain such efforts over a longer period of time. The BPM community would benefit from shared development efforts using a limited number of large open-source software platforms (instead of developing many throw-away prototypes), but this requires a change in research culture.

Given all of these challenges, I’m looking forward to a new decade of exhilarating and simulating BPM research!

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. van der Aalst, W.M.P.: Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 1–65. Springer, Heidelberg (2004)
3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
4. van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.): BPM 2005. LNCS, vol. 3649. Springer, Heidelberg (2005)
5. van der Aalst, W.M.P., Best, E. (eds.): ICATPN 2003. LNCS, vol. 2679. Springer, Heidelberg (2003)
6. van der Aalst, W.M.P., Desel, J., Kaschek, R. (eds.): Software Architectures for Business Process Management (SABPM 1999), Heidelberg, Germany. Forschungsbericht Nr. 390, University of Karlsruhe, Institut AIFB, Karlsruhe (1999)
7. van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.): Business Process Management. LNCS, vol. 1806. Springer, Heidelberg (2000)
8. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2004)
9. 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)
10. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
11. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.): BPM 2003. LNCS, vol. 2678. Springer, Heidelberg (2003)
12. van der Aalst, W.M.P., De Michelis, G., Ellis, C.A. (eds.): Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM 1998), Lisbon, Portugal. UNINOVA, Lisbon (1998)
13. van der Aalst, W.M.P., Stahl, C.: Modeling Business Processes: A Petri Net Oriented Approach. MIT Press, Cambridge (2011)
14. Alonso, G., Dadam, P., Rosemann, M. (eds.): BPM 2007. LNCS, vol. 4714. Springer, Heidelberg (2007)
15. Barros, A., Gal, A., Kindler, E. (eds.): BPM 2012. LNCS, vol. 7481. Springer, Heidelberg (2012)

16. Chen, P.P.: The Entity-Relationship Model: Towards a unified view of Data. ACM Transactions on Database Systems 1, 9–36 (1976)
17. Codd, E.F.: A Relational Model for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)
18. Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.): BPM 2009. LNCS, vol. 5701. Springer, Heidelberg (2009)
19. Desel, J., Pernici, B., Weske, M. (eds.): BPM 2004. LNCS, vol. 3080. Springer, Heidelberg (2004)
20. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
21. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons (2005)
22. Dumas, M., Reichert, M., Shan, M.-C. (eds.): BPM 2008. LNCS, vol. 5240. Springer, Heidelberg (2008)
23. Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.): BPM 2006. LNCS, vol. 4102. Springer, Heidelberg (2006)
24. Ellis, C.A.: Information Control Nets: A Mathematical Model of Office Information Flow. In: Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, Boulder, Colorado, pp. 225–240. ACM Press (1979)
25. Ellis, C.A., Nutt, G.J.: Workflow: The Process Spectrum. In: Sheth, A. (ed.) Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems, Athens, Georgia, pp. 140–145 (May 1996)
26. Ellis, C.A., Nutt, G.J.: Office Information Systems and Computer Science. ACM Computing Surveys 12(1), 27–60 (1980)
27. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
28. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer, Berlin (2010)
29. Holt, A.W.: Coordination Technology and Petri Nets. In: Rozenberg, G. (ed.) APN 1985. LNCS, vol. 222, pp. 278–296. Springer, Heidelberg (1986)
30. Holt, A.W., Saint, H., Shapiro, R., Warshall, S.: Final Report on the Information Systems Theory Project. Technical Report RADC-TR-68-305, Griffiss Air Force Base, New York (1968)
31. Hull, R., Mendling, J., Tai, S. (eds.): BPM 2010. LNCS, vol. 6336. Springer, Heidelberg (2010)
32. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London (1996)
33. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River (1999)
34. zur Muehlen, M.: Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems. Logos, Berlin (2004)
35. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, Berlin (2012)
36. Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.): BPM 2011. LNCS, vol. 6896. Springer, Heidelberg (2011)
37. 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)

38. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process Flexibility: A Survey of Contemporary Approaches. In: Dietz, J.L.G., Albani, A., Barjis, J. (eds.) CIAO! 2008 and EOMAS 2008. LNBIP, vol. 10, pp. 16–30. Springer, Heidelberg (2008)
39. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Berlin (2007)
40. Zisman, M.D.: Representation, Specification and Automation of Office Procedures. PhD thesis, University of Pennsylvania, Warton School of Business (1977)
41. Zisman, M.D.: Office Automation: Revolution or Evolution. Sloan Management Review 13(3), 1–16 (1978)
42. Zisman, M.D.: Use of Production Systems for Modeling Asynchronous Concurrent Processes. In: Pattern-Directed Inference Systems, pp. 53–68 (1978)

From Calls to Events: Architecting Future BPM Systems

Alejandro Buchmann, Stefan Appel, Tobias Freudenreich, Sebastian Frischbier, and Pablo E. Guerrero

TU Darmstadt, Germany
lastname@dvs.tu-darmstadt.de

Abstract. Contemporary BPM systems fit very well with traditional architectures that are based on a pull invocation principle, such as SOA. The proliferation of sensors and streams of events has led to event driven architectures that decouple event producers and consumers. EDAs are push-based and support different control structures. Future BPM systems must therefore deal both with pull and push-based architectures. In this talk we will analyze the interplay of the different architectures, their components and the desirable and achievable correctness notions and non-functional properties.

1 Introduction

By 2020 it is predicted that a large portion of enterprises will be process driven [20]. The key driving forces that are mentioned are customer centricity, operational excellence, new regulations, new business models, and a global workforce. In this world Business Process Management (BPM) will play a deciding role. A BPM system, according to [28] is "a generic software system that is driven by explicit process designs to enact and manage operational business processes". We will refer to BPM systems in their most general form, including not only traditional workflow management (WFM) but also Business Process Analysis (BPA), Business Process Monitoring (BAM) and Process Aware Information Systems (PAIS).

The traditional BPM lifecycle consists of process design, system configuration, process enactment, and diagnosis. The result of business process diagnosis may result in process redesign. The automatic extraction of new business processes through process mining of the event log is an alternative to manual process (re)design. Emergent software [13] is a closely related topic. Researchers in this area are trying to develop software systems that can adapt to changing environments producing new behaviors from local events. The goal of Emergent Enterprise Software Systems is to facilitate the cooperation across enterprise boundaries by self-adapting the business process.

The degree of human involvement in the (re)design of business processes can vary and is a continuum ranging from completely manual design to autonomic behavior with enforcement of self-x properties, and finally the automatic extraction of new processes. This continuum is the top layer of Figure 1.

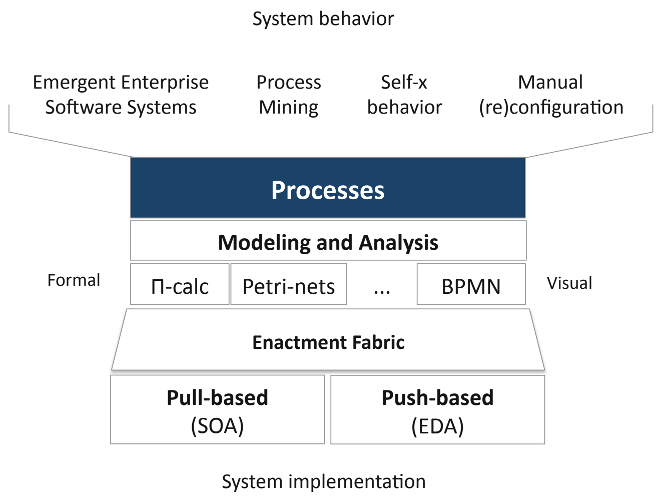


Fig. 1. Business processes from the system perspective

Processes are at the core of process aware software systems. Processes must be represented by a model in the corresponding formalism or language. The modeling formalism may range from intuitive and mostly graphic representations, such as BPMN, to more formal notations based, for example, on petri-nets or variants of process algebras, such as CCS or Pi-calculus. To automate the process design or redesign, a formal model is needed. Informal or intuitive graphical notations are good for human consumption but unsuitable for automatic process extraction. This is the middle tier in Figure 1.

Processes must be mapped to the corresponding execution environment for enactment. The by now established platform for enactment of business processes are web services. Many of the most interesting new applications, however, are monitoring and reactive applications responding to events that are sensed by a multitude of sensors or are generated through event aggregation, composition and/or derivation. These applications are not well served by web services and service oriented architectures. There are two main reasons: 1) Web services employ a request-respond invocation mechanism, i.e., they are primarily pull-based. Monitoring and reactive applications are better served by push-based dissemination (of streams) of events and their associated data. 2) Web services have been conceived as black boxes that hide the middleware they are running on. Only recent efforts [21] have tried to provide for vertical service composition that enables the execution of services on the best suited middleware platform. The lower portion of Figure 1 shows the two complementary paradigms and the corresponding architectures: Service Oriented Architecture (SOA) for more traditional pull-based invocation and Event Driven Architecture (EDA) for push-based invocation. It is our conviction that environments for execution of a broad range of business processes must offer both kinds of invocation. However, since SOA is well established for the processing of business processes, we will concentrate on EDA, its advantages and challenges.

The advantages of EDA consist in the loose coupling between producers and consumers of events that lead to easy extensibility of systems, and the celerity with which events are detected, propagated and reacted to. The challenges consist in establishing a common understanding of the semantics of events and the execution of triggered tasks, and in establishing understandable correctness criteria for the execution of event driven business processes.

The remainder of this paper discusses events and event composition in Section 2; Section 3 presents briefly EDA; Section 4 discusses quality of service in Event Driven Architectures; Section 5 presents the concluding discussion.

2 Events and Event Composition

There exist many interpretations of what an event is. Often these different interpretations are community-specific. Therefore, we start by introducing a classification proposed by Chandy and Schulte in their book 'Event Processing, Designing IT Systems for Agile Companies' [7] that reflects some of the ongoing debate. There are three views of what an event is, each being true from one perspective but none describing the whole truth or being equally useful in different situations.

– *An event is a happening of interest.* This is an activity-based view of events and quite intuitive. For example, a person entering a room or a container leaving a warehouse. While easy to understand and useful for describing a situation it is not helpful computationally and must be translated into measurable quantities.
– *An event is a (meaningful) change of state.* This definition considers any change in the model of reality that is observed as a potential event. Such a change could be the detection of an RFID tag on a container at the warehouse gate. This approach is very useful in implementing event based systems but often requires deriving the more abstract event (e.g. the container leaving the warehouse) from one or more observations. This definition depends on detecting change and makes it difficult to handle observations that may be of interest even though no change occurred.
– *An event is a detectable condition that can trigger a notification.* This is a reporting-based view of events. This definition is somewhat more general than the change of state view in that it also considers the absence of change as a detectable condition. However, it depends on a reporting capability in the form of notifications, defined as an event-triggered signal sent to a run-time recipient. This view of events is quite useful to detect all kinds of events, even observations that do not depend on a change, but anything that is either not observed or not reported is not considered an event.

In [14] we expanded the second definition above, by including time as a basic dimension, thereby allowing us to consider two observations taken at different times to be considered as two events even if none of the other dimensions describing the state have changed. Notifications are the natural way of notifying parties interested in an event, usually via a publish/subscribe notification service.

2.1 Important Concepts in Event Driven Architectures

Event types and event instances: The event type determines the attributes of an event and its structure. An event instance is a particular materialization of an event type and may be identified either through an event-identifier or through a unique combination of attribute values.

Event objects, event representations and notifications: An event carries a timestamp and descriptive parameters and is typically represented as a tuple of values. This representation is called the event object. Event objects can have other representations, for example, an XML document. When the event object is packaged into a message, we talk about an event notification.

Temporal events are first class citizens. We distinguish between absolute temporal events and relative temporal events. Absolute temporal events conceptually consist only of a timestamp and the source and have a simple representation. The timestamp has a given granularity that is often encoded in the format of the timestamp and an identifier of the clock that may be omitted in centralized systems. Relative temporal events are time offsets relative to a base event. The base event can be either an absolute temporal event or it could be any other event. The granularity of the offset can be specified as a function of the capability of the clock and the requirements of the application.

State events are sometimes identified as a special kind of event to distinguish them from change events, i.e. the change of state. State events are observations in which the observed quantity may not have changed but only time has advanced. Given the framework proposed earlier to include time as an integral part of the definition of state, there is no need to make a distinction for state events. Subscribers to state events must provide the frequency at which they need the observation events.

Change events refer to any change of state. We must restrict ourselves to meaningful state changes. What is meaningful is determined by an event consumer who subscribes to events. The producer may generate events at a certain rate, for example, every second, but one consumer of those events is only interested in every tenth event while another consumer needs that type of event only every minute, i.e. every 60th event.

Event filtering is applied to eliminate those events that were observed but for which no interest exists. Event filtering can occur in principle anywhere along the path between producer and consumer of a certain event type. Placing the filter near the event source minimizes the notification costs and the processing cost for the event consumer.

Simple events are basic events that are not the result of some composition.

Complex events are all the events that are the result of an event processing step that combines several events or enriches the events with context information. We distinguish between composite and derived events.

Composite events are the result of combining multiple events, typically through the operations of an event algebra. Although this is not always accepted, the glossary of the Event Processing Technical Society (EPTS) [19] defines that composite events always must carry all the constituting events. A special case of composite events are aggregate events which are formed through applications of the standard aggregation operators, such as average, sum, max, min, count, or top-k. Aggregate events rarely carry the full set of simple events that were aggregated. Particularly in wireless sensor networks it is the goal to reduce the volume of transferred events and only the aggregation is propagated.

Derived events are events of a higher level of abstraction. For example, if we observe 3 failed login attempts we might consider this to be an attempt to penetrate the system, or 10 temperature readings that increase monotonically imply a failure of the air conditioning system. Derived events are the events one typically expects when talking about complex event processing. They can be quite abstract and may involve correlation of events with external information. For example, the sale of stock by a manager of a pharmaceutical company two days before it is publicly known that a new drug will not be approved by the FDA might be an insider trading event. The combination of events with additional information, either from external sources or from databases, is generally referred to as event enrichment or event contextualization.

Complex event processing encompasses event filtering, aggregation, composition and derivation, as well as event contextualization. Complex event processing can be accomplished with several mechanisms. Most common is the processing of streams of event objects on which windows are defined [5]. Stream processing systems apply the operators of an algebra, for example those of relational algebra, to the instances of event objects in a window. Windows can be defined based on a number of objects in the window or based on time intervals. They can also be sliding or tumbling. Sliding windows open a new window at predefined intervals. If the interval for opening a new window is the size of the window, we speak of tumbling windows.

An alternative (or complement) to stream processing is the correlation of events, e.g. based on time. This approach is typical in sensor fusion. More sophisticated event compositions are based on event algebras that typically contain operators for sequencing (ordered events), intersection (`AND`, two events occurring in any order), union (`OR`, any of two events occurring), negation (an event `NOT` occurring in a well-defined interval), accumulation (`ANY n` events occurring in a well-defined interval), etc. [10,6]. Complex events are then represented as expressions of the event algebra. These expressions are transformed into a tree structure in which the inner nodes are the operators of the algebra and the operands (event objects) are at the leaves. These graphs are evaluated similarly to query graphs in a database from the leaves to the root with the composite event being the result of the evaluation of the root node.

2.2 Events in BPM

Usage of events is supported in mainstream business process models such as BPMN and UML activity diagrams by means of two workflow patterns: the "deferred choice" pattern [25] and the "event-based task trigger" pattern [26]. These offer an activity-based view of events; events are sent as messages to trigger subprocesses. The producer of the event sends a single event as a direct message to activate the subprocess. In this sense it is comparable with imperative programming. The notion of streams of events, subscriptions and composition/derivation of events is not considered yet.

3 Event Driven Architecture

A basic tenet of EDA is the loose coupling between event producers and event consumers. This means that the producer of events need not be aware of who will eventually consume the produced events. It also means that producer and consumer of events should be decoupled in space and time. Spatial decoupling results in distributed systems, temporal decoupling in asynchronous systems. The main properties an EDA should fulfill as stated in [7] are:

- Reporting of current events as they happen
- Pushing notifications of events from the producer to the consumer
- Responding immediately to recognized events
- Communicating one-way without the need for acknowledgements
- Reacting to event notifications and not to commands

Events should be reported as soon as they happen rather than being stored and later forwarded or requested by the consumer. Events will be reported as discrete event objects that are packaged in a notification. A notification service is thus responsible for prompt delivery of notifications. Consumers of events, i.e. the applications, should respond immediately to relevant events. These three conditions guarantee timely response of event driven systems.

Decoupling is important since events occur independently of the reactions. Interested parties must subscribe to events in order to receive the corresponding event notifications. Subscribers can also unsubscribe and this does not affect the future detection of the events, only their notification. The event based interaction pattern does not require any answer, i.e., the event producer will not block. Because the event producer is not aware of the event consumers, it cannot request the event consumer to execute any actions. Event driven systems are, therefore, consumer controlled. Loose coupling provides the desired flexibility because components need not be active at the same time and new components can be added without affecting existing components as long as the notifications do not change.

3.1 Components of an Event Driven Architecture

An EDA minimally consists of three components: event producer, notification mechanism, and event consumer, as shown in Figure 2. Event objects are accepted by the notification mechanism and packaged into notifications.

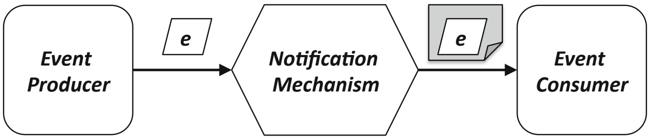


Fig. 2. Components of an EDA

Event producers detect events and produce event objects. The event object has a structure that is defined by the event type and contains the necessary event parameters. Event parameters are instantiated by the event detection process and the event contextualization process. The event detection process typically will probe the environment. The event contextualization process will add context information, such as location of the detector and timestamp but may rely on external data sources as shown in Figure 3, although for practical reasons type and context information may be held locally.

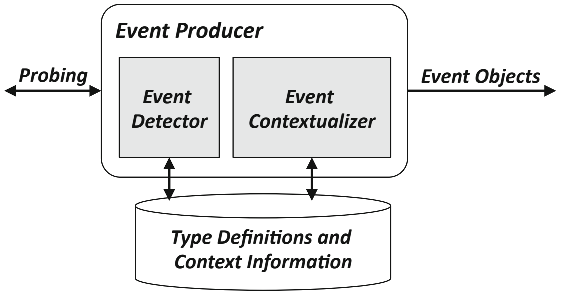


Fig. 3. Event Producer

Event consumers receive event notifications from the notification mechanism. Event consumers must unpack the event notification, extract the event object and execute an action in response to the received event. In principle, the response may be a local action, the invocation of a (remote) service or business process, a rule that must be triggered, an event composition or storage of the event for logging. Event consumers may act as event producers, for example, when they produce a composite event that is forwarded. Figure 4 shows schematically an event consumer.

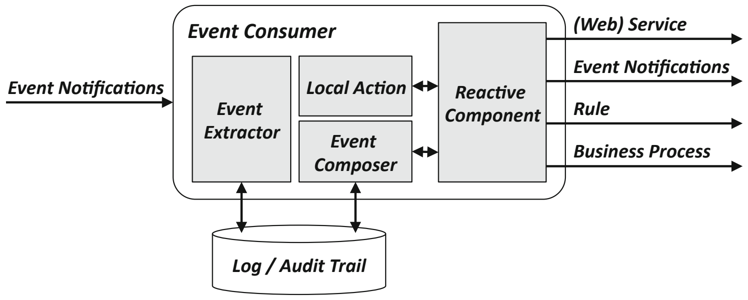


Fig. 4. Event Consumer

The notification mechanism is the most interesting component of the whole EDA. Its function is that of a communication channel that pushes events from the producers to the consumers thereby providing an end-to-end push-style communication. Because in an event driven interaction the producer does not know the consumer, the notification mechanism must mediate the communication. In its simplest form this could be a dedicated communication channel carrying all the events of a producer to the consumer, or it could be a sophisticated publish/subscribe system. We will analyze the spectrum of options. The relevant questions are:

– How are producers and consumers brought together?
– Does the channel deliver all messages or does it filter?
– If filtering is done, on what criteria and where are the filters placed?
– Are events transformed or only routed by the notification mechanism?
– If transformations are applied, where are they applied and what are they?

3.2 Channel-Based Notification Systems

Common Area Channels of two kinds are popular: blackboards and queues. In a blackboard-type channel, publishers post their detected events to a common area and consumers pick up events from there. Examples of blackboard-type common areas are tuple-spaces. An extreme form of a persistent tuple-space is a relational database. Queues are message buffering structures administered by a queue manager. Queues come in a variety of flavors: persistent vs. non-persistent, transactional vs. non-transactional. Common-area channels provide asynchrony and loose coupling but do not fulfill the requirement of end-to-end push-style notification required for EDA, since consumers must pull events from the common area. Therefore, we will not pursue common-area channels further in this discussion.

Notification Routing Channels contain one or more brokers that implement some form of routing table. Routing tables are built based on event subscriptions. Consumers declare their interest in certain (types of) events and the brokers mediate between producers and consumers. If no filtering occurs in the broker, we

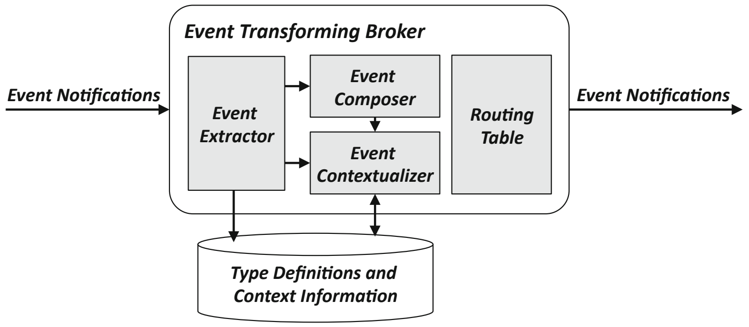


Fig. 5. Event Transforming Broker

speak about flooding. The power of notification routing channels, however, lies in the ability to filter event notifications and deliver only the relevant notifications to the consumer.

Notification Transforming Channels consist of a network of brokers, each of which can act as a consumer and producer of events. Notification transforming channels in their simplest form take in an event and change the structure of the event object and/or the format of one or more parameters. Events can also be enriched with external context data, aggregated or composed with other events. The Notification Transforming Broker receives an event notification, unpacks it, transforms the event, re-packs it into a notification and routes the new notification according to its routing table. This type of broker is shown schematically in Figure 5. Event transforming brokers are used in some advanced types of publish/subscribe systems and Enterprise Service Busses.

3.3 Publish/Subscribe Notification Systems

Publish/Subscribe is the mechanism of choice in Event Driven Architectures. Although it is possible to implement an EDA with other notification channels, Pub/Sub offers many advantages. In a Pub/Sub system, consumers subscribe to events of interest. Subscriptions are mapped to routing tables and to filters. Optionally, event producers can advertise the types of events they are prepared to produce. Depending on the filtering and the kind of routing performed by the brokers, we can distinguish different classes of publish/subscribe systems:

Channel-based Pub/Sub provides a named channel to which subscribers can subscribe. All the events of a given type are dumped into the channel. The subscriber receives all the notifications published to this channel and must apply the filters. This is the approach used in early middleware platforms, e.g. CORBA Event Service [22]. The CORBA Notification Service [23] improves the Event Service by providing filters.

Type-based Pub/Sub uses path expressions and subtype inclusion tests to select notifications. Through multiple inheritance, the subject tree can thus be converted into a type lattice with multiple rooted paths to the same node. This approach circumvents some of the limitations of simple subject hierarchies [24].

Topic-based Pub/Sub is the approach associated with the Java Messaging Service [27]. Topic-based pub/sub uses channels that include filters. Filtering is done through Boolean predicates defined on the envelope of the notification. While the filters that can be expressed are quite flexible their expressiveness is limited by the fact that they can only be defined over the metadata contained in the header.

Content-based Pub/Sub extends matching to the content of the body of the message rather than limiting it to the header. The expressiveness of this approach is determined to a large extent by the data model used for the content and the corresponding formalism for expressing the filter predicates. Systems have been proposed based on simple template matching [9], filter expressions on name/value pairs [1] and XPath expressions on XML documents [11]. A tacit assumption for content-based pub/sub to work is the existence of a common name space and context used by publishers and subscribers. This is often the case in small systems but rarely in large, heterogeneous environments.

Concept-based Pub/Sub [8,12] addresses the problem of implicitly assumed semantics and makes the semantics of advertisements, subscriptions and notifications explicit. Concept-based pub/sub associates a context with each notification and filter. Matching compares first if the contexts of filter and notification are the same. If true, the normal content-based pub/sub approach kicks in. If the contexts of filter and notification are different, an ontology service is invoked to resolve the different semantics, for example by converting currencies or formats of dates. This approach can be used in conjunction with many of the other pub/sub approaches in large, heterogeneous environments.

3.4 Hybrid Architectures

Real-life systems rarely conform to the pure reference architecture. Therefore, it is necessary to combine different invocation styles in the form of hybrid architectures. Referring to Figure 4, the event consumer may trigger a request-driven interaction to a business process or a service in a SOA, resulting in an event driven SOA. Likewise, an event may trigger a request to a queue to pick up something posted there. Both architectural styles must coexist.

4 Quality of Service in Event Driven Architectures

End to end Quality of Service (QoS) in an EDA can be affected by various components. Large event driven systems are often distributed, so the network characteristics play a big role. For example, messages may be delayed, lost or arrive out of order. This must be considered when looking at the achievable QoS of the components of an EDA. We will consider four aspects in this section: the QoS of the stream processing mechanism, the QoS of the algebra-based event composition mechanism, the QoS provided by the notification mechanism, and the effect on transactional behavior of business (sub)processes if these are triggered by an event.

4.1 QoS of Stream Processing

Stream processing is used to detect patterns and raise alarms or trigger rules or actions when certain patterns are observed. Stream processing is highly dependent on the nature of the streams. While the operators used in stream processing resemble the operators of relational algebra but applied to windows (when the event objects are tuples), the nature of streams is quite different from the nature of data in a database. Event objects in a stream arrive continuously and are usually processed in arrival order. They are generated by external sources that are outside the control of the stream processing system. Therefore, the input characteristics can't be controlled and event objects may be lost or corrupted. The volume of incoming streams may vary widely depending on the subscription pattern of the application, and the structure of the event objects may range from simple tuples to XML documents or even unstructured data, such as images.

Stream processing requires continuous processing of incoming event objects. Since applications depending on stream processing typically have timing constraints to meet, the timeliness of a response is one of the most relevant QoS requirements. Many applications that depend on stream processing can tolerate approximate results. Therefore, it is common to trade-off accuracy for timeliness. A closely related QoS metric is the achievable throughput. Throughput is less sensitive to load than response time.

Stream processing systems attempt to optimize continuous query execution to maximize throughput. However, load shedding is often the only practical approach, resulting in a trade-off of accuracy for timeliness. Application processes must be aware of this and specifications of expected timeliness and tolerable accuracy are required during business process design.

Another issue that is application dependent is the tolerance to events being processed out of order. Research on how to deal with out of order events in streams [4] has been incorporated into some of the industrial offerings.

4.2 QoS of Event Composition

The achievable QoS in event composition depends largely on the possibility to establish an ordering between events. While operators such as intersection and union do not require ordering, the sequence operator, which is part of most event algebras, requires an ordering of events. The natural ordering is done on time. This is perfectly fine if there is only one central clock and at most one event can occur per clock tick. As soon as multiple events can occur simultaneously and are time-stamped by different clocks it becomes impossible to establish a total order.

The granularity of time is also important when trying to establish an ordering. Two events with distinguishable order with timestamps of fine granularity (e.g.

milliseconds) may not be distinguishable with coarser timestamps (e.g. seconds). Some applications are not affected, while for others it is essential. For example, if a tagged container passes two RFID readers, the proper sequence determines the derived event that the container has either entered or left the warehouse. It is often the case that application semantics or additional information sources must be brought in to resolve ambiguities. However, in case of ambiguity, the underlying middleware should never arbitrarily choose a solution and pretend there exists an unambiguous ordering.

The delay or loss of messages, especially in wide area networks, is another source of potential ambiguity. To evaluate the negation operator, i.e. to determine whether an event did not occur in a given interval, one must be able to establish that the message with the notification is neither lost nor delayed. In networks with bounded delay, the 2g precedence model is adequate [15]. It establishes that anything outside an interval formed by one maximum delay before to one maximum delay after a given point in time can be known with certainty. For unbounded networks, such as the Internet, an approach based on sweeper events has been proposed [16]. It does not assume ordering, but requires only that two events in the same channel do not overtake each other. By injecting the heartbeat events from an outside time service, the recipient knows that everything coming over that channel after the heartbeat must be younger. Delivery in publishing order can be ensured by the messaging middleware. The past before the heartbeat thus becomes certain while the past between the heartbeat's timestamp and the present is still uncertain.

The last issue impacting the QoS of the event composition is the order in which events are consumed. Event expressions are written based on event types. Expressions are instantiated by the arrival of instances of events that are part of an expression. If we do not specify in what order the events should be consumed, we can't have clear semantics. For example, the expression `E AND C` with an event stream `e1`, `e2`, `c1` would consume `e1 AND c1` under chronological consumption, but `e2 AND c1` if we use the most current instances of an event type. A good solution to this problem was given in [6], but the domain expert must decide what semantics fit the application. It is equally important that the event composition software offers the right choices.

4.3 QoS of the Notification Mechanism

The notification mechanism is essential to disseminate event notifications in an asynchronous and decoupled way. Event driven business process components subscribe to events and a single event notification can trigger or change the execution of a business process. It is thus necessary to make business process components aware of the QoS of the underlying notification mechanisms. Components that rely on events should therefore be able to express QoS demands. Different QoS properties are adopted in current notification middleware, e.g., in JMS brokers.

- *Persistence*: The middleware takes extra care to ensure that no event notifications are lost in case of a server crash by buffering them on persistent storage.
- *Delivery Mode*: The delivery mode determines whether events are delivered at least once, at most once, or exactly once.
- *Durability*: With non-durable subscriptions a subscriber will only receive notifications that are published while he is active. With durable subscriptions notifications are buffered in case subscribers temporarily disconnect.
- *Transactions*: A notification session can be transactional or non-transactional. A transaction is a set of notification operations that is executed as an atomic unit of work, e.g., send all or discard all notifications in a session.
- *Order*: When order of event notifications is guaranteed, the middleware ensures that notifications arrive in the order they were published.
- *Performance*: The number of event notifications that can be handled by the middleware in time (throughput and latency).

A more detailed discussion of quality of service in Publish/Subscribe systems is presented in [2].

4.4 QoS of Transaction Management

Business processes often require transactional behavior. However, transactions come in many different flavors. Database transactions are tightly coupled and guarantee full ACID properties (atomicity, consistency, isolation and durability). This is possible because the DBMS has full control over (synchronous) communication, execution, storage, and release of results. In object transactions, the components communicate directly with each other 1:1 and communication is reference based, i.e., each component knows its counterpart and how to address it directly. Interaction requires communicating components to be present at the same time and the requestor blocks while the other component answers. This ought to be compared to a mediated communication based on publish/subscribe, where n producers communicate with m consumers, the addressing is not reference based but logical, e.g. content-based, and asynchronous. If transactions are to be executed successfully when producers and consumers of notifications are completely decoupled by the middleware, the middleware must be incorporated into the transaction.

This is the approach originally purposed by Middleware Mediated Transactions (MMT) [18,17]. The key to this proposal is to incorporate the sending and receiving of notifications into the transactional boundaries of the producer and/or the consumer of the notification. This, together with a controlled delivery mode by the messaging middleware as described above, defines a very flexible and powerful transaction model for event driven systems.

The key properties of MMTs are grouped by so-called coupling modes that reflect the visibility rules, commit and abort dependencies of complex transaction models [3].

- *Visibility* refers to when a notification is sent to consumers relative to the completion of the producer's transaction: with immediate visibility, notifications are sent to the middleware and on to consumers before the producer's transaction commits. On commit (abort), notifications are sent out only after a commit (abort) of the producing transaction. Deferred visibility means that notifications are propagated when the producer begins the commit process.
- *Context* allows the recipient of a notification to join the same transaction context as the producer (shared context) or the middleware or the consumer may establish a separate context for the recipient (separate context).
- *Forward dependency* limits the freedom of the consumer of a notification to commit. A forward commit dependency means that the consumer of a notification may only commit if the producer commits. Likewise, a forward abort dependency states that the consumer can only commit if the producer aborts.
- *Backward dependency* limits the freedom of the producer of an event notification to commit. If vitally coupled, the producer may only commit if the consumer transaction committed. A marked-rollback producer may complete but may be rolled back on request of the consumer.
- *Production* of a notification in transactional mode limits the delivery of the event notification to the mediating middleware to after the commit of the producer. An independent production policy leaves the decision to the producing transaction how a failure in delivery should be handled.
- *Consumption* refers to when the event notification is considered to have been delivered. It could be either on delivery, or when the recipient returns from executing its reaction, or when the consumer begins commit.

QoS of event driven systems is still a wide open area. We do not advocate a specific solution for stream processing, event composition, notification or a transaction model for EDA. Instead, we raise awareness of the issues that must be addressed jointly by researchers, product vendors and domain experts.

5 Summary and Conclusions

An increasing number of sensors and other sources are generating streams of valuable information that business processes should exploit. To support this, process models and the formalisms used for their description need to be expanded to represent more powerful notions of events and their integration as first class citizens in the specification and design of business processes.

We made a case for hybrid architectures combining a Service Oriented Architecture with an Event Driven Architecture. Both architectural styles are needed to satisfy the requirements of modern process oriented enterprises.

Domain experts should exploit the benefits of event processing to improve timeliness and agility. At the same time they must be aware of limitations and potential pitfalls.

End to end QoS is important for business processes. We identified four aspects that can affect the end to end QoS in an Event Driven Architecture. Additional elements are required in the modeling formalisms for the specification of quality of service expected by business processes and the acceptable trade-offs.

Acknowledgements. We wish to acknowledge many interesting discussions with Mani Chandy, Dimitrios Georgakopoulos, Annika Hinze and Christoph Liebig. Research underlying the present position paper was sponsored by German Federal Ministry of Education and Research (BMBF) under research grants ADiWa (01IS09020D) and Software-Cluster EMERGENT (01IC10S01) as well as by Hessen Ministry of Higher Education, Research and the Arts under research grant LOEWE Dynamo PLV. The authors assume responsibility for the content.

References

1. Antollini, J., Antollini, M., Guerrero, P., Cilia, M.: Extending Rebeca to Support Concept-Based Addressing. In: ASIS 2004 (2004)
2. Behnel, S., Fiege, L., Mühl, G.: On Quality-of-Service and Publish/Subscribe. In: DEBS 2006 (2006)
3. Buchmann, A.P., Ozsu, M.T., Hornick, M., Georgakopoulos, D., Manola, F.A.: A Transaction Model for Active Distributed Object Systems. In: Database Transaction Models for Advanced Applications, pp. 123–158. Morgan-Kaufmann (1992)
4. Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Monitoring Streams - A New Class of Data Management Applications. In: VLDB 2002. Morgan Kaufmann (2002)
5. Chakravarthy, S., Jiang, Q.: Stream Data Processing: A Quality of Service Perspective. Springer (2009)
6. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.-K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: VLDB 1994 (1994)
7. Chandy, K.M., Schulte, W.R.: Event Processing: Designing IT Systems for Agile Companies. McGraw-Hill, Inc. (2010)
8. Cilia, M.A., Bornhövd, C., Buchmann, A.: CREAM: An Infrastructure for Distributed, Heterogeneous Event-Based Applications. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS/DOA/ODBASE 2003. LNCS, vol. 2888, pp. 482–502. Springer, Heidelberg (2003)
9. Cugola, G., Di Nitto, E., Fuggetta, A.: The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. IEEE Transactions on Software Engineering (TSE) 27, 827–850 (2001)
10. Dayal, U., Buchmann, A.P., McCarthy, D.R.: Rules are Objects Too: A Knowledge Model for an Active, Object-Oriented Database System. In: Dittrich, K.R. (ed.) OODBS 1988. LNCS, vol. 334, pp. 129–143. Springer, Heidelberg (1988)
11. Diao, Y., Franklin, M.J.: XML Publish/Subscribe. In: Encyclopedia of Database Systems, pp. 3608–3613 (2009)
12. Freudenreich, T., Appel, S., Frischbier, S., Buchmann, A.: ACTrESS - Automatic Context Transformation in Event-based Software Systems. In: DEBS 2012 (2012)
13. Frischbier, S., Gesmann, M., Mayer, D., Roth, A., Webel, C.: Emergence as Competitive Advantage - Engineering Tomorrow's Enterprise Software Systems. In: ICEIS 2012 (2012)

14. Hinze, A., Sachs, K., Buchmann, A.: Event-Based Applications and Enabling Technologies. In: DEBS 2009 (2009)
15. Kopetz, H.: Sparse Time versus Dense Time in Distributed Real-Time Systems. In: ICDCS 1992 (1992)
16. Liebig, C., Cilia, M., Buchmann, A.: Event Composition in Time-dependent Distributed Systems. In: CoopIS 1999 (1999)
17. Liebig, C., Malva, M., Buchman, A.: Integrating Notifications and Transactions: Concepts and X^2TS Prototype. In: Emmerich, W., Tai, S. (eds.) EDO 2000. LNCS, vol. 1999, pp. 194–214. Springer, Heidelberg (2001)
18. Liebig, C., Tai, S.: Middleware mediated transactions. In: Blair, G., Schmidt, D., Takizawa, M. (eds.) DOA 2001. IEEE Computer Society (September 2001)
19. Luckham, D., Schulte, R., Adkins, J., Bizarro, P., Jacobsen, H.-A., Mavashev, A., Michelson, B.M., Niblett, P., Tucker, D.: Event processing glossary (2011)
20. Mann, S.: ebizQ (2012), http://www.ebizq.net/topics/int_sbp/features/13366.html
21. Mietzner, R., Fehling, C., Karastoyanova, D., Leymann, F.: Combining Horizontal and Vertical Composition of Services. In: SOCA 2010 (2010)
22. OMG. CORBA Event Service (2004), http://www.omg.org/spec/EVNT/1.2/PDF/
23. OMG. CORBA Notification Service (2004), http://www.omg.org/spec/NOT/1.1/
24. Pietzuch, P., Bacon, J.: Hermes: A distributed event-based middleware architecture. In: ICDCSW 2002 (2002)
25. Russell, N., ter Hofstede, A.H.M., Mulyar, N.: Workflow ControlFlow Patterns: A Revised View. Technical report (2006)
26. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Data Patterns: Identification, Representation and Tool Support. In: Delcambre, L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 353–368. Springer, Heidelberg (2005)
27. Sun Microsystems, Inc. Java Message Service (JMS) Specification - Ver. 1.1 (2002)
28. Weske, M., van der Aalst, W.M.P., Verbeek, H.M.W.: Advances in Business Process Management. Data Knowl. Eng. 50(1), 1–8 (2004)

Tying Process Model Quality to the Modeling Process: The Impact of Structuring, Movement, and Speed

Jan Claes[1], Irene Vanderfeesten[2], Hajo A. Reijers[2], Jakob Pinggera[3], Matthias Weidlich[4], Stefan Zugal[3], Dirk Fahland[2], Barbara Weber[3], Jan Mendling[5], and Geert Poels[1]

[1] Ghent University, Belgium
{jan.claes,geert.poels}@ugent.be
[2] Eindhoven University of Technology, The Netherlands
{i.t.p.vanderfeesten,h.a.reijers,d.fahland}@tue.nl
[3] University of Innsbruck, Austria
{jakob.pinggera,stefan.zugal,barbara.weber}@uibk.ac.at
[4] Technion - Israel Institute of Technology, Israel
weidlich@tx.technion.ac.il
[5] Wirtschaftsuniversität Wien, Austria
jan.mendling@wu.ac.at

Abstract. In an investigation into the *process of process modeling*, we examined how modeling behavior relates to the quality of the process model that emerges from that. Specifically, we considered whether (i) a modeler's structured modeling style, (ii) the frequency of moving existing objects over the modeling canvas, and (iii) the overall modeling speed is in any way connected to the ease with which the resulting process model can be understood. In this paper, we describe the exploratory study to build these three conjectures, clarify the experimental set-up and infrastructure that was used to collect data, and explain the used metrics for the various concepts to test the conjectures empirically. We discuss various implications for research and practice from the conjectures, all of which were confirmed by the experiment.

Keywords: business process modeling, process model quality, empirical research, modeling process.

1 Introduction

Business process modeling is utilized at an increasing scale in various companies. The fact that modeling initiatives in multinational companies have to rely on the support of dozens of modelers requires a thorough understanding of the factors that impact modeling quality [1–3]. One of the central challenges in this area is to provide modelers with efficient and effective training such that they are enabled to produce high-quality process models. There is clearly a need to offer operational guidance on how models of high quality are to be created [4, 5].

Recent research has investigated several factors and their influence on different measures of process model quality [6, 7]. In essence, this stream of research identifies both process model *complexity* and the reader's modeling *competence* as the major factors among these. While these insights are in themselves valuable, they offer few insights into how we can help process modelers to create better models right from the start. In order to give specific hints to the modeler, we have to shed light on how good process models are typically created, and in which way this creation process differs from drawing process models of lower quality.

In this paper, we look deeper into the modeling process in its relation to the creation of a high-quality process model. The research question we deal with, is whether it is possible to identify certain aspects of modeling style and model creation that relate to good modeling results. Our approach has been to leverage the Cheetah Experimental Platform [8], which allows for tracing the creation of process models on a detailed level. This permitted us to quantify the process of process modeling with respect to three different aspects. We also determined an objective measure for the quality of the resulting process models, putting the focus on the ease with which such models can be read. Based on an experiment with 103 graduate students following a process modeling course, we were able to demonstrate a strong statistical connection between three aspects of the modeling process on the one hand with our notion of model quality on the other. These findings have strong implications, as they pave the way for explicating and teaching successful modeling patterns.

The structure of the paper is as follows. Section 2 discusses cognitive concepts that are relevant for investigating the process of process modeling. In addition, we describe how the capabilities of the Cheetah Experimental Platform are conducive to document the process of process modeling in detail. Section 3 presents our research design. We explain how we developed three conjectures about process-related factors that result in better process models. Each of these three factors as well as the notion for process model quality is operationalized, such that the conjectures can be experimentally tested. Section 4 reports on the conduct and results of our experiment. We discuss the results and reflect upon the threats to their validity. The paper closes with conclusions and an outlook on future research.

2 Background on the Process of Process Modeling

In this section, we revisit findings on process model quality and the process of process modeling. Section 2.1 summarizes prior research in this area, after which Section 2.2 discusses how the process of process modeling can be analyzed.

2.1 The Process of Process Modeling and Process Model Quality

There is a wide body of literature that centers on the quality of process models, ranging from high-level, comprehensive quality frameworks (e.g., [3, 4, 9]) to a variety of metrics that pin down the quality notion in specific ways (e.g., [2, 10, 11]). Mostly, the process model is considered in these papers as a given, complete, and finished artifact. Recently, approaches are emerging that aim to connect the way that a process model has come into being with the properties of the ensuing model. In this context, various authors refer to the actual construction of a process model as *the process of process modeling* [8, 12, 13].

In general, modeling is often characterized as an iterative and highly flexible process [14, 15], dependent on the individual modeler and the modeling task at hand [16]. A central element in the further understanding of the process of process modeling is the identification of the recurring activities or common phases that comprise this process. Inspired by views on problem solving, Soffer et al. [13] distinguish between the phase in which a modeler forms a mental model of the domain and the phase in which the modeler maps the mental model to modeling constructs. The work presented in [16] is in line with this view by its explicit recognition of a *comprehension* phase and a *modeling* phase, yet extends it by the additional recognition of a *reconciliation* phase. During the latter phase, modelers may reorganize the process model at hand (e.g., rename activities) and utilize the process model's secondary notation (e.g., layout). While modeling and comprehension phases generally alternate, they may be interspersed with reconciliation actions [16]. In the same work, a so-called *modeling phase diagram* is introduced that can be used to categorize a modeler’s actions using these phases.

At this point, several preliminary insights exist that relate the modeling process with the modeling outcome, i.e., the business process model. First of all, the structure of the informal specification that is used as the basis for a process modeling effort seems to be of influence on the accuracy of the ensuing process model [17]. The reason may be that pre-structuring such a specification lowers the mental effort for modelers, resulting in a process model that better reflects the actual domain. Another insight is that the specific reasoning tools that are at the disposal to the modeler, e.g., workflow patterns vs. behavioral patterns, seem to affect the mental model that the modeler creates of a domain and, in this way, influence the semantic quality of the process model [13]. Finally, in [18] it is empirically shown that providing modelers in a distributed setting with specific model building blocks will minimize model quality issues such as variations in terminology and abstraction that individual modelers use.

The work that is presented in this paper must be seen as an attempt to extend the list of factors that can be connected to the quality of a process model, in the spirit of [13, 17, 18]. Another similarity with these works is that an empirical angle is taken to investigate conjectures about the influence of attributes of the modeling process.

2.2 Tracing the Process of Process Modeling with Cheetah Experimental Platform

The process of process modeling can be analyzed by recording editor operations as a sequence of modeling events. In this paper, we rely on Cheetah Experimental Platform[1]. This platform has been specifically designed for investigating the process of process modeling in a systematic manner [8]. In particular, the platform instruments a basic process modeling editor to record each user's interactions together with the corresponding time stamp in an event log, describing the creation of the process model step by step.

When modeling with Cheetah Experimental Platform, the platform records the sequence of adding nodes, i.e., activities, gateways and events, and edges to the process model, naming or renaming activities, and adding conditions to edges.

[1] For download and information we refer to `http://www.cheetahplatform.org`

In addition, modelers can influence the process model's secondary notation, e.g., by laying out the process model using move operations for nodes or by utilizing bend points to influence the routing of edges (see Table 1 for an overview of all recorded operations). By capturing all of the described interactions with the modeling tool, we are able to replay a recorded modeling process at any point in time without interfering with the modeler or her problem solving efforts. This allows for observing how the process model unfolds on the modeling canvas. We refer to [16] for technical details.

Table 1. Recorded events in Cheetah Experimental Platform and their classification

Create	Move	Delete
CREATE_START_EVENT	MOVE_START_EVENT	DELETE_START_EVENT
CREATE_END_EVENT	MOVE_END_EVENT	DELETE_END_EVENT
CREATE_ACTIVITY	MOVE_ACTIVITY	DELETE_ACTIVITY
CREATE_XOR	MOVE_XOR	DELETE_XOR
CREATE_AND	MOVE_AND	DELETE_AND
CREATE_EDGE	MOVE_EDGE_LABEL	DELETE_EDGE
RECONNECT_EDGE (**)	CREATE_EDGE_BENDPOINT (*)	RECONNECT_EDGE (**)
	MOVE_EDGE_BENDPOINT (*)	
	DELETE_EDGE_BENDBPOINT (*)	
Other : NAME_ACTIVITY, RENAME_ACITIVTY, NAME_EDGE, RENAME_EDGE		

(*) create, move and delete edge bendpoint were considered as actions to move an edge

(**) reconnect edge was considered as deleting and creating an edge

3 Foundations of the Experimental Design

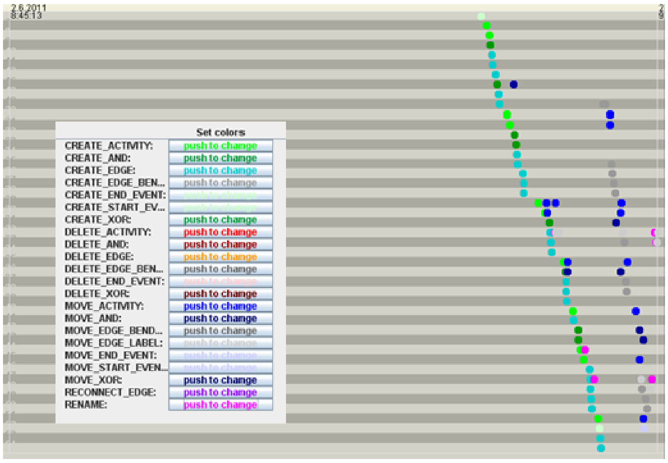
In this section we present the foundations of our experimental research design. Section 3.1 summarizes three conjectures that we derived from exploratory modeling sessions. Section 3.2 provides operational definitions for objectively measuring the process of process modeling. Section 3.3 builds an operational definition of quality for a resulting process model, which is suitable for our experimental setting.

3.1 Conjectures from Exploratory Modeling Sessions

To derive insights in the modeling process, we performed three small-scale experiments that involved 40 modelers in total. These were conducted at sites of the participating researchers throughout 2010. In these experiments modelers were asked to draw a process model on the basis of a given informal description, which was the same at all sites. We analyzed the results of these experiments by visualizing the recorded data in charts and by replaying individual modeling cases. To this end, we designed a visualization of the process of process modeling in terms of a PPMChart (Process of Process Modeling Chart)[2]. Fig. 1 is an example of such a chart. The horizontal axis represents a time interval of one hour. Vertically, each line represents

[2] We used the Dotted Chart Analysis plug-in of the process mining tool ProM for visualizing the PPMChart.

one object of the model as it was present during modeling. Each dot represents one action performed on the object; the color of the dot represents the type of action: create, move, delete or (re)name. The objects are vertically sorted by the time of the first action; the first action performed on each model object is its creation. The dots are aligned to the right such that the last action performed by the modeler is shown to occur at the end of the one hour interval. In the example in Fig. 1, we observe a short process (about 17 min) where most of the model objects were moved after creation (second dot on many lines). Furthermore, we see that the modeler has worked in 'blocks', i.e. two activities were created followed by gateways and edges. Fig. 2 shows the clear and well-structured process model resulting from the creation process.



	Set colors
CREATE_ACTIVITY:	push to change
CREATE_AND:	push to change
CREATE_EDGE:	push to change
CREATE_EDGE_BEN...	push to change
CREATE_END_EVENT:	push to change
CREATE_START_EV...	push to change
CREATE_XOR:	push to change
DELETE_ACTIVITY:	push to change
DELETE_AND:	push to change
DELETE_EDGE:	push to change
DELETE_EDGE_BEN...	push to change
DELETE_END_EVENT:	push to change
DELETE_XOR:	push to change
MOVE_ACTIVITY:	push to change
MOVE_AND:	push to change
MOVE_EDGE_BEND...	push to change
MOVE_EDGE_LABEL:	push to change
MOVE_END_EVENT:	push to change
MOVE_START_EVEN...	push to change
MOVE_XOR:	push to change
RECONNECT_EDGE:	push to change
RENAME:	push to change

Fig. 1. Visualization of the operations in the creation of one model by one modeler[3]

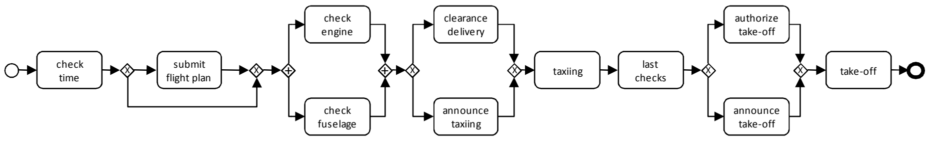


Fig. 2. Process model as result of the modeling process in Fig. 1

The interesting point of our exploratory session was the variation that we could observe in the PPMCharts. Fig. 3 shows different examples: Fig. 3a shows a process where objects were barely touched after creation, while Fig. 3b depicts a process with more actions, but mostly not long after the creation of the touched object. Fig. 3c shows a process where move actions occurred after creation of *all* objects. Fig. 3d visualizes a process with a rather chaotic actions pattern. Note that each PPMChart in Fig. 3 visualizes the creation of a process model based on the *same* textual description. It can clearly be observed, therefore, that some modelers create more elements, take more time to create their model, or move around objects on the canvas more frequently than other modelers.

[3] High resolution graphs are available from
`http://bpm.q-e.at/paper/ModelQuality.`

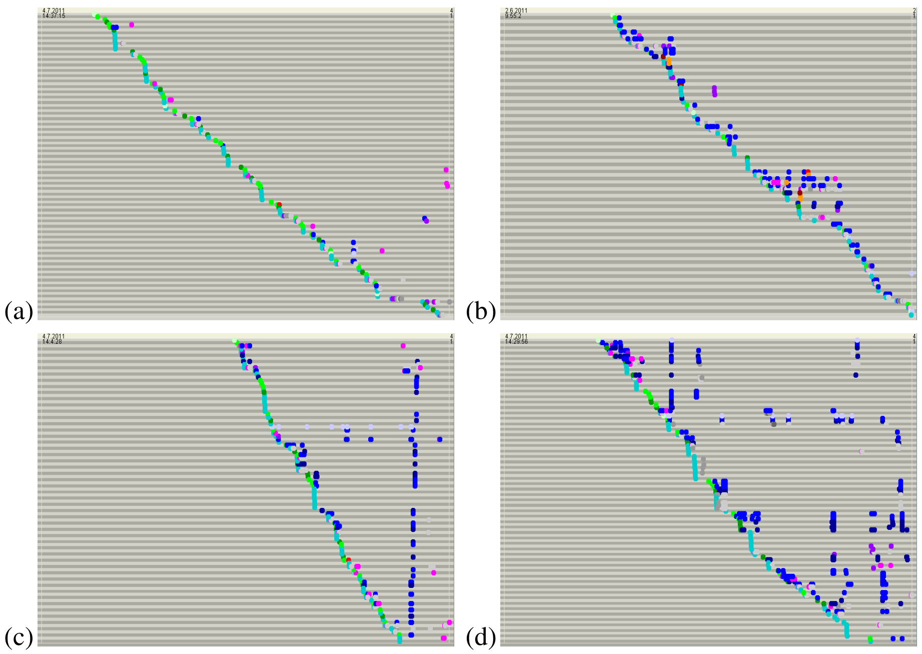


Fig. 3. More examples of the visualization of the operations in the process of process modeling

The utilization of PPMCharts helped us to identify patterns of modeling and connections between the process of process modeling and the quality of the resulting process models. More specifically, we found three conjectures:

Conjecture 1: *Structured modeling is positively related with the understandability of the resulting model.*

The conjecture is related to the limited amount of items that humans can hold in their *working memory* [19]. Cognitive Load Theory suggests that problems arise when one’s working memory is overloaded [20]. We therefore surmise that working on the complete model at once will make overloading of the working memory more likely, as compared to working on calculable pieces of the model, one at a time. Conjecture 1 defines this style of working as *structured modeling*. In other words, we assume that focusing on a specific, bounded part of the model (e.g., a block as apparent in the modeling process in Fig. 1) and finishing it before starting to work on another such part will help to reduce one’s cognitive load. Hence, this style will result in better models.

Conjecture 2: *A high number of move operations is negatively related to the understandability of the resulting model.*

While studying the results of our exploratory experiments, we observed a notable difference in the structure of the modeling process across modelers. The data of the sessions suggest that modelers who frequently move model elements seem to have no clear idea in mind of how the process is supposed to be modeled. They will therefore potentially make more mistakes, which results models of lower quality.

Conjecture 3: *Slow modeling is negatively related to the understandability of the resulting model.*

Finally, we noticed a difference in the modeling speed of different modelers (i.e., in terms of the total time between the first and last recorded modeling actions). Presumably, modelers who are in doubt about the structure of the process or about the way to capture it, will spend more time thinking about the process, trying out different strategies to organize and re-organize the model. This will ultimately take more time to finalize the process model. We presume that the more time it takes the modeler to create the model, the lower the quality of the resulting model will be. Such an effect would be congruent with the result that faster programmers tend to deliver code with fewer defects than median or below-median performing programmers [21].

3.2 Operational Measurement of Process-Based Factors

The challenge arising for these conjectures relates to their operational definition. For **Conjecture 1**, we need to provide an operational definition for a structured style of modeling based on the notion of blocks. In this context, a *block* consists of all involved model elements in two, or more, parallel or optional paths in the model. Mostly, this will concern a structure that consists of one split gateway, some successive activities, and one join gateway to complete it. We consider the modeling process to be *structured* if the modeler is not working on more than one block at the same time. The degree of structured modeling is determined based on the replay of the modeling process as visually assessed by an expert. This assessment provides the values of two metrics for structured modeling.

MaxSimulBlock is the maximum number of blocks that were simultaneously in construction. A block was considered in construction from the time the first element was created until the time the last element was created. If a block was changed afterwards (e.g., deleting and creating an activity), it had no effect on this metric.

PercNumBlockAsAWhole is the number of blocks that were made as a whole in relation to the total number of blocks. A block was considered to be made as a whole if no other elements (except for edges) were created between the creation of the first and last created element of the block.

We observed many modelers positioning activities and gateways in a block structure while adding the edges much later. For this reason, we did not consider the edges to be part of the block when calculating these metrics. As we are interested in the timing of the *creation* of elements in a block, we did not consider changes after the original creation of a block. Therefore, only those elements that were present at the initial completion of a block (this is the point in time when its last element is added) were considered to be part of the block.

For **Conjecture 2**, we consider how many elements were moved and how many moves were performed on these elements. This was calculated by a program that determined which of the recorded actions are move actions according to the list presented in Table 1. We define the following two metrics.

AvgMoveOnMovedElements is the average amount of move operations on elements with at least one move operation.

PercNumElementsWithMoves is the number of elements with move operations in relation to the total number of elements.

For **Conjecture 3**, we also wrote a small program to calculate the time spent until the model was finished. As we observed many modelers moving lots of elements around after finishing the creation of all elements, we distinguish the time between first and last action and between first and last *create* action.

TotTime is the total time between the first and last recorded action of the modeling process.

TotCreateTime is the total time between the first and last recorded *create* action of the modeling process.

3.3 Operational Measurement of Process Model Quality

There is a wide body of literature available on quality measures for process models. In this paper process model quality is defined as the ease with which the process model can be understood. In order to objectify this notion (and automate its assessment) we consider it from the structural correctness point of view; not from the semantical point of view. Prior research has defined an extensive amount of formal, structural correctness criteria for process models [22]. In the context of our experiments, we utilized BPMN as a modeling language. The problem with existing criteria, such as soundness, is that they are not directly applicable to BPMN models because BPMN does not enforce a WF-net structure [23]. Therefore, we consider a relaxed notion of quality, namely that the resulting process model should be *perspicuous*[4]. We operationalize the definition of a perspicuous model as *"a model that is unambiguously interpretable and can be made sound with only small adaptations based on minimal assumptions on the modeler's intentions with the model"*.

To make our notion of *model quality* robust against the familiarity of a modeler with notational conventions, we translate each model to a syntactically correct BPMN model whenever the model structure strongly hints at the modeler's intentions. The resulting BPMN model is then transformed into a WF-net according to the mapping defined in [24]. For such a WF-net, we checked soundness using LoLA [25]. A BPMN model is classified as being *perspicuous* if the respective WF-net is sound; otherwise, it is classified as *non-perspicuous*. In the remainder of this section, we describe the transformation to derive a syntactically correct BPMN model that can be transformed into a WF-net based on structural characteristics. The transformation is inspired by the preprocessing discussed in [24] and applied in the presented order[5].

Handling of Start and End Events. Many modeling languages do not have specific symbols for the start or end of the process (e.g., Petri-nets and EPCs). Modelers who are not aware of these specific events in BPMN may, therefore, forget to include them in their model. In line with the BPMN specification, we normalize such models:

[4] See Merriam-Webster at `http://www.webster.com/dictionary/perspicuous`.

[5] Note that these transformation rules may be generalized to any kind of modeling language.

- *Transform a process that does not have a start or an end event into a process that does, by preceding each task without incoming flows by a start event and succeeding each task without outgoing flows by an end event.* [24]

Further, some modeling languages allow for several starting points in the model (e.g., EPC, BPMN), cf. [26]. Also, it is allowed or even required that each end point in the process model is indicated separately (e.g., EPCs, COSA, BPMN). Modelers may be familiar with this explicit modeling of each start or end point, so that a WF-net structure is obtained by the following transformations:

- *Transform a process that has multiple start (end) events by replacing all start (end) events with only one start (end) event succeeded (preceded) by an XOR-split (XOR-join) gateway, and connect this gateway to each activity that was preceded (followed) by one of the original start (end) events.* [24]
- *If we determine only one origin for the multiple flows, i.e., all starting (ending) paths join in (originate from) the same gateway, we use the sign (i.e., AND or XOR) of this gateway.*

Note that the latter rule, in particular, relates to the intention of a modeler and, therefore, is specific to the notion of a model being *perspicuous*. Fig. 4 illustrates the transformations for exemplary cases.

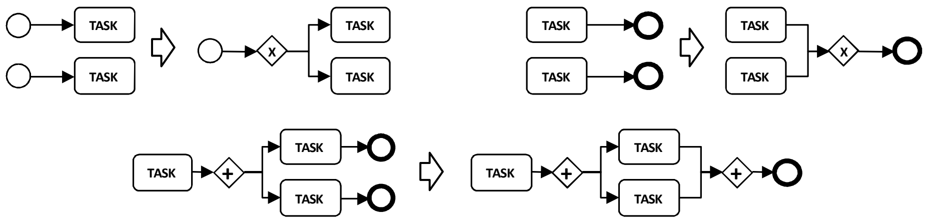


Fig. 4. Transformations related to the handling of start and event events

Split and Join Semantics. BPMN allows for modeling nodes with more than one incoming or outgoing flow. To translate the BPMN model into a WF-net, we make those split and join semantics explicit:

- *Transform multiple incoming (outgoing) flows to an event or activity into one incoming (outgoing) flow, by preceding (following) the corresponding object with an XOR-join (AND-split) gateway that has all the incoming (outgoing) flows of the object.* [24]
- *If we determine only one origin (destination) for the multiple incoming (outgoing) flows, we use the sign of this gateway.*

Again, the latter transformation relates to the modeler's intentions. We deviate from the standard processing, if the model structure provides a strong hint to do so. Fig. 5 illustrates the transformations. In the example in the lower half, none of the split gateways qualifies to induce the type of the join gateway, so that the default transformation applies.

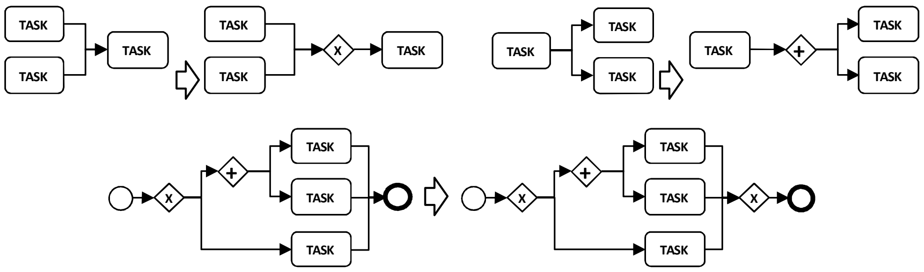


Fig. 5. Transformations related to split and join semantics

Mixed Gateways. BPMN allows for the specification of mixed gateways that combine split and join semantics. Those may be split up into a pair of a join and a split gateway of equal type [24]. However, we do not adopt this transformation for several reasons. When building the conjectures based on preliminary studies, we observed that modelers would often be unsure about semantics of mixed gateways. In contrast to the handling of start and end events and split and join semantics mentioned earlier, however, the process model structure does not provide a strong hint on the modeler's intentions regarding a mixed gateway. As such, mixed gateways lead to a non-perspicuous model. Note that those considerations are in line with the recommendation of the BPMN specification not to use mixed gateways ([27], p288).

4 Experimental Results

In this section we summarize the results of our experiment. Section 4.1 describes the experiment. Section 4.2 presents the results, while Section 4.3 provides a discussion.

4.1 Modeling Session in Eindhoven

In order to test our conjectures, we designed an experiment that would rely on the use of Cheetah Experimental Platform. The task in this experiment was to create a formal process model in BPMN from an informal description. The object that was to be modeled was the process of preparing the take-off of an aircraft[6]. We decided to use a subset of BPMN for our experiment and provided no sophisticated tool features (e.g. automated layout support or automatic syntax checkers) to prevent the modelers to become confused or overwhelmed with tool aspects [14]. A pre-test was conducted at the University of Innsbruck to ensure the usability of the tool and the understandability of the task description. This led to some minor improvements of Cheetah Experiment Platform and a few updates to the task description.

The modeling session was conducted in November 2010 with 103 students following a graduate course on Business Process Management at Eindhoven University of Technology. The modeling session started with a modeling tool tutorial, which explained the basic features of the platform. After that, the actual modeling

[6] The case description is available at:
`http://bpm.q-e.at/experiment/Pre-Flight`

task was presented according to which the students had to model the process shown in Fig. 2. By conducting the experiment during class and closely monitoring the students, we mitigated the risk of external distractions that might otherwise have affected the modeling process. No time restrictions were imposed on the students.

4.2 Results

We used the collected data of the experiment to calculate the values of the six process-based metrics of Section 3.2 for the modeling process of each student. We also determined for each modeling process the value (0 or 1) for the perspicuity metric as a measurement of process model quality. As it turned out, 54 students (52%) managed to create a perspicuous model while the remaining 49 (48%) did not.

As a next step, we looked at the distribution of the metrical values. All distributions deviated from normality, being more skewed than a characteristic Bell-curve. Therefore, we turned to the representation of these distributions as boxplots [28]. A *boxplot* (a.k.a. a *box and whisker plot*) consist of a *box*, which represents the middle 50 percent of the data. The upper boundary (also known as the *hinge*) of the box locates the 75th percentile of the data set, while the lower boundary indicates the 25th percentile. The area between these two boundaries is known as the *inter-quartile range* and this gives a useful indication of the spread of the middle 50 percent of the data. There is also a line in the box that indicates the *median* of the data (which may coincide with a box boundary) and a cross that indicates the *average value*. The *whiskers* of the box-plot are the horizontal lines that extend from the box. These indicate the minimum and maximum values in the dataset. If there are *outliers* in the data, shown as open rectangles, the whiskers extend to their maximum of 1.5 times the inter-quartile range. The boxplots for all metrics are shown in Fig. 6, 7 and 8.

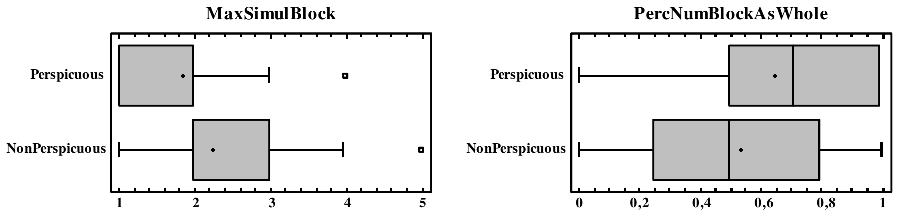


Fig. 6. Boxplots of the metrics for conjecture 1

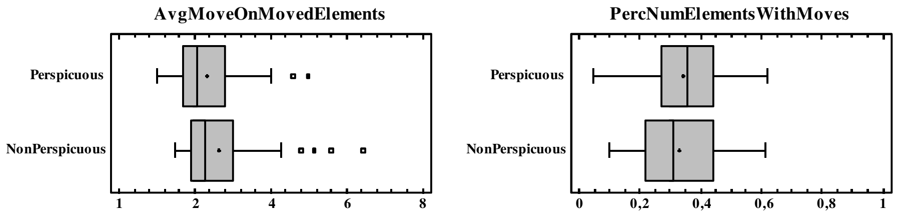


Fig. 7. Boxplots of the metrics for conjecture 2

What can be seen in Fig. 6 is that people who created perspicuous models tend to simultaneously work on a *smaller* number of blocks (*MaxSimulBlock*) than people who delivered a non-perspicuous model. Overall, those who developed a perspicuous model tend to complete a *higher* percentage of blocks as a whole too (*PercNumBlocksAsWhole).* Both aspects provide support to conjecture 1.

In Fig. 7 it can be seen that modelers of perspicuous models tend to *less frequently* move elements than the other modelers (*AvgMoveOnMovedElements*); this is in line with conjecture 2. The groups, however, do not seem to differ very much with respect to the overall *number* of elements being moved around (*PercNumElementsWithMoves*). This can be seen from the distributions that cover about the same area. So, this gives no additional support for conjecture 2.

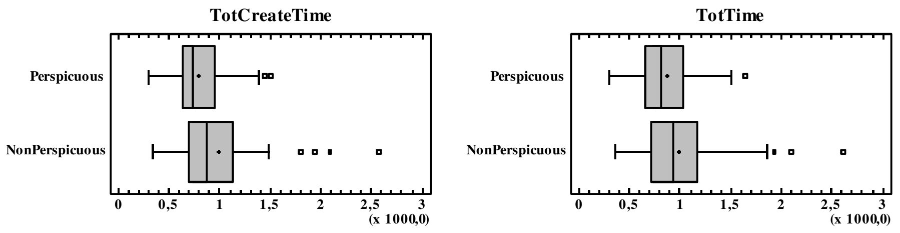


Fig. 8. Boxplots of the metrics for conjecture 3

Finally, Fig. 8 shows that the total time between the first and last recorded action of the modeling process (*TotTime*), as well as the total time between the first and last recorded *create* action of the modeling process (*TotCreateTime*), seem slightly lower for the group of modelers who created perspicuous models. It is this insight, i.e., that both distributions for modelers of perspicuous models cover a relatively lower range, that supports conjecture 3.

While these visual insights are promising, it is necessary to subject these to more rigorous testing. For this purpose, we carried out a t-test[7] for each of the six metrics in order to compare the respondents who created a perspicuous model with those who delivered a non-perspicuous model. The results are shown in Table 2.

What can be derived from these results is that there is a significant difference between the groups for all investigated metrics when assuming a 95% confidence interval (i.e., the P-values are lower than 0.05), except for *PercNumElementsWithMoves* (P-value equals 0.648 >> 0.05). In other words, the group of modelers who created a perspicuous model scored *significantly different* than the group who delivered non-perspicuous models with respect to *all our measures but one*, and in *exactly the direction* we conjectured. For example, the respondents who created a perspicuous model indeed were working on a lower maximum number of blocks simultaneously (*MaxSimulBlock*) and completed more blocks as one related whole (*PercNumBlockAsAWhole*) than the other group. From these results, we conclude that we have found strong support for conjectures 1 and 3 (i.e.,

[7] In large samples, the t-test is valid for any distribution of outcomes [32], even if we can not assume normality as is the case here.

through support for all related metrics), and mild support for conjecture 2 (i.e., via support for just one of the two related metrics).

Table 2. Results student t-test

Conjecture	Metric	T-value	df	P-value (sig.)
C1	MaxSimulBlock	-2.231	101	0.028*
	PercNumBlockAsAWhole	2.199	101	0.030*
C2	AvgMoveOnMovedElements	-1.984	101	0.049*
	PercNumElementsWithMoves	0.457	101	0.648
C3	TotTime	-2.183	101	0.031*
	TotCreateTime	-2.505	101	0.014*

(*) statistically significant values at the 95% confidence level

4.3 Discussion

Our findings warrant a reflection on their potential impact on research and practice. From a scientific point of view, our study confirms that the properties of a modeling process can be related to its outcome. Specifically, our work shows that aspects of a modeler's style can be operationalized and quantified, providing means to distinguish between more and less effective approaches to create a process model. As such, this work opens the venue towards a more sophisticated understanding of what makes someone a good modeler or, more precisely, what is a good modeling process. Values, beliefs, cognitive abilities, and personality traits may be as important in the field of process modeling as they are in the area of computer programming (see [29]). It is also noteworthy that the attractive aspect of structured modeling in particular echoes the large interest for the formal property of *structuredness* in the process modeling field [30, 31].

From a practical point of view, our findings suggest, cf. the support for conjecture 1, that an approach that emphasizes successive phases of thorough and localized modeling (i.e., within blocks) is more attractive than diverting one's attention across different parts of a model at the same time. Similarly, yet less pronounced via mild support for conjecture 2, excessive reshaping of a model and moving its elements around seem to be anathema to good modeling practice. These are both actionable items that can be shaped into modeling instructions, which can be incorporated in process modeling courses (beyond the more traditional syntactical and formal topics). Our insight with respect to modeling speed, cf. the support for conjecture 3, seems particularly relevant to distinguish more from less proficient modelers. Such an insight may be particularly useful when composing project teams (a fast modeler is an asset, both time- and quality-wise) or assigning modeling tasks to professionals (a faster modeler will deliver a readable model).

The interpretation of our findings is presented with the explicit acknowledgement of a number of limitations to our study. First of all, our respondents represented a rather homogeneous and inexperienced group. Although relative differences in experience were notable, the group is not representative for the modeling community at large. At this stage, in particular, the question can be raised whether experienced modelers follow a similar approach to process modeling as that of skillful yet inexperienced modelers. Note that we are cautiously optimistic about the usefulness of the presented insights on the basis of modeling behavior of graduate students, since we have established in previous work that such subjects perform comparably in process modeling tasks as some professional modelers [7].

We cannot claim construct validity: In our approach we derive process metrics at the syntactical level of recorded actions of a modeler and we needed to make slight assumptions on the modelers' intentions to calculate our metrics. Nevertheless, we are hopeful that we can verify the results in later experiments, because the t-tests provided significant results (except for *PercNumElementsWithMoves*).

5 Conclusion

This paper reports on research about the *process of process modeling* by examining relations between the modeling process and the modeling outcome (i.e., a process model). We have been particularly interested in the notion of understandability as a quality criterion for process models and searched for related properties of the modeling process that would ensure an *understandable* modeling result.

We formulated three conjectures, i.e., that (i) structured modeling ties to model quality, whereas (ii) lots of movement of modeling objects, and (iii) low modeling speed relate to low model quality. To validate or reject these conjectures, we performed an experiment with 103 modelers and recorded for each modeler all the actions performed with the modeling tool. This allowed us to measure the related concepts of our conjectures (i.e., structuredness, movement, speed, and understandability) in metrics on the modeling process the modeling result. T-tests point at significant differences, in line with our conjectures about the quality of the model in terms of its perspicuity. We believe this provides firm empirical support for two of our conjectures and, to a lesser extent, for the remaining one.

This paper forms a basis for a deeper understanding of the process of process modeling and its impact on the quality (*in casu* understandability) of the resulting process model. If we manage to better comprehend the factors that directly influence the result of the modeling process, we would be able to comprise this knowledge in training and tools supporting process modeling. This, in turn, could result in more understandable process models, as well as a more efficient modeling process.

In this paper, we have limited ourselves to visual inspection of the distributions and t-tests to study three conjectures. Future work will include additional statistical tests on the collected data set to identify further factors describing the process of process modeling and to assess their influence on the quality of the resulting process model. Next to a further investigation of the collected data set, we will focus on validating our observations in modeling sessions while varying the modeling task to be able to generalize our findings. We also wish to include modeling experts to be able to observe a more heterogeneous group of modelers during the act of modeling.

What is also open to further study is how effective modeling instructions can be developed on the basis of our findings. Beyond instruction, we expect that tool support may be another important ingredient in achieving good modeling practice.

Acknowledgements. We wish to thank Jan Recker for his advice with respect to the development of the research method, as well as all the modelers for their contribution to this experiment.

References

1. Rittgen, P.: Quality and perceived usefulness of process models. In: Proc. SAC 2010, pp. 65–72. ACM (2010)
2. Mendling, J.: Metrics for Process Models. LNBIP, vol. 6. Springer, Heidelberg (2008)
3. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. Eur. J. of Information Systems 15(1), 91–102 (2006)
4. Becker, J., Rosemann, M., von Uthmann, C.: Guidelines of Business Process Modeling. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 30–49. Springer, Heidelberg (2000)
5. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7PMG). Information and Software Technology 52(2), 127–136 (2010)
6. Mendling, J., Sánchez-González, L., García, F., La Rosa, M.: Thresholds for error probability measures of business process models. Journal of Systems and Software 85(5), 1188–1197 (2012)
7. Reijers, H.A., Mendling, J.: A study into the factors that influence the understandability of business process models. IEEE Transactions on Systems, Man, and Cybernetics, Part A 41(3), 449–462 (2011)
8. Pinggera, J., Zugal, S., Weber, B.: Investigating the process of process modeling with cheetah experimental platform. In: Proc. ER-POIS 2010, pp. 13–18 (2010)
9. Reijers, H.A., Mendling, J., Recker, J.C.: Business Process Quality Management. In: Handbook on Business Process Management, vol. 1, pp. 167–185. Springer (2010)
10. Gruhn, V., Laue, R.: Complexity metrics for business process models. In: Proc. ICBIS 2006, pp. 1–12 (2006)
11. Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Quality metrics for business process models. In: BPM and Workflow Handbook, pp. 179–190 (2007)
12. Indulska, M., Recker, J., Rosemann, M., Green, P.: Business Process Modeling: Current Issues and Future Challenges. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 501–514. Springer, Heidelberg (2009)
13. Soffer, P., Kaner, M., Wand, Y.: Towards Understanding the Process of Process Modeling: Theoretical and Empirical Considerations. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 357–369. Springer, Heidelberg (2012)
14. Crapo, A.W., Waisel, L.B., Wallace, W.A., Willemain, T.R.: Visualization and the process of modeling: a cognitive-theoretic view. In: Proc. ACM SIGKDD 2000, pp. 218–226 (2000)
15. Morris, W.T.: On the art of modeling. Management Science 13(12), 707–717 (1967)
16. Pinggera, J., Zugal, S., Weidlich, M., Fahland, D., Weber, B., Mendling, J., Reijers, H.A.: Tracing the Process of Process Modeling with Modeling Phase Diagrams. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 370–382. Springer, Heidelberg (2012)

17. Pinggera, J., Zugal, S., Weber, B., Fahland, D., Weidlich, M., Mendling, J., Reijers, H.A.: How the Structuring of Domain Knowledge Helps Casual Process Modelers. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 445–451. Springer, Heidelberg (2010)
18. Breuker, D., Pfeiffer, D., Becker, J.: Reducing the variations in intra-and interorganizational business process modeling–an empirical evaluation. In: Proc. WI 2009, pp. 203–212 (2009)
19. Miller, G.: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological Review 63(2), 81–97 (1956)
20. Paas, F., Tuovinen, J., Tabbers, H.: Cognitive load measurement as a means to advance cognitive load theory. Educational 38(1), 63–71 (2003)
21. Demarco, T., Lister, T.: Programmer performance and the effects of the workplace. In: Proc. ICSE 1985, pp. 268–272 (1985)
22. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, H., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing 23(3), 333–363 (2011)
23. van der Aalst, W.M.P., et al.: The application of Petri nets to workflow management. Journal of Circuits Systems and Computers 8(1), 21–66 (1998)
24. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)
25. Wolf, K.: Generating Petri Net State Spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)
26. Decker, G., Mendling, J.: Process instantiation. Data & Knowledge Engineering 68(9), 777–792 (2009)
27. OMG: Business Process Model and Notation (BPMN) version 2.0 (2011)
28. McGill, R., Tukey, J., Larsen, W.A.: Variations of box plots. The American Statistician 32(1), 12–16 (1978)
29. Cegielski, C.: What makes a good programmer? Communications of the ACM 49(10), 73–75 (2006)
30. 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)
31. Laue, R., Mendling, J.: The Impact of Structuredness on Error Probability of Process Models. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 585–590. Springer, Heidelberg (2008)
32. Lumley, T., Diehr, P., Emerson, S., Chen, L.: The importance of the normality assumption in large public health data sets. Annual Review of Public Health 23, 151–169 (2002)

Capabilities and Levels of Maturity in IT-Based Case Management

Jana Koehler[1], Joerg Hofstetter[1], and Roland Woodtly[2]

[1] Lucerne University of Applied Sciences and Arts, School of Engineering and Technology
[2] Lucerne University of Applied Sciences and Arts, School of Social Work
firstname.lastname@hslu.ch

Abstract. We present the results of a case study where we compared the needs of case managers with the capabilities of *case management software systems* (CMS) in social work, health care, and the handling of complex claims in insurance. Building on existing maturity models, we relate capabilities with maturity levels and present the C3M maturity model for IT-based case management.

Whereas vendors of *business process management suites* (BPMS) argue that case management requires flexible process guidance and improved context-sensitive information handling, we identify case assessment and case similarity as key capabilities of future CMS. We show how these and other capabilities are implemented in CMS today and discuss future trends of how CMS capabilities will evolve further. Furthermore, we discuss the impact of CMS technology on the practice of case management in an organization.

Our results are beneficial for the evaluation of CMS. They support organizations in mastering levels of maturity when using CMS, help them exploiting their benefits and addressing associated risks. The results also help BPMS vendors in adding the 'right' case-management capabilities to their BPM software when addressing case-oriented work.

1 Introduction

Case management (or case handling) has become a popular term within the business process management community as well as among vendors of BPM suites. It is commonly used to emphasize that a business process management suite (BPMS) offers flexible process support, e.g., that users of BPMS can change process flows during execution, and that multiple information sources related to a case are handled by the BPMS in a uniform and context-sensitive manner.

In its original meaning, case management denotes a specific management approach for the coordinated handling of complex situations in social work, health care, and insurance and is supported by well-established case management software systems (CMS) today. Examples of CMS that are widely used in Switzerland are CaseNet (www.diartis.ch), a web-based solution, and e-Case (infogate.ch), a rich client. One may thus wonder how trends of flexibilizing BPMS relate to the capabilities of currently existing CMS. We therefore conducted a case study during which investigated the capabilities of CMS and their usage in more detail. We present the following contributions in this paper, derived from the results of our case study:

- Building on [1], we introduce a revised four-quadrants model based on the dimensions of knowledge complexity and relationship complexity to position and distinguish case management among different types of human work.
- We present a set of capabilities for CMS that is derived from case management work in its original domains.
- We relate these capabilities to different levels of maturity in using information technology in case management and present the C3M maturity model for IT-based case management.[1]
- We summarize current trends in IT-based case management and discuss their impact on the practice of case management in an institution.

Lucerne University of Applied Sciences and Arts has currently defined four interdisciplinary focus areas that bundle a wide area of research and education activities. One of these focus areas is devoted to the future of social security and the social insurance system.[2] Social security is not only an important factor guaranteeing the stability of a society by offering support in case of disease, accident, unemployment, or poverty. The social security system also constitutes an important economic factor, e.g., in Switzerland, more than 140 Billion swiss francs or one third of the GDP result from social transfer services. All social security systems face enormous challenges due to dramatically growing expenses, demographic changes, and the need to better coordinate and align existing services as well as to develop new service models. Solutions to these challenges comprise, among many others, legal and technological aspects. In particular, information technology is considered as an important source of potential solutions, think for example of the EU research initiatives on ambient assisted living to help an aging population.

In our research project, we are investigating case management practices and how IT solutions can help improving the intra- and inter-organizational coordination and alignment of services in the area of health care, e.g., related to diagnosis-related groups, personal health management, and prevention, e.g., child welfare and protection of minors. In a first phase, we explored current trends in using IT technology for case management in selected insurances and public organizations that are partners in the case management network of Switzerland, which joins over 100 private and public organizations (see http://www.netzwerk-cm.ch). Furthermore, we assessed the state of the art of IT-based case management solutions used by our partners and explored how existing IT solutions impact their practice of case management. We also talked to vendors of case management systems and IT providers of innovative solutions used in CMS.

In particular, we address the following questions:

- What capabilities provide current IT-based case management solutions and how are they used in an organization?

[1] Note that we focus on the assessment of the maturity in using IT systems for case management, but do not assess the maturity of case management itself. The latter is completely out of our focus and would require a very different approach, see for example [2,3]. However, to the best of our knowledge, no maturity models for case management exist in literature.

[2] See `http://www.hslu.ch/hochschule-luzern/h-interdsiziplinaere-schwerpunkte.htm` (in german only).

– How do these capabilities impact an organizations's practice of case management and what benefits and risks can be observed?
– Which IT capabilities require users and what future trends can be identified?

We conducted interviews with stakeholders in the field of case management, reviewed sources from the swiss case management network, and analyzed the currently available literature. We also compared the identified capabilities as seen by users and vendors of CMS with those capabilities offered or envisioned by BPM researchers and vendors. Although we cannot claim our analysis to be representative, in particular not at a global scale, we believe that our results make interesting contributions to the future of IT-based case management as well as BPM suites.

The paper is organized as follows: Section 2 reviews the origins of case management and explores case management work in more detail. It positions case management with respect to business process work, intelligent problem-solving, and social collaboration in a four-quadrants model and discusses the applicability of the case management metaphor to other types of knowledge-intensive work. Section 3 summarizes capabilities of IT-based case management solutions and relates them to maturity levels. The C3M maturity model for IT-based case management is proposed that relates key capabilities to benefits and risks at each level. Section 4 briefly summarizes trends of how IT-based CMS will evolve in the future. Section 5 reviews related work, whereas Section 6 concludes with a brief summary.

2 What Constitutes Case Management in Its Original Meaning?

Case management practices have recently been very influential on discussions around the next evolutions of business process management. BPM authors often speak of adaptive case management [4] or case handling [5] and refer to a better support for weakly structured and knowledge-intensive processes. As the authors of [5] state, data and business goals play a much more prominent role than predefined workflows: "In case handling, the knowledge worker in charge of a particular case actively decides on how the goal of that case is reached, and the role of a case handling system is assisting rather than guiding her in doing so."

The original setting of case management as it was defined in the context of social work emphasizes additional aspects. For example, the definition of case management by the case management network of Switzerland[3] states that case management is

> *. . . a specific approach for the coordinated handling of complex situations in social work, health care and insurance. A bundle of services is provided to a client based on her/his individual needs in a systematic and cooperative process in order to effectively achieve jointly defined objectives in high quality. Case management coordinates inter-professional and inter-institutional services and respects the autonomy of the clients while preserving resources in the client's and the supporting systems.*

[3] Netzwerk Case Management Schweiz, `http://www.netzwerk-cm.ch`

Case management defines how a complex situation is handled and how the services, which respond to the needs of the client, are determined and implemented. Five phases are commonly distinguished in the client-facing processes of case management:

1. **Clearing and Intake**: Is a client in a situation in which case management can and should be applied?
2. **Assessment**: What detailed situation is the client facing? How is the case structured? What services could be of help, reaching which possible objectives?
3. **Planning**: What objectives can be jointly agreed with the client? Which services are possible and can be bundled to achieve the objectives?
4. **Linking and Monitoring**: How are the services put in place and how is the partner network established? How effective are the services?
5. **Evaluation**: Which results and change are achieved in the client's situation before she/he exits the case-management process? Are the objectives met?

Phase 1 covers the entering of a client into the case-management process. Phases 2 to 4 are highly iterative. The assessment often happens in a continuous way leading to changes in the planning and linking of the services when necessary. Figure 1 summarizes the phases.

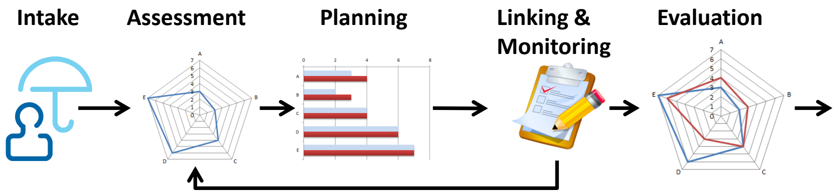


Fig. 1. The 5 phases of the case-management process

Three characteristics stand out when comparing case management to business process management: (1) the setting of objectives jointly with the client, (2) a planning phase where the case manager selects possible services, but also needs the buy-in of the client that these services can be applied and constitute a solution to the complex problems faced by the client, (3) the controlling (and revision) of service execution towards achieving the objectives.

Historically, case management has emerged as a management discipline within social work to ensure the continuity of care in the United States in the 1970/1980 years where social work and health care were extended into a coordinated end-to-end process involving different institutions and professions. Elements of case management can be found much earlier in social work, but the management discipline was coined in this decade. A core metaphor of case management is to tailor the care-giving process to the needs of the individual, i.e., the creation of a personalized instance of this process. The tailoring itself is a qualified process conducted by the case manager who needs a variety of competences to succeed. Furthermore, transparency of the tailoring is required, i.e., the result must logically follow from the assessment outcome and the defined objectives, which set the constraints for what is possible and meaningful for a client.

Case management is thus considered as a coordinated response to a differentiated landscape of offerings that can constitute a solution to a client's complex problem. It has the goal of empowering clients and also often initiates change in the resource system when necessary. Thus, in response to [4], one may argue whether a term such as *adaptive* case management is meaningful as case management is by definition a highly adaptive process.

Following [1], we use a four-quadrants model to characterize human work along two dimensions: the complexity of knowledge (KC) required to successfully accomplish a work task and the complexity of the personal and business relationships (RC) required. Four quadrants can be distinguished, which allow us to differentiate and position modern IT solutions supporting human work, cf. Figure 2.

- KC low/RC low: This quadrant comprises the domain of well-defined business processes that can be highly automated. It is the classical playground for BPM and workflow technology. The knowledge required can be clearly identified and captured in automated solutions. Relationships between human workers are well-defined and can be mapped to pre-defined roles that interact in a predefined workflow.
- KC high/RC low: This quadrant is the domain of creative and highly intelligent work, but also of intelligent systems that automate certain complex tasks within a well-defined scope. Examples are the detection of credit card fraud, complex event processing, document and internet search.
- KC low/RC high: In this quadrant, unstructured interactions between humans dominate and thus, social computing solutions prosper. Recent years have seen a tremendous development in the IT solutions that support and facilitate (but do not automate) these interactions. New trends such as *social business* emerge from the technology and show first impact on the business world by levering a thinking in terms of networks and platforms, cf. [6].
- KC high/RC high: The domain of highly qualified work combined with comprehensive collaboration needs. Workers use various IT systems, but are facing insufficient IT support today. Case management is only one metaphor to characterize this type of work. Other examples are research and development, business development and management, or project-oriented work to solve complex problems.

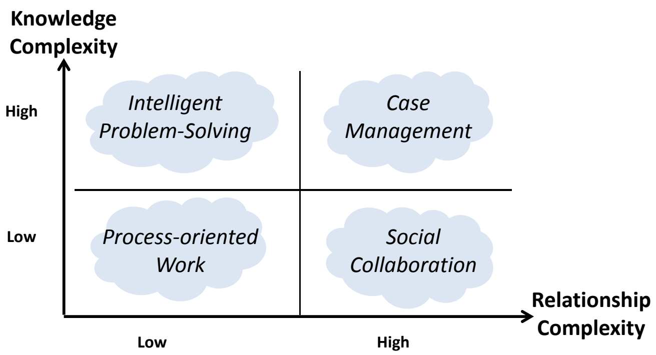


Fig. 2. Four quadrants of human work [1] (credits also to Pascal Sieber, Sieber & Partners)

Following [7] and the feedback from our interview partners, we distinguish two types of how case management work is performed:

1. *Consumer-driven CM*: Case management is considered as a clearly distinguished and complex activity that is separated from other (more structured) business processes. It is strongly client-centered and purely driven by humans, i.e., the case managers. Phases of the case-management process are reflected in a CMS (or other tools) used by case managers, however, the CMS does not drive these phases nor trigger certain tasks autonomously. Case managers work fully self-dependent and maintain the responsibility over the process. Their work style is problem- and solution-oriented.
2. *System-driven CM*: Case management is part of a larger end-to-end business process, which comprises structured and unstructured activities. The unstructured activities require a complex coordination between different stakeholders in the process as well as deep knowledge/expertise either from stakeholders or sources that are external to the process. A typical example is the investigation of complex insurance claims under suspicion of a potential insurance fraud. In this example, system-driven case work is usually state-driven, i.e., state changes in the related information objects drive the progress of the handled case (an insurance claim may be *under investigation, under dispute, settled, rejected*). In system-driven CM, the CMS plays a more active role in driving and monitoring the case work and is often either an extension of the BPMS or integrated with it.

BPM suites are currently expanding from the lower left quadrant into all other quadrants by offering for example, business intelligence, collaboration features, and recently support for case management, the latter being mostly tackled by focusing on Type 2 of our classification, i.e., system-driven case management. Dedicated CMS used by case managers, however, focus more on Type 1, i.e., consumer-driven case management. Interestingly, we found that successful CMS rarely follow a paradigm of guiding case workers through a predefined set of activities triggered by events or states of the case. They rather focus on providing customizable forms and document templates, easy recording of assessments, objectives and plans, as well as on collaboration support. Their success lies in respecting the qualified nature of the case work, i.e., preserving the autonomy of the case workers in their decisions and work organization, as well as in facilitating the planning phase of the case-management process. They offer quite different features than BPMS, which we explore in more detail in this paper.

In our analysis, we found the following characteristics of case management work to be of central importance:

– Complex assessment instruments are used and a holistic view is applied to the case situation.
– Setting objectives is a collaborative process itself and central to the success of case management due to the motivation of the client.
– Case management work comprises complex coordination, controlling and monitoring as well as assessment activities.

These characteristics are also discussed in the literature, e.g., [8], and applied to characterize knowledge-intensive work in general. It seems that the above-mentioned characteristics are typical for many modern and highly-qualified work tasks, which explains why the case management metaphor has received such interest by the BPM community. The case-management community itself feels rather uneasy when the term case management is used outside their domain, because other types of case management seem to exhibit only some of the elements from the case management definition. For case management however to succeed in the context of social work, all elements must be applied successfully and the case-management process itself must be well implemented with all its phases.

We find two aspects of case management especially interesting when applied to other types of work such as for example complex research and development projects, crisis management, etc. First, there is the personalization of the solution for the client, which is a key challenge and trend in the service economy. Companies such as IBM for example have adopted their terminology and speak of *clients* instead of *customers*. Instead of selling goods to customer, they address the needs of their clients and help them to succeed.[4] Second, the role of the business professional increases in the service economy and *information* is the key resource, see also [9]. Similar to the case manager, who is instrumental in the case-management process and balances complex needs and resources in an information-heavy process, many business professionals face corresponding challenges.

One can also gain new insights by better understanding the different roles in which a case manager acts, see also [10]:

– *Advocacy*: The case manager takes the side of the client to obtain the services the client needs.
– *Broker*: The case manager acts as a neutral intermediary between the client and the resource system to determine the optimal service bundle achieving the objectives while saving resources.
– *Gate-keeper*: The case manager assigns the available and required services to the client balancing constraints and needs in a fair way.
– *Supportive companion*: The case manager supports a client in a severe crisis situation such that the client becomes able to accept help and supporting services.

In each role, a case manager develops different information needs and handles the case-management process in a different way. Roles may also change during case management phases. This type of adaptive behavior depending on the role played by a responsible stakeholder in a process is not yet well-reflected in today's BPMS. We also would like to point out that these roles are different from the organizational roles usually considered in the design of business processes. They relate to the individual tailoring of a process and the personal positioning of a stakeholder with respect to the client, which may also vary depending on whether a stakeholder actively handles a case, assumes a management or supervising position, or acts on behalf of a regulatory authority. Summarizing, applying the case management metaphor to other areas of knowledge-intensive work seems to be well justified, nevertheless, it should be done with care and by acknowledging the many important aspects of case management.

[4] See for example, IBM client success stories at
`http://www.ibm.com/software/success`

3 Capabilities and Maturity in IT-Based Case Management

The IT landscape of CMS is currently characterized by local providers and solutions. We could not identify global players with global, but customizable products in the case management IT market. The reason lies in the high tailoring of CMS to local law and national social systems.

We begin by listing a set of capabilities required by case workers, but also offered by the IT systems used in case management. We group these capabilities by functional areas that we identified as relevant for more than one phase of the case-management process: management of the information and data belonging to a case, tracking and obtaining insights into the case history, recording and managing case-related decisions, support for collaboration among case workers and organizations, support for administrative tasks such as benefits or work-time accounting.

Table 1. Capabilities grouped by functional area and degree of IT usage

Capability	Degree of IT usage		
	low	average	advanced
Information	spreaded/paper-dominated	coordinated	integrated & consistent
- visualization	genogram, ecomap (paper)	diagrams	task-specific
- forms	simple	templates	intelligent
- access	individualized	role-based	inter-organizational
- assessment	guided	unified	standardized
Case History	spreaded across documents	tracked	visualized
- management	difficult	available	advanced insights
- insights	descriptive	diagnostic	predictive
Decisions	individually taken	systematically recorded	best practices
- case groups	none	possible	case similarity
Collaboration	disintegrated	partially integrated	seamlessly integrated
- transfer	difficult	supported	inter-organizational
Administration	separated	embedded forms	partially automated

Table 1 gives an overview over the capabilities as they take shape from lower to advanced usages of IT systems. We distinguish three levels of IT usage. “Low” means no dedicated CMS is used, but the IT support comes from other tools, e.g., office tools or database applications. “Average” stands for today’s typical CMS capabilities. “Advanced” represents innovative CMS extensions implemented or envisioned by some players in the case management market. At the end of this section, we refine the advanced level of IT usage further and arrive at five levels of maturity for IT-based case management.

Information-related Capabilities: Handling the data and information of a case is a key capability during case management. Case information is often unstructured. In particular in the context of social work, graphical representations such as genograms or ecomaps are used to capture the situation of a client. Notes taken by a case manager, emails, and interviews are predominant entities of information. When the usage of IT systems is low, many of these information entities are often recorded on paper. Others are spread across various IT tools. With the introduction of a CMS, paper-based documents are replaced by electronic solutions and information entities are better coordinated. For example, data and documents can be easily transfered between different tasks within the case-management process. Diagrams are increasingly used besides text to visualize case assessments for example. Still, duplicated information recording happens. With an advanced usage, duplicated information is eliminated and information entities are integrated and checked for consistency. Advanced visualizations are available, for example, combining case-specific data with geographic and socio-demographic information. Figure 3 shows an example of an advanced visualization that combines patient data with geo-spatial information used by an insurance company to understand how patients move between health care providers.

Forms are used at all three levels, but with the introduction of a CMS, customizable templates improve electronic forms with pre-configured elements to initiate and manage services. Intelligent forms ease information recording and analysis at the advanced level. Access to a case remains with the individual case worker at the lowest level, whereas role-based access control is introduced with the usage of a CMS. At the advanced level, information can be exchanged and coordinated between organizations with security and privacy issues being resolved, e.g., by using an information broker that provides functionality far beyond the case folder introduced by some BPMS today.

A key capability is the correct assessment of the case. At the lowest level, the quality of the assessment depends on the qualification of the responsible case worker who follows organizational guidelines. With the introduction of a CMS, assessments are unified by templates and forms encoding guidelines. At the advanced level, sophisticated standardized assessments are introduced, which encode deep insights into a case management domain and enable the multi-faceted analysis of a case.

History-related Capabilities: Tracking cases and obtaining aggregated information about a case or a case group is a major management need and often the reason why CMS are introduced into an organization. Understanding the history of a case, in particular, how effective the planning and linking worked, but also controlling and predicting its potential development, and recognizing complex cases early (including the detection of social trends), is a major challenge today. Data analysis techniques to visualize and understand the temporal progress of a case are requested by CMS users. The history of

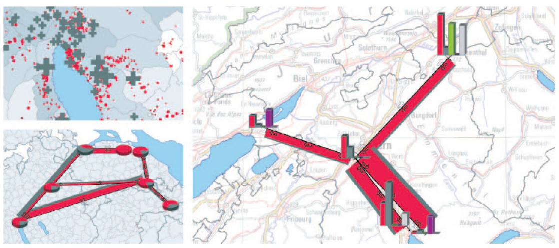


Fig. 3. Advanced case visualizations in D-Care, see www.lcc-consulting.ch

a case comprises the assessments and evaluation(s), the objectives agreed between case stakeholders, the benefits and services provided as well as their outcomes. At the lowest level, this information is spread across many documents and a unified view on the history of a case is very hard to obtain. With the introduction of a CMS, the information related to a case is managed in a more coordinated manner and the history is tracked, but problem-specific views on the history might not yet be available. Advanced levels integrate heterogeneous and unstructured information sources and provide sophisticated visualizations of the case history.

The main challenge for the case manager lies in arriving at an adequate assessment of a case and in keeping an overview of the big picture of the client situation and how it evolves. Case managers often face the problem that an improvement in one dimension causes a degradation in another. Deep human judgment is required to assess whether the overall situation has improved and which steps should be next. Sometimes, clients pose additional challenges by moving between service providers and trying to cut off a case history. For example, parents relocate to enter a challenged child into a new school, which cannot know about the history of the case due to data privacy issues.

Decision-related Capabilities: Knowledge of how a case is handled is often formulated as rules that guide or constrain the human decision maker. Some of these rules are made explicit in a case-management organization and are regulated by law, others remain implicit. Achieving and maintaining compliance of the case handling with legal regulations is a major challenge today as the complexity of cases as well as the regulation of case management is increasing. Furthermore, some, but not all of the knowledge about the case is available in the case-related documents. A significant source of knowledge also comes from observations made by the case manager and other case workers involved in a case. This knowledge is not always consistent and made explicit in the case-related documents, but plays an important role in the case-management process through the empathy and intuition applied when taking decisions. Effectiveness of decision-making is critical for the case-management process to succeed— an aspect where it often differs from business process management. Different case evolutions require different responses: different wrt. time to react, costs, coordination, benefits, experience, and qualification of the case manager. A better understanding of the state of the client, the events that happen as well as the case provider network helps in taking effective decisions.

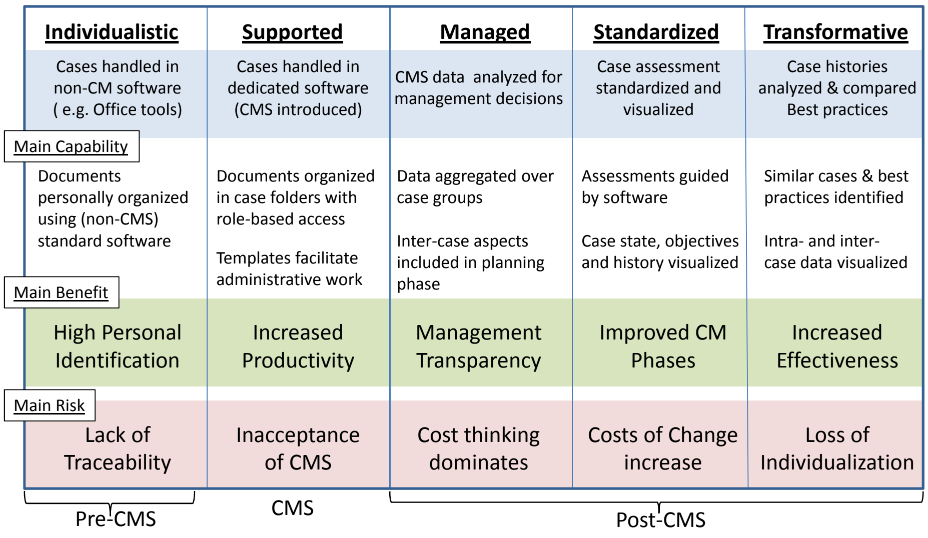
At the lowest level, decision-making is not directly supported by the IT infrastructure. In a CMS, decisions are supported by an improved view on the case situation. With advanced usages of CMS, benefit/service usage patterns can be extracted from the case data and a specific case can be compared to a representative case group. A refined understanding of case groups can help in establishing best practices, but also risks that a "one-size-fits-all" approach replaces the key paradigm of case management, namely that individualization is key to success, i.e., results do not improve, but cost savings are achievable by accepting less optimal solutions for clients.

Collaboration-related Capabilities: The longer a case lasts, the more stakeholders get involved and the more information needs to be coordinated between stakeholders. Low IT system usage hinders effective collaboration as information is scattered, must be manually transfered, and easily gets out of sync. With the introduction of a CMS, role-based access control is established, documents can be transfered using small workflows, and document exchange with office tools is made easy. Advanced solutions require to address in particular inter-organizational issues, which are mostly unresolved today. Each stakeholder of a case acts within his own law-regulated space and is not or only partially aware of the spaces of other stakeholders. Coordination and opportunity finding is therefore difficult. Furthermore, coordination needs vary for each case, which is a challenge for advanced CMS implementations.

Administration-related Capabilities: Accounting of benefits and tracking work efforts put into a case by stakeholders requires calculating forms and templates. With low IT usage, these forms exist either on paper or in separated IT systems. CMS embed and facilitate accounting. Advanced levels partially automate these tasks.

We are now ready to present our proposed C3M maturity model for IT-based case management. The model distinguishes a *pre-CMS level* and a *CMS level*, where a CMS is introduced into an organization. It refines the advanced usage of CMS into three *post-CMS levels*. In our investigations, we found that many organizations still work at the pre-CMS or CMS level. Some organizations, in particular larger insurance companies, begin to evolve their CMS systems and enter the post-CMS area. For each level, the model shows the main capability that we consider as characteristic for this level and combines this capability with two other aspects, namely the main benefit an organization can gain and the main risk it has to address.

At the *individualistic* level, the individualization paradigm on the side of the client as well as the case workers dominates. Documents are personally organized with the help of various IT systems. The main benefit is the high personal identification of the case manager with a case. The main risk lies in the lack of traceability. At the *supported*



	Individualistic	Supported	Managed	Standardized	Transformative
Main Capability	Cases handled in non-CM software (e.g. Office tools)	Cases handled in dedicated software (CMS introduced)	CMS data analyzed for management decisions	Case assessment standardized and visualized	Case histories analyzed & compared Best practices
	Documents personally organized using (non-CMS) standard software	Documents organized in case folders with role-based access Templates facilitate administrative work	Data aggregated over case groups Inter-case aspects included in planning phase	Assessments guided by software Case state, objectives and history visualized	Similar cases & best practices identified Intra- and inter-case data visualized
Main Benefit	High Personal Identification	Increased Productivity	Management Transparency	Improved CM Phases	Increased Effectiveness
Main Risk	Lack of Traceability	Inacceptance of CMS	Cost thinking dominates	Costs of Change increase	Loss of Individualization
	Pre-CMS	CMS	Post-CMS		

Fig. 4. The C3M maturity model for IT-based case management

level, a CMS is used to better organize documents and provide templates that facilitate the case work. Productivity increases, but an organization might face acceptance problems of the CMS as well as lack of management support in particular in the initial phase of technology adoption. At the *managed* level, the organization exploits the data aggregation and analytics features of the CMS as a basis for management decisions at the level of higher management, but also for the handling of the individual case. Data of one case can be compared with data of other cases. Management transparency is the main benefit, but the risk is a cost thinking that overdominates other aspects. At the *standardized* level, a unified assessment methodology is implemented in the CMS and standardized assessments are introduced. Similar assessment outcomes lead to similar objectives and measures in the subsequent phases, helping to improve the effectiveness of the case-management phases as the main benefit. Visualizations of the case state, its objectives and history are provided by the CMS and exploited during decision making. As the main risk, changing assessments and their implications becomes more costly as with any work approach that is implemented in a software system. At the *transformative* level, similarity of cases is defined including data from the case history, which enables an organization to extract best practices and feed them back into case management. This can help the organization as a whole to improve the effectiveness of its case management, but also bears the risk that the tailoring of a solution to the individual needs of a client is lost as cases are managed based on the most similar case group.

A refinement of the levels with the characteristic instances of all capabilities and a more detailed map of risks and benefits is possible, but goes far beyond the scope of this paper. We believe that this model is not only beneficial for maturity assessment, but also for IT governance purposes as it addresses the impact of IT innovations on a case-management organization. A systematic refinement of the model enables us to consider benefits and risks for all aspects of a business system. Organizations need to respond to IT developments. Their response decides whether they can built new business models upon an IT innovation and manage the associated risks in order to keep up with the competition or whether they will disappear from the market.

The impact of IT on case management is also reflected in the four-quadrants model of human work. For example, we saw tendencies that the further division of labor within case management moves some activities from the upper right quadrant into the lower left, i.e., many administrative tasks within case management receive better IT support, get partially automated and thus, become structured business processes. A combination of intelligent and social computing technologies leads to novel IT capabilities provided by CMS. For example, information visualization changes the way how case managers can look at case-relevant data. Intelligent decision support exploits insights based on case similarity and leads to decisions with more effective outcomes. We briefly discuss some of these trends in the next section.

4 Trends in IT-Based Case Management

Several trends and emerging needs have also been identified in our interviews that we briefly summarize in the following:

Adoption of Mobile Technologies: The trend to replace paper-based work and isolated legacy solutions with integrated CMS continues. Organizations still using proprietary extensions of legacy systems such as MS Access or Filemaker feel an increasing pressure to replace and modernize their IT environments. Mobile clients are clearly a need that help case managers work in different locations. Furthermore, web services or apps that extend mobile clients with specific context- or location-specific capabilities such as data access have been mentioned. Security needs increase as typical case workers are not really IT savvy, but information must be increasingly shared, including the access of clients to their personal data stored in a CMS. Furthermore, the intelligent linking of different information sources to enable the effective management of a case also leads to unresolved data privacy issues.

Improved Collaboration: The management and hosting of sensitive data is an unsolved issue, in particular when shared between different organizations. Complex cases require the delegation of tasks among case workers including a transfer of access to sensitive documents. The increased need for interdisciplinary cooperation and inter-organizational coordination requires effective solutions. In particular, the information need of involved stakeholders that join or take over a case management situation should be met with low effort and low cost, but effectively. Three major groups of users adopting different views and understandings must be addressed: the case workers, their management, and regulatory authorities. Collaboration solutions must also be seamlessly integrated with the default communication systems, which also evolve at a high pace.

Applied Business Intelligence: The visualization of case-related data will gain further importance. On the one hand, case-related data will be presented more often in graphical form than text-based. On the other hand, different data sources will be presented in an integrated way and trends in the evolution of a single case or a group of cases will be displayed graphically. There is a clear desire to obtain insights into hidden patterns and states of cases. Furthermore, CMS will help case managers to learn from previous decision histories and also illustrate potential decision options. Decision support using aggregated and extended data, for example using external sources such as household statistics, is currently built into CMS. The early recognition of cases will be partially automated and thus improve the intake phase. There is a general need to integrate analysis tools and operative process support. Standardized assessments seem to gain further importance requiring deeper skills and involving more stakeholders. Cases that require cross-organizational collaboration undergo a detailed assessment, e.g., clarifying insurance conditions, during which it is determined which organization takes the lead in handling the case. Case similarity becomes a key concept at the advanced level, but effective criteria on which to measure the similarity of cases are an unresolved research problem in most domains. “Case intelligence/insight” in general is a promising future research area to achieve better decisions. Instead of descriptively recording in a CMS what is happening, decisions should be based on understanding why it did happen and finally move to prescriptive decisions that can actively influence what will happen.

Measuring and Scheduling: In particular larger organizations have an increased need to improve the scheduling of the case management work force. Furthermore, work-time reporting (and the related billing, benefits accounting, and cost transfer) should be further automated and simplified. In general, the desire increases to better measure the costs and quality of service benefits and their effectiveness for a case. An improved root cause analysis over a group of cases has also been mentioned. Questions such as why do cases increase? in which areas? are asked more often and cannot easily be answered with today's CMS. Recurring routine activities, for example those required to administer a case, should be easy to automate.

5 Related Work

Humphrey's seminal maturity model for the software process with its five maturity levels *initial, repeatable, defined, managed, optimizing* has inspired maturity models in various areas. Its original focus, as also carried on in the famous CMMI (Capability Maturity Model Integration) is on process improvement, i.e., it provides organizations with the essential elements of effective processes, which will improve their performance. It is thus very natural that maturity levels have been defined for business processes by the BPM community as well. We build on these models as we could not find in the literature any maturity models or capabilities sets defined for case management.

One of the first BPM maturity models is defined by Fischer in 2004 [11] who considers the dimensions (levers of change) *strategy*, *controls* (governance), *process*, *people*, and *technology* and defined the following five maturity levels based on capabilities reached along each dimension: *siloed*, *tactically integrated*, *process driven*, *optimized enterprise*, *intelligent operating network*. De Bruin and Rosemann present an improved model in 2005 [12] that replaces the process dimension (which is in fact the one to be defined and should thus not be part of the input) by the dimensions of *methods* and *culture*. The five maturity levels are preserved and follow more closely the original CMMI levels: *initial*, *defined*, *repeated*, *managed*, *optimized*. In 2006, Wolf and Harmon [13] present a maturity model with slightly changed levels focusing on the degree of process organization: *unaware* (no organized processes), *opportunistic* (some processes organized), *standards* (most processes organized), *enterprise* (processes are managed), *transformative* (processes are continuously improved). Also in 2006, Gartner [14] presents a maturity model distinguishing 6 phases, which refines the standards level into two levels of intra-process and inter-process automation and control. In 2007, Hammer [15] introduces the PEMM (Process and Enterprise Maturity Model) that distinguishes four levels of process maturity based on enablers such as *design*, *performers*, *owner*, *infrastructure*, and *metrics* and combines them with four levels of enterprise-wide capabilities based on *leadership*, *culture*, *expertise*, and *governance*. PEMM does not aggregate the two groups into overall maturity levels. Its focus is more on analyzing and guiding transformation processes than on a general assessment of maturity. Finally, the OMG publishes a BPM maturity model specification in 2008 [16] with the five levels *initial*, *managed*, *standardized*, *predictable*, *innovating* and defines detailed process areas. Despite minor differences in naming or emphasis on certain aspects, all models essential share similar levels of maturity.

Usually, the models focus on the maturity levels, and less on the capability levels, which play a much more prominent role in CMMI. Capability levels apply to individual process areas and enable a continuous and incremental evolution of processes, whereas

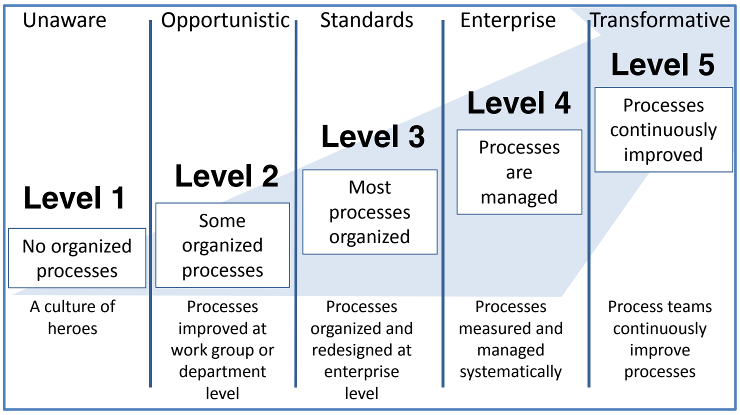

Unaware
Opportunistic
Standards
Enterprise
Transformative
Level 1
No organized processes
A culture of heroes
Level 2
Some organized processes
Processes improved at work group or department level
Level 3
Most processes organized
Processes organized and redesigned at enterprise level
Level 4
Processes are managed
Processes measured and managed systematically
Level 5
Processes continuously improved
Process teams continuously improve processes

Fig. 5. The Wolf and Harmon BPM maturity model as presented in [17]

maturity levels address entire process areas and allow an organization to advance in stages. In our work, we focus on capabilities and their support by IT. Thus in contrast to BPM and other maturity models, we focus on the degree of technology adoption by an organization. It is important to acknowledge that higher maturity levels not necessarily mean better case-management processes. Each organization must decide which maturity level in using IT leads to the best support of case-management work. Our highest maturity level corresponds to the most comprehensive and sophisticated usage of IT technology, but this is not identical with the best case-management practices. As De Bruin and Rosemann pointed out [12], "it is a case-by-case challenge to identify the most appropriate (BPM) maturity level based on context, underlying objectives, related constraints, possible business cases, etc."

Recent years seem to have seen less interest in maturity models. Measuring and comparing processes and capabilities is interesting, but not necessarily useful unless it can help guiding improvements and transformations of a business. Our maturity model thus focuses less on measurement, but more on the identification of capabilities, for example as a foundation for a detailed requirements analysis. Furthermore, we link capabilities to benefits and risks to help governing case-management-related IT decisions and manage their impact. This helps organizations assessing whether a specific capability is needed and identifying its associated benefits and risks. The model thus supports organizations in evaluating software products and it simplifies purchasing decisions. Software vendors can position their product roadmaps with respect to the model. Furthermore, the model makes explicit the impact of technology on the business.

6 Conclusion

We present a detailed characterization of case management and contrast it with the ongoing discussion of case management within the BPM community. We position case management in a four-quadrants model of human work and compare it with process-oriented work, social collaboration, and intelligent problem-solving. We derive a set of capabilities required by case workers and show how these capabilities are supported at the low, average, and advanced levels of using IT-based case management systems. We propose the C3M maturity model for IT-based case management consisting of five levels where we relate the characteristic capability of each level with the main benefit and risk of technology adoption.

References

1. Davenport, T.H.: Thinking for a Living: How to Get Better Performance and Results from Knowledge Workers. Mcgraw-Hill (2005)
2. McLellan, A.T., et al.: Does clinical case management improve outpatient addiction treatment? Drug and Alcohol Dependence 55, 91–103 (1999)
3. Ziguras, S.J., Stuart, G.W.: A meta-analysis of the effectiveness of mental health case management over 20 years. Psychatric Services 51(11), 1410–1421 (2000)
4. Swenson, K.: Mastering The Unpredictable: How Adaptive Case Management Will Revolutionize The Way That Knowledge Workers Get Things Done. Meghan-Kiffer Press (2010)
5. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering 53(2), 129–162 (2005)
6. Levin, R., Iansiti, M.: The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability. Mcgraw-Hill (2004)
7. Moxley, D.P.: Case management by design: reflections on principles and practices. Nelson-Hall Publishers (1997)
8. Kraft, F.M.: Improving knowledge work. In: Swenson, K. (ed.): Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done, pp. 181–210. Meghan-Kiffer Press (2010)
9. Fitzsimmons, J., Fitzsimmons, M.: Service Management - Operations, Strategy, Information Technology. McGraw Hill (2011)
10. Kie, T., Monzer, M.: Case Management und Soziale Dienste. In: Evers, A., Heinze, R.G. (eds.): Handbuch Soziale Dienste, pp. 499–515. VS Verlag für Sozialwissenschaften (2011)
11. Fisher, D.M.: The business process maturity model - a practical approach for identifying opportunities for optimization. BPTrends (September 2004)
12. Bruin, T.D., Rosemann, M.: Towards a business process management maturity model. In: Proc. 13th European Conference on Information Systems (ECIS) (2005)
13. Wolf, C., Harmon, P.: The state of business process management. BP Trends (June 2006)
14. Melenovsky, M.J., Sinur, J.: BPM maturity model identifies six phases for successful BPM adoption. Research Report G00142643, Gartner (2006)
15. Hammer, M.: The process audit. Harvard Business Review, 111–123 (April 2007)
16. Object Management Group: Business Process Maturity Model 1.0 (BPMM), OMG document number formal/2008-06-01 (2008)
17. Snabe, J.H., Rosenberg, A., Moller, C., Scavillo, M.: Business Process Management - the SAP Roadmap. SAP Press (2008)

Business Process Architecture: Use and Correctness

Rami-Habib Eid-Sabbagh[1], Remco Dijkman[2], and Mathias Weske[1]

[1] Hasso Plattner Institute at the University of Potsdam
{rami.eidsabbagh,mathias.weske}@hpi.uni-potsdam.de
[2] Eindhoven University of Technology, The Netherlands
r.m.dijkman@tue.nl

Abstract. Becoming more and more process oriented, companies develop collections of hundreds or even thousands of business process models that represent the complex system of cooperating entities that form an organization. Designing and analyzing the structure of this system of business process models emerges as a new challenge, which is covered by the field of business process architecture. This paper presents a formal conceptual framework for representing and analyzing business process architectures. It identifies patterns of relations between process models, and it introduces anti-patterns that represent erroneous relations between them. The conceptual framework and the patterns are evaluated using a real-world process model collection. The evaluation shows that explicitly representing and analyzing relations between process models can help improving the correctness and consistency of the business process architecture as a whole.

1 Introduction

Companies develop and maintain collections of hundreds or even thousands of business process models. These models and the business processes that they depict, represent the complex system of cooperating entities that is an entire organization. Designing and analyzing the structure of this system of business process models emerges as a new kind of problem, which is covered by the field of business process architecture.

In business process management, so far mostly individual processes were in the centre of interest, and a myriad of interesting research results regarding individual processes have been established by the BPM community. In this paper, we take a different view by studying the interrelationships between the business process models of an organization, and not the individual models themselves. We address design questions, such as:

- While I know my individual processes to be sound, are my processes also sound in their relations with each other?
- Where should one business process start and another begin and how does this affect assignment of responsibility?

- Which path does a client go through in administration offices as whole, when, for example, that client wants to open a new enterprise?

While most of the process architecture design styles presented in Dijkman et al. [1] help to answer these questions, they lack a formal definition to analyze the overall interdependencies of all processes of the system represented in the business process architecture. There are two kinds of interdependencies that can be observed in a process architecture, triggering and information flows. While both the analysis of process and service interaction across organizational boundaries and also instantiation behavior for single processes is dealt with in literature [2,3,4,5,6], the combination of both within a business process architecture has not been looked at in detail. To overcome this gap, this paper introduces a conceptual framework and structural patterns that shall help to design correct process architectures and anti-patterns to detect erroneous interdependencies between process models in process architectures.

The remainder of this paper is organized as follows Section 2 defines and formalizes the concept of business process architecture. Section 3 presents structural patterns and anti-patterns that are applied and evaluated with the SAP Reference Model collection in Section 4. Section 5 puts our work into research context followed by the conclusion in Section 6.

2 Business Process Architecture

This section defines and formalizes the concept of business process architecture and defines the behavioral contract that must be observed by processes that cooperate in the context of that business process architecture.

2.1 Business Process Architecture Syntax

A business process architecture is a collection of business processes and their interdependencies with each other. In previous work [1] we surveyed the different types of relations that have been used by business process architecture design methods, these mainly include:

- *composition*, which represents that one business process is composed of a number of other business processes, also called the sub-processes;
- *specialization*, which represents that one business process specializes another;
- *trigger*, which represents that one business process causes another business process to instantiate and start; and
- *information flow*, which represents that information or other objects flow from one business process to another.

This paper focuses on behavioral relations between business process models: trigger and information flow. We use the concepts of throwing and catching intermediate events, inspired by the BPMN, where throwing events are of active nature (like sending a message) and catching events are of passive nature (like receiving a message), since they wait for throwing events to happen. We define a business process architecture as follows.

Definition 1 (Business Process Architecture). *A business process architecture is a tuple* (E, P, T, F, M)*, in which:*

- E *is a set of events, partitioned into start events,* E^S*, end events* E^E*, intermediate throwing events* E^T *and intermediate catching events* E^C*.*
- P *is a partition of* E *and represents a set of business processes and each member represents a business process.*
- $T : E \rightarrow E$ *is the trigger relation, partitioned into synchronous triggers* T^S *and asynchronous triggers* T^A*.*
- $F : E \rightarrow E$ *is the flow relation, partitioned into synchronous flows* F^S *and asynchronous flows* F^A*.*
- $M : (E^T \cup E^C) \rightarrow \mathcal{P}(\mathbb{N}) \cup \{\{0 \ldots n\}, \{1 \ldots n\}\}$ *is a multiplicity function that defines for each intermediate event of a process instance how often it can be thrown or caught respectively.*

For an event $e \in E$, we denote the set of events $\bullet e = \{e' \in E | (e', e) \in T \vee (e', e) \in F\}$ that contains the events with an outgoing relation to e. Similarly, we denote the set of events $e\bullet = \{e' \in E | (e, e') \in T \vee (e, e') \in F\}$ that contains the events with an incoming relation from e. Note that it is not necessary for start and catching events to have incoming relations, and for end and throwing events to have outgoing relations. We also call such events *external* events. They are events that interact with factors outside of the process architecture.

Figure 1 shows a graphical representation of a sample business process architecture. The architecture has start events $s_1, \ldots, s_5 \in E^S$, end events $e_1, \ldots, e_4 \in E^E$, and intermediate events $i_1, \ldots, i_8 \in E^T \cup E^C$. It contains process models (or processes) p, p_1, p_2, q, and $r \in P$, where, for example, $p = \{s_1, e_1, i_7, i_8\}$. The architecture also has flows and triggers connecting the events of the different processes. The logic of the business process architecture depicted in Figure 1 is described in Definition 2.

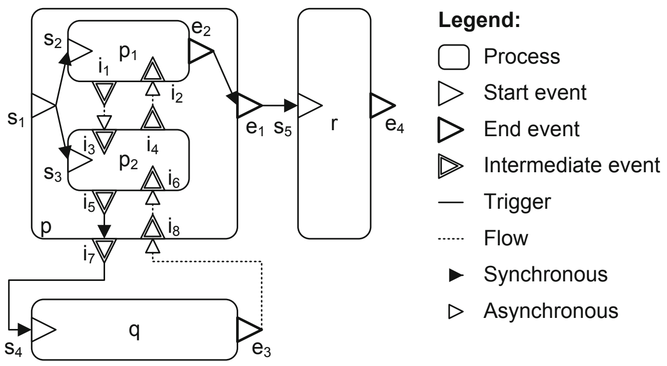


Fig. 1. Example of a business process architecture

2.2 Business Process Architecture as a Behavioral Contract

A business process architecture provides a minimum contract that all business processes that cooperate in its context must observe. The contract specifies when business processes can be instantiated, when they complete and at what time which events can throw or catch. This section defines the behavioral contract that must be observed by processes in a process architecture, but first we define the behavioral semantics of trigger and flow relations between processes.

A trigger relation from an end event $e \in E^E$ to a start event $s \in E^S$ represents that the event e triggers the event s and therewith the instantiation of the process that contains s. Note that we assume that a start event can only occur once for an instance, when it occurs again, it again creates a new instance of the process. If the triggering relation is synchronous, e and s occur at the same time; if the relation is asynchronous, the event e is buffered and can trigger s at a later time. Once a process is instantiated, eventually that instance will complete with the occurrence of one of its end events. A trigger relation from an event e to a non start event e' simply represents that e triggers e', thus passing on the trigger, but not instantiating any process.

A trigger relation of multiple events $e_1, \ldots, e_n$ with an another event e signifies that the occurrence of any one of $e_1, \ldots, e_n$ trigger event e. A trigger relation of event e with multiple events $e_1, \ldots, e_n$ signifies that the occurrence of e triggers all of $e_1, \ldots, e_n$. This behavior is motivated by the nature of events: if an event occurs, it occurs for all parties that are interested in it. For example, if the event 'order ready' occurs, this signifies that the order is ready for all parties involved, such that the occurrence of one 'order ready' event is sufficient to trigger all others. Note, however, that the message that the event has occurred may take some time to convey. This depicted, motivates the introduction of asynchronous triggers in the business process architecture.

A flow relation from an event e to an event e' represents that upon the occurrence of event e information is passed to event e', causing it to occur as well. However, a flow relation can never instantiate and start a process. Therefore, it should never have a start event as its target.

Using this behavioral semantics for trigger and flow relations, we define the behavioral contract implied by a business process architecture.

Definition 2 (Behavioral Contract of a Business Process Architecture). *Processes in a business process architecture $A = (E, P, T, F, M)$ must observe the following behavioral contract.*

- ◦ *An instance of process $p \in P$ must become active when a start event $s \in E^S \cap p$ occurs.*
- ◦ *A start event $e \in E^S$ is always ready to occur. An intermediate event $e \in E^T \cup E^C$ is ready to occur for active instances of $p \in P$ for which $e \in p$. An end event $e \in E^E$ is ready to occur for active instances of $p \in P$ for which $e \in p$. This implies that each intermediate event $i \in p \cap (E^T \cup E^C)$ has occurred a number of times specified in $M(i)$ for that instance.*
- ◦ *If $e \in E \wedge \bullet e = \emptyset$ holds, event e can occur for an instance, if it is ready to occur for that instance.*

- ◦ *If $e \in E \wedge \bullet e \neq \emptyset$ holds, event e must occur for an instance if and only if: (i) it is ready to occur for that instance; and (ii) it gets a trigger or flow object.*
- ◦ *An instance of process $p \in P$ ceases to be active if an end event $e \in E^E \cap p$ occurs.*
- ◦ *For a synchronous relation $(e_1, e_2) \in T^S \cup F^S$, if e_1 occurs, e_2 must get a trigger or flow object at the same time.*
- ◦ *For an asynchronous relation $(e_1, e_2) \in T^A \cup F^A$, if e_1 occurs, e_2 must get a trigger or flow object at some time in the future.*

A process instance should not end due to an event that belongs to another instance. For that reason, we pass along a trigger stack with each trigger. An invocation stack is the stack of process instance identifiers of active processes in the context of which a trigger is given. Each time a process is instantiated, it pushes its own identifier onto the stack that is received with the trigger. An end event can then only occur for a process instance due to a trigger, if the identifier at the top of the trigger stack corresponds to the identifier of the process instance. When the end event occurs, it pops the top from the stack. To enforce this behavior, the following contractual rules must be enforced.

Definition 3 (Instantiation Contract)
- ◦ *A trigger or flow $(e_1, e_2) \in T \cup F$ passes a stack from e_1 to e_2.*
- ◦ *When a start event occurs and an instance of p becomes active, a unique identifier for that instance is pushed onto the stack.*
- ◦ *An end event can only occur for a process instance due to a trigger, if the conditions from Definition 2 hold and the element at the top of the stack of the trigger corresponds to the identifier of the process instance.*
- ◦ *When an end event occurs, it pops the top from the stack of the process instance.*

For example, in Figure 1 process p can initially instantiate due to s_1, creating instance p^1 that is pushed onto the stack. The stack is passed on to s_2 and s_3. s_2 instantiates p_1, creating an instance p_1^1 with stack $[p^1, p_1^1]$. When p_1^1 ends due to e_2, the identifier is popped from the stack and the stack is passed on to e_1. The stack now contains $[p^1]$ and consequently e_1 can only occur for instance $[p^1]$.

The behavioral contract can be enriched by defining an abstract control flow for each process. A process must then not only observe the contract determined by the business process architecture, but also observe the order of events implied by the abstract control flow. An abstract control flow of a business process $p \in P$ is a business process model that defines the behavioral relations between the events in p and that does not contain any other events, tasks or sub-processes. Figure 2 shows a process with an abstract control flow, modeled in BPMN. We leave the incorporation of abstract control flows for future work.

Where the correctness of individual processes is usually verified by assuming that they have a clearly identifiable start and end [7], these properties cannot be used for a process architecture. For example, consider a process that continuously collects bills from other processes and, at the end of the month, pays those bills. Although we could consider the start and the end of the month as the

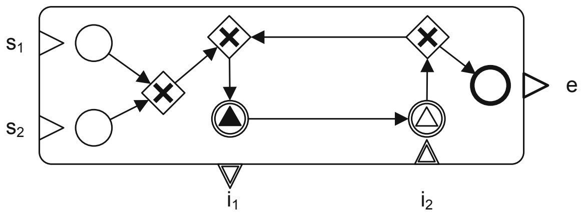


Fig. 2. Example of a process with abstract control flow

start and the end of the payment process, the other processes, which provide the payment process with its information and are therefore part of the same process architecture, most likely do not share the same start and end. Consequently, we rely on other notions of correctness. In particular, we assume a process architecture to be behaviorally correct, if it is free from deadlocks, livelocks, dead events and lost triggers or flow objects.

Definition 4 (Dead event, Lost trigger or flow object, Deadlock, Livelock, Correctness)

- *A dead event is an event that can never occur.*
- *A lost trigger or flow object is a trigger or flow object that either gets emitted by a synchronous relation at a moment at which the target event is not ready, or gets emitted by an asynchronous relation to an event that may never become ready.*
- *A process instance is in a deadlock, if it is active, has one or more end events, and none of its end events can occur at any moment in the future.*
- *A business process architecture is in a livelock, if it is in a state, from which it is not possible to reach a state in which all its processes are not active.*

A business process architecture is correct if and only if it is free from deadlocks, livelocks, dead events and lost triggers or flow objects.

In the remainder of this paper, we will define structural patterns that can be used to detect incorrect behavior in a process architecture. The definition of formal correctness verification rules that can be used to exhaustively detect incorrect behavior is left for future work.

3 Structural Patterns

The structural patterns in the following sections describe message flows between processes as well as their instantiation interdependencies in synchronous and asynchronous environments. This in particular is important to detect undesired behavior in a process architecture. The patterns consist of basic constructs, composite constructs and constructs of multiple instances.

For the design of correct process architectures hierarchical interdependencies need to be taken into account as well. Interdependencies within one hierarchical level are described in patterns 1-13 and 15-28. Hierarchical interdependencies are reflected in the patterns 14, 29 and 30. Multi-event patterns, though of interest, will be elaborated in detail in future work. The main attention will be given to anti-patterns as they expose erroneous interdependencies. We will describe the patterns in a brief and semi-formal way with the help of following definitions.

Definition 5 (Trigger and Flow relations). *For convenience, we define the relations:*

- $triggers^{s}_{int} = T^{s} \cap (E^{t} \times E^{s})$ *for processes that trigger a start of another process with an intermediate throwing event in synchronous environments.*
- $triggers^{s}_{end} = T^{s} \cap (E^{e} \times E^{s})$ *for processes that trigger a start of another process with an end event.*
- $sends^{s}_{int} = F^{s} \cap (E^{t} \times E^{c})$ *for processes that send an intermediate message to another process.*
- $sends^{s}_{end} = F^{s} \cap (E^{e} \times E^{c})$ *for processes that send a message to another process through an end event.*
- $triggers^{as}_{int} = T^{a} \cap (E^{t} \times E^{s})$ *for processes in an asynchronous environment that trigger a start of another process with intermediate throwing event.*
- $triggers^{as}_{end} = T^{a} \cap (E^{e} \times E^{s})$ *for processes in an asynchronous environment that trigger a start of another process with an end event.*
- $sends^{as}_{int} = F^{a} \cap (E^{t} \times E^{c})$ *for processes in an asynchronous environment that send an intermediate message to another process.*
- $sends^{as}_{end} = F^{a} \cap (E^{e} \times E^{c})$ *for processes in an asynchronous environment that send a message to another process through an end event.*

The patterns and anti-patterns are depicted in Fig. 3 and in Fig. 4 in which process p generally is the starting point of reading the patterns. The semantics of the visualization of the patterns is as follows. The triangles depicted in the patterns show incoming events when pointing inside the process symbol and outgoing events when pointing away from the process symbol respectively. Intermediate events either point up or down and start or end events to the right.

3.1 Basic Patterns

The very basic interaction patterns with only one relation between two processes are trigger patterns and flow patterns. One process is linked to another process or itself by a relation between a throwing (end, intermediate) and a catching event (start, intermediate). In trigger patterns one process instantiates another process. Flow patterns depict an information flow that can be observed between two processes. These are shown in patterns 1-4 in Fig. 3.

Trigger Patterns. Pattern 1 depicts a process p that triggers process r. Process p finishes with end event e_1 and process r starts with start event s_1, hence $(p, r) \in triggers^{s}_{end}$. Similarly, pattern 2 shows process p triggering process r but through its intermediate throwing event t_1, so that $(p, r) \in triggers^{s}_{int}$. Both patterns hold for asynchronous and synchronous environments.

Flow Pattern. Pattern 3-4 show an information flow from process p to process r through a throwing intermediate event or an end event respectively: $(p, r) \in sends^s_{int}$ in pattern 3 and $(p, r) \in sends^s_{end}$ in pattern 4. In both patterns process r must be instantiated in synchronous environments for the information exchange to occur. Else the information flow will be lost. In asynchronous environments, r does not need to be instantiated at the moment when process p passes the information. Hence, $(p, r) \in sends^{as}_{end}$ as well. The message will be read after instantiation when the process execution reaches the according state of the process where this input is required.

3.2 Composite Patterns

Patterns 5-8 in Fig. 3 show regular composite patterns that display a combination of triggering and information flow behavior between two processes p and r.

Flow Feedback Patterns. Patterns 5 and 7 describe feedback of information from process r to process p after having been instantiated by process p. In pattern 5 and 7 process r is triggered by a throwing intermediate event. During execution process r passes information to process p through a throwing intermediate event in pattern 5 and through an end event in pattern 7. In Pattern 5 $(p, r) \in triggers^s_{int}$ and $(r, p) \in sends^s_{int}$ whereas in pattern 7 $(p, r) \in triggers^s_{int} \wedge$ $(r, p) \in sends^s_{end}$.

Unidirectional Interaction Patterns. In contrast to patterns 5 and 7, pattern 6 displays only unidirectional interaction from process p to process r. Process p instantiates process r and passes a message to process r when it finishes. Process r needs to be still active in synchronous environments when process p finishes. Else the information flow object would be lost. In asynchronous environments the information object could then be read by the next triggered instance of process r, hence $(p, r) \in triggers^s_{int} \wedge sends^s_{end}$ and $(p, r) \in triggers^{as}_{int} \wedge sends^{as}_{end}$.

Send-Receive Pattern. In pattern 8, process p passes a message to process r through an outgoing intermediate event and vice versa, so that $(p, r) \in sends^s_{int}$ and $(r, p) \in sends^s_{int}$. Both processes need to be active in synchronous environments. In asynchronous environments they could be active at different times.

Broadcast. In pattern 13 a broadcast of the end event of process p to processes q and r can be observed. Both processes will be triggered.

3.3 Multi-instances Patterns

To describe multi-instantiation patterns we introduced cardinalities in section 2 that define the number of occurrences of one event in a relation between two processes. The cardinalities describe the minimum and maximum number of occurrences of the event. The three vertical lines depicted in process r in patterns 9-12 of Fig. 3 represent the possible parallel execution of multiple instances of process r.

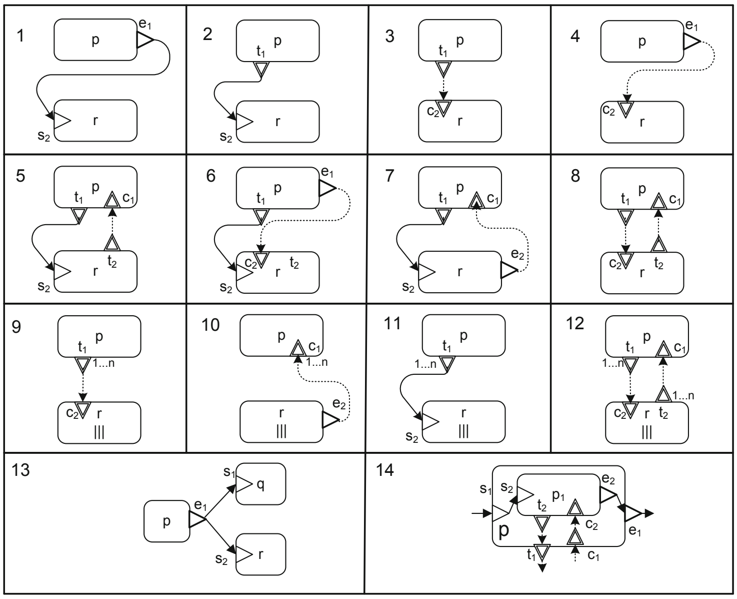


Fig. 3. Basic Patterns

One-to-Many-Broadcast. Pattern 9 displays a one to many broadcast pattern where process p sends information to many instances of process r. In synchronous environments instances of r need to be active as information objects are sent by p. In asynchronous environments the instances could also become active in sequential order.

Information Sink. Pattern 10 displays an information sink. Process p collects many information objects from several instances of process r that run in parallel. In synchronous environments all process instances need to be active. In asynchronous environments p can become active at a later stage to receive all information objects from process r.

Trigger Fleet. Pattern 11 depicts similar behavior like pattern 2. However, process p triggers many instances of process r, so that $(p, r) \in triggers^s_{int}$ and $M(t_1) = \{1 \ldots n\}$. In synchronous environments all process instances of process r become active at the same time. In asynchronous environment, the instances of r could become active at different points in time.

Pattern 12 is a combination of pattern 9 and 10 whereas the process instances of process r also send many information objects to process p. In this pattern $(p, r) \in sends^s_{int}$ with $M(t_1) = \{1 \ldots n\}$ and $(r, p) \in sends^s_{int}$ with $M(t_2) = \{1 \ldots n\}$.

3.4 Basic Anti-patterns

There are four basic process anti-patterns that demonstrate self-reflexive/looping behavior that expose incorrect behavior of the system. They are depicted in patterns 15-18 of Fig. 4. Anti-patterns that are considered regular patterns in asynchronous environments are marked with a * in Fig. 4 and in the heading of the pattern description.

Loop. Patterns 15 and 16 describe looping behavior in which a process p triggers itself either through its own throwing intermediate event or through its own end event. In pattern 15 process p triggers itself when finishing. This relation should not occur in a process architecture as such process could never be instantiated. When instantiated once, process p would trigger itself for infinite times as there is no other end event. Similarly in pattern 16, process p triggers itself with its throwing intermediate event t_1. Hence it can never become active as it waits for its own throwing intermediate event to trigger itself. Also if instantiated once, infinite instances of process p could possibly be instantiated as there is no other end event. $(p,p) \in triggers^s_{int}$, $(p,p) \in triggers^s_{end}$, $(p,p) \in triggers^{as}_{end}$, or $(p,p) \in triggers^{as}_{int}$ should never become true. The looping behavior is undesired in synchronous as well as asynchronous environments.

Dead Event*. Pattern 17 of Fig. 4 depicts a dead event. When process p is active it waits for an information flow from its own end event. t_1 will never occur, so that process p will never end. c_1 is a dead event as it never occurs.

If considering the intermediate events being optional, the information flow would be lost in synchronous environment as p sends a message to itself the moment it ceases. In an asynchronous environment this pattern can be considered a regular pattern as the message would be stored until p is active again. Hence $(p,p) \notin sends^s_{int}$ but $(p,p) \in sends^{as}_{int}$.

Self Messaging. Pattern 18 does not formally depict undesired behavior. However a process p that passes an information object to itself, displays improper behavior. The throwing intermediate event t_1 sends an information object to its catching intermediate event c_1. In both synchronous and asynchronous environments this could lead to process p not finishing as the message flow to itself would never stop or never start.

3.5 Composite Anti-patterns

The anti-patterns consist of lost flow patterns, inhibiting flows or triggers, and looping behavior.

Trigger Loop. Deriving from patterns 15 and 16, patterns 19 and 20 depict looping behavior over two processes. Process p triggers process r through a throwing intermediate event t_1. Process r in return triggers process p, so that the combination $(p,r) \in triggers^s_{int} \wedge (r,p) \in triggers^s_{int}$ or $(p,r) \in triggers^s_{int} \wedge (r,p) \in triggers^s_{end}$ resolves in a dead structure. This symmetric behavior describes a loop spanning over two processes in which neither process can be instantiated as they depend on being triggered by each other. If once active, multiple instances of process p could exist in both patterns at the same time. Patterns 25 and 26 depict another version of looping behavior, except that process p when finishing triggers process r. Either pattern cannot be instantiated as both processes are triggered by each other. In pattern 21 as well, process p is triggered by process r and wants to send an information object to r. $(r, p) \in triggers^s_{int} \wedge (p, r) sends^s_{int}$ lead to process p inhibiting the execution of process r. Process r is instantiated and waits for the throwing event t_1 of process p which cannot occur as process p is triggered by intermediate event t_2 of process r.

Lost Flows *. Patterns 22-24 in Fig. 4 show different constructs in which information objects are lost in synchronous environments. An information object gets lost when the receiver process is not active. In patterns 22 and 24 process p triggers process r when it finishes. In pattern 22, process r, while executing, passes a message to the not instantiated process p through a throwing intermediate event, so that $(p, r) \in triggers^s_{end}$ but $(r, p) \notin sends^s_{int}$. In pattern 24, process r, when it finishes, passes an information object to the not instantiated process p so that $(p, r) \in triggers^s_{end}$ and $(r, p) \notin sends^s_{end}$.

Pattern 23 describes a unidirectional relation information object loss. Process p is active and wants to pass an information object to process r. However, process r is not instantiated yet, as it is triggered by process p when process p finishes. Hence the information object will be lost, such that $(p, r) \notin sends^s_{int}$ but $(p, r) \in triggers^s_{end}$.

However, in asynchronous environments, patterns 22-24 are considered regular patterns as the information flows will be stored until the according process will be active and able to process it.

Multi-cast *. Patterns 27-28 depict a situation in which the receiving process r is triggered and gets an information object from one event of the sending process p at the same time in synchronous environments. The message object will be lost due to that. In asynchronous environments these pattern describe regular behavior as the message object will be buffered.

3.6 Nesting

A process architecture often exhibits hierarchical interdependencies of processes. The interdependencies between processes within one hierarchical level need to remain intact when decomposing a process into its subprocesses. To ensure strict hierarchy events should always be forwarded to events of the same type. If a parent process p consists of one or several child processes $p_{1..n}$, catching events of the parent process need to be forwarded to at least one of its child processes and throwing events from at least one child to the parent. For instance, in pattern 14 the throwing event e_2 of p_1 is forwarded to e_1 of process p.

3.7 Nesting Anti-patterns

Patterns 29 and 30 describe anti-patterns in hierarchical relations. In pattern 29 trigger or flow relations are not forwarded from the parent process to its child processes or the child processes throwing events are not forwarded to the according partner process of the parent. The trigger relations that exist between process n and process p should be forwarded to the child process p_1.

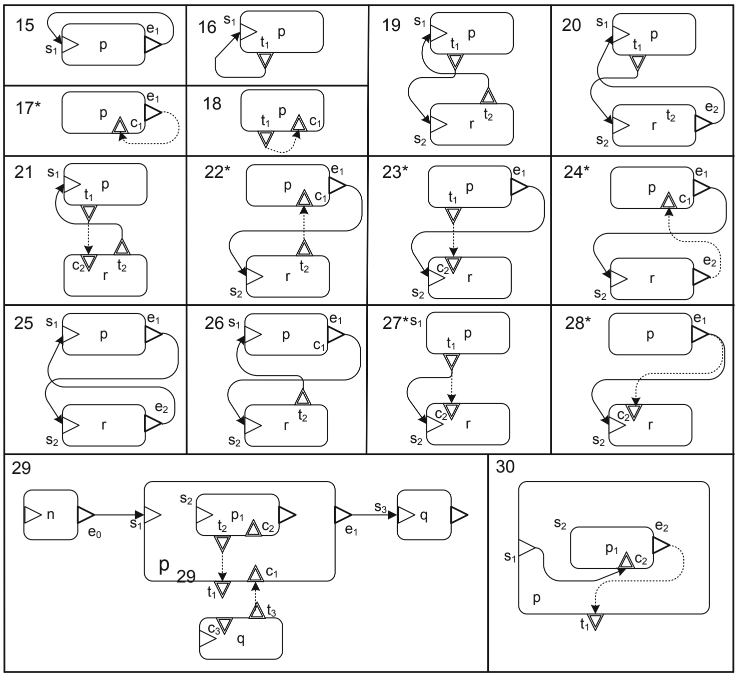


Fig. 4. Anti-Patterns

Pattern 30 shows wrong forwarding of events from parent to child process and vice versa. The trigger relation from process p should be forward to process p_1 such that start event s_1 of process p relates to start event s_2 of child process p_1. As s_1 of process p relates to an event of different type, the intermediate event c_2 of process p_1, the hierarchical decomposition is erroneous.

4 Evaluation

A business process architecture aims to give an overview of a collection of business process models, which helps to manage the complexity of interactions between models in that collection and, therewith, avoid errors. To evaluate the extent to which business process architecture can help do that, we analyzed the complexity of interactions between models in the SAP reference model collection [8], by constructing a process architecture for it and analyzing its architectural patterns.

Constructing a business process architecture for the SAP reference model is complicated by three factors.

First, the SAP Reference Model has a control flow semantics that is difficult to interpret automatically, in particular because it is known to contain errors in the control flow [9]. Since constructing the process architecture requires insight in the control flow semantics (in particular to analyze how often an event occurs

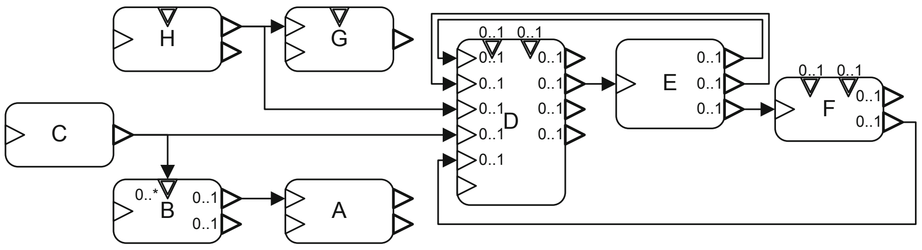


Fig. 5. Part of the SAP Reference Model Process Architecture

per instance), we were forced to investigate the collection manually. To make this feasible, we investigated **7** branches of the model with a total of **12** subbranches. These branches contain **95** of the total collection of 604 models. The interaction observed between process models remained within their branches. However, interaction across subbranches could be observed quite frequently.

Second, the SAP reference model does not explicitly distinguish between start, end and intermediate events, while a process architecture distinguishes between these different types of events. To identify these different types of events, we used the following definitions. A start event is an event that is meant to occur before any activity. An end event is an event that is meant to occur after all activities have occurred. An intermediate event is an event that is meant to occur after an activity has already occurred and before all activities have occurred. To determine whether an event is expected to occur at all for a single instance of a process, we assume that the processes are meant to be one-safe in a token-based execution semantics. For example, if a process has two start events A and B that are followed by an XOR-join, then we assume that either A or B should occur in a single process instance, but not both.

Third, the SAP Reference Model also does not distinguish explicitly between 'trigger' and 'flow' relations or between synchronous and asynchronous relations. Instead, it distinguishes conditions that can become true. We interpret each of these conditions as a synchronous trigger relation, as a condition that becomes true, becomes true at all places in the architecture at the same time.

Using these assumptions, we constructed process architectures for the various branches, like the one shown in Figure 5, and subsequently we identified the architectural patterns that are shown in Table 1. Figure 5 shows a process architecture for one of the smaller branches of the SAP Reference Model, consisting of only leaf processes. The processes have been anonymized for copyright reasons. The architecture provides some interesting insights. For example, it is clearly visible that process *C* can be executed multiple times and process *B* collects the results of the various instances of process *C*. During our examination of 95 business processes of the SAP Reference Model we mainly found normal patterns that derive from patterns 1-4. In general we found more triggering relations than flow relations between processes. Looking more closely we could observe more end trigger than intermediate trigger relations. Flow relations were dominated by interactions between two intermediate events, rather than flows between end

Table 1. Patterns in SAP-Reference Model

Pattern	Counts	Pattern	Counts
Trigger (Pattern 1 and 2)	95	Self-Messaging	2
Flow (Pattern 3 and 4)	13	Anti-Pattern 25	1
Broadcast	10	Anti-Pattern 26	3
Nesting	27	Nesting Anti-Patterns 29 and 30	38
Loop (Anti-Pattern 13 and 14)	4		

events and catching intermediate events. Two of the analyzed branches showed correct behavior as they did not expose any anti-patterns.

In the other five branches **48** anti-patterns could be observed. Nesting anti-patterns 29 and 30 could be observed with a high count of 14 (anti-pattern 29) and 24 (anti-pattern 30) respectively, in contrast to the observance of 27 regular nesting patterns. This is due to the fact that most of the parent processes of one branch in the SAP Reference Model are a composite of all subprocesses in their branch but exhibit not all intermediate events of the subprocesses that relate to other processes of other branches. Anti-pattern 16 and 23 were found each two times, and loop anti-patterns four times. However, one count of the loop anti-pattern and the two counts of anti-pattern 23 could resolve in a regular behavior when taking their abstract workflows into consideration as each process exposes several start and end events. Anti-pattern 26 could be observed three times in derivative form and anti-pattern 25 only once.

This discussion shows that the errors found in that process model collection could have been avoided by using our approach. It can lead to better understanding of the interdependencies between related process models and, ultimately, to better and more consistent process model collections. Future work will deal with refactoring approaches for repairing anti-patterns.

5 Related Work

A large part of literature on process architectures focuses on the organization of processes and the description of their interrelations within process collections. They facilitate processes classification [1] and mainly build on hierarchical interrelations between processes. However, most approaches do not take care of the interdependencies of processes in regard to their instantiation semantics and message flows. These topics are mainly dealt with in literature on service interaction patterns [3,4,5,6], process instantiation [10], process choreographies [11] and workflow modules [12,13]. In addition, Milner [14], Decker et al.[15] and Lucchi and Mazzara [16] use the π-calculus to analyze communication interaction between communicating parts.

Examining service properties, Barros et al. [4] present service interaction patterns. Their aim is to provide common patterns to benchmark technology. The basic patterns presented in previous sections align and cover the eight basic patterns proposed by Barros et al. [4]. We expand on this work by also considering trigger relations and using anti-patterns to analyze correctness.

Van der Aalst et al. [3] introduce the foundational concepts of service interaction and a collection of basic service interaction and correlation (anti-)patterns. To examine the compatibility of service interactions, they use a refined Petri Net specification. Similarly, our conceptual framework supports the correct design of process architectures, and the detection of undesired interdependencies between processes. Van der Aalst et al. [3] focus on the exchange of messages between services rather than triggering interdependencies between several processes.

In contrast, Decker and Mendling [10] focus only on instantiation semantics disregarding any message flows between processes. They propose the CASU classification framework for describing the instantiation semantics of single processes. Extending this, our approach takes into account triggering and flow relations to identify incorrect process interdependencies in a process architecture. So far, only some of their creation and activation patterns are covered by our approach. Patterns on subscription and unsubscription as well as multi-event patterns will be examined in the near future.

Examining interaction of workflow modules, Glabeek and Storck [13] focus on the property of termination. They propose to ensure global termination of workflow nets by checking local properties only. We look at termination rather from a holistic perspective and assume that multiple instances of one process can exist during one instance of another process. Martens [12] analyzes the interaction and replacement of parts of composite web services in regard to deadlocks and soundness. He introduces communication and usability graphs to evaluate the behavior of the composite workflow modules.

Decker and Weske [11] emphasize the need for behavioral consistency checking in process choreographies to ensure integrated processes interaction. They examine various consistency and compatibility definitions and introduce a compatibility and consistency framework. Wombacher [17] investigates consistency checking in cross-organizational workflows. Their definitions of consistency aim to define overall correct behavior when processes interact with each other.

Milner [14] looks at interaction between two components from a different angle, considering each component as communicating part. Their communication properties are defined and described by the π-calculus. Based on Milner's work, Lucchi and Mazzara [16] define BPEL's error handling constructs with the π-calculus and extend its transaction semantics to be able to analyze web service orchestration. Decker et al. [15] introduce a formalism for Let's Dance, based on the π-calculus to be able to analyze execution semantics of process choreographies and detect unreachable interactions between processes.

6 Conclusions

This paper presents a formal conceptualization of process architecture. A process architecture is a collection of business processes and their relations with each other. The conceptualization serves as a starting point for analyzing the correctness of business processes in their relation to each other.

The paper also presented a collection of patterns that define possible relations that business processes can have with each other in an organization, including anti-patterns that lead to incorrect behavior. Incorrect behavior includes dead events, lost triggers or flow objects, deadlock and livelock of the process architecture as a whole, even if the individual processes do not contain such incorrect behavior.

An evaluation on a collection of 95 industry-strength business processes shows that process architecture can be used to better understand the relations between business processes and that it can be used to uncover incorrect behavior. In particular 48 anti-patterns of different types were uncovered during the evaluation.

Although the paper demonstrates that the anti-pattern based approach can be used to detect incorrect behavior, it cannot be claimed that this approach can be used to uncover all problems related to dead events, lost triggers or flow objects, deadlock and livelock. Problems that are not captured as an anti-pattern will not be detected. Therefore, the concepts of dead events, lost triggers or flow objects, deadlock and livelock in a process architecture should be defined formally and formal techniques, and tools should be developed to detect these problems.

Acknowledgments. We acknowledge the comments of the reviewers, especially regarding BPMN semantics for abstract control flows.

References

1. Dijkman, R.M., Vanderfeesten, I., Reijers, H.A.: The Road to a Business Process Architecture: An Overview of Approaches and their Use. BETA Working Paper WP-350, Eindhoven University of Technology, The Netherlands (2011)
2. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd edn. Springer (2012)
3. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
4. Barros, A., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
5. Barros, A., Börger, E.: A Compositional Framework for Service Interaction Patterns and Interaction Flows. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 5–35. Springer, Heidelberg (2005)
6. Tut, M.T., Edmond, D.: The Use of Patterns in Service Composition. In: Bussler, C., Hull, R., McIlraith, S., Orlowska, M.E., Pernici, B., Yang, J. (eds.) CAiSE 2002 and WES 2002. LNCS, vol. 2512, pp. 28–40. Springer, Heidelberg (2002)
7. van der Aalst, W.M.P.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
8. Curran, T.A., Keller, G.: SAP R/3 Business Blueprint - Business Engineering mit den R/3-Referenzprozessen. Addison-Wesley, Germany (1999)
9. Mendling, J., van der Aalst, W.M.P., van Dongen, B., Verbeek, E.: Errors in the SAP Reference Model. BPTrends 4(6), 1–5 (2006)

10. Decker, G., Mendling, J.: Process instantiation. TKDE 68(9), 777–792 (2009)
11. Decker, G., Weske, M.: Behavioral Consistency for B2B Process Integration. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 81–95. Springer, Heidelberg (2007)
12. Martens, A.: Analyzing Web Service Based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
13. van Glabbeek, R.J., Stork, D.G.: Query Nets: Interacting Workflow Modules That Ensure Global Termination. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 184–199. Springer, Heidelberg (2003)
14. Milner, R.: Communicating and mobile systems - the Pi-calculus. Cambridge University Press (1999)
15. Decker, G., Zaha, J.M., Dumas, M.: Execution Semantics for Service Choreographies. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 163–177. Springer, Heidelberg (2006)
16. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming 70(1), 96–118 (2007)
17. Wombacher, A.: Decentralized Consistency Checking in Cross-organizational Workflows. In: CECEEE 2006, pp. 39–46. IEEE Computer Society Press (2006)

Aligning Event Logs and Declarative Process Models for Conformance Checking

Massimiliano de Leoni*, Fabrizio Maria Maggi, and Wil M. P. van der Aalst

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.d.leoni,f.m.maggi,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining can be seen as the "missing link" between data mining and business process management. Although nowadays, in the context of process mining, process discovery attracts the lion's share of attention, conformance checking is at least as important. Conformance checking techniques verify whether the observed behavior recorded in an event log matches a modeled behavior. This type of analysis is crucial, because often real process executions deviate from the predefined process models. Although there exist solid conformance checking techniques for procedural models, little work has been done to adequately support conformance checking for *declarative models*. Typically, traces are classified as fitting or non-fitting without providing any detailed diagnostics. This paper aligns event logs and declarative models, i.e., events in the log are related to activities in the model if possible. The alignment provides then sophisticated diagnostics that pinpoint where deviations occur and how severe they are. The approach has been implemented in ProM and has been evaluated using both synthetic logs and real-life logs from Dutch municipalities.

1 Introduction

Traditional Workflow Management Systems (WFMSs) are based on the idea that processes are described by procedural languages where the completion of a task may enable the execution of other tasks. While such a high degree of support and guidance is certainly an advantage when processes are repeatedly executed in the same way, in dynamic settings (e.g., healthcare) a WFMS is considered to be too restrictive. Users often need to react to exceptional situations and execute the process in the most appropriate manner. Therefore, in these environments systems tend to provide more freedom and do not restrict users in their actions. Comparing such dynamic process executions with procedural models may reveal many deviations that are, however, not harmful. In fact, people may exploit the flexibility offered to better handle cases. In such situations we advocate the use of *declarative models*. Instead of providing a procedural model that enumerates all process behaviors that are allowed, a declarative model simply lists the constraints that specify the forbidden behavior, i.e., "everything is allowed unless explicitly forbidden".

In this paper, we focus on *conformance checking for declarative models*. Conformance checking techniques take an event log and a process model and compare the

* The research of Dr. de Leoni has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no 257593 (ACSI).

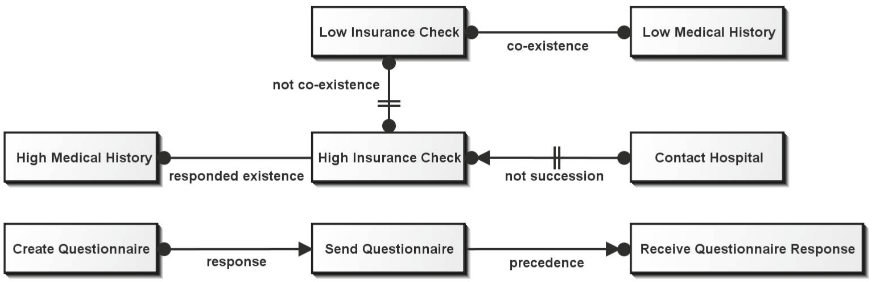


Fig. 1. Declare model consisting of six constraints and eight activities

observed behavior with the modeled behavior [3,4,5,16]. Along with *process discovery* (learning process models from event logs) and *process enhancement* (e.g., extending process models using information extracted from the actual executions recorded in event logs), conformance checking belongs to the area of *process mining* [3].

Since often the actual execution deviates from the prescriptions of theoretical models, this type of analysis is critical in many domains, e.g., process auditing [17] or risk analysis [11]. However, while there is a large stream of research about conformance checking of procedural models, little work has been conducted for declarative models. Existing approaches exclusively focus on determining whether a given process instance conforms with a given process model or not [7,6,9,1]. In this paper, we want to provide diagnostics at the event log level rather than a simple yes/no answer at the trace level. Moreover, our approach is able to pinpoint where deviations more frequently occur and how severely a process instance does not comply the process model.

There are different quality dimensions for comparing process models and event logs, e.g., *fitness*, *simplicity*, *precision*, and *generalization* [3,15,10]. In this paper, we focus on *fitness*: a model with good fitness allows for most of the behavior seen in the event log. A model has a *perfect* fitness if all traces in the log can be replayed by the model from the beginning to the end. Our approach is applied to Declare, a declarative language supported by a toolset that includes a designer, a workflow engine, a worklist handler, and various analysis tools [20,2].[1] Due to space reasons we cannot provide a detailed description of Declare and only highlight some of the most relevant features of the language through an example.

Example 1. *A travel agency has enacted a process to handle health insurance claims. A claim can be classified as high or low, depending on the amount that the customer claims to receive back. Fig. 1 shows a simple Declare model with some example constraints to describe this process. The model includes eight activities (depicted as rectangles, e.g.,* Contact Hospital*) and six constraints (shown as connectors between the activities). For low claims, two tasks* Low Insurance Check *and* Low Medical History *need to be executed. The* co-existence *constraint indicates that these activities always occur together (in any order). If a claim is classified as low, no activities referring to high claims can be executed and vice versa. The* not co-existence *constraint indicates that* Low Insurance Check *and* High Insurance Check *can never coexist in the same process instance. Moreover, in case of high claims, the medical history check (*High Medical History*)*

[1] Declare web site - http://www.win.tue.nl/declare/

*can only be executed together with the insurance check (*High Insurance Check*), even though they can be executed in any order. Nevertheless, it is possible to execute* High Insurance Check *without executing* High Medical History*. All this is enforced by the* responded existence *constraint. For every claim, it is also possible to contact the doctor/hospital for verification. However, in case of high claims, this cannot be done before the insurance check. This is defined by the* not succession *constraint, that means that* Contact Hospital *cannot be followed in the same process instance by* High Insurance Check*. In this process, a questionnaire is also created and eventually sent to the applicant (modeled by the* response *constraint); after having sent the questionnaire, the applicant can possibly decide whether to fill it in or not (*precedence *constraint).*

The approach we propose is based on the principle of finding an *alignment* of an event log and a process model. The concept of *alignment* has successfully been used in the context of procedural models (e.g., [4,5,12]); here, we adapt it for declarative models. Similarly to what has been proposed for procedural models, in our approach, events in the log are mapped onto executions of activities in the process model. A cost/weight is assigned to every potential deviation. We use the A* algorithm [4,8] to find, for each trace in the event log, an optimal alignment, i.e., an alignment that minimizes the cost of the deviations. The application of the A* algorithm is more challenging for declarative models than for procedural models. This is due to the fact that, since in a declarative model everything is allowed unless constrained otherwise, the set of admissible behaviors is generally far larger than the set of behaviors allowed by procedural models. This implies that the search space where to find an optimal alignment of a log and a declarative model is much larger. Therefore, for this type of models, it is essential to avoid exploring search-space nodes that certainly lead to non-optimal solutions.

In addition to simply returning an optimal alignment for each trace, we also provide the process analyst with a summary that gives a helicopter view of the conformance of the model with respect to the entire log. In particular, we aggregate the information associated to the optimal alignments and visualize the deviations upon the process model. In fact, we generate a "map" thus highlighting on a Declare model which constraints are more often violated during the performance and which activities are mostly involved in the deviations. The approach has been implemented in ProM and has been evaluated on a variety of synthetic and real-life logs.

The paper is structured as follows. Section 2 introduces event logs and Declare models. Section 3 describes the notion of alignment and how it can be used for conformance checking. Section 4 describes the application of the A* algorithm to find an optimal alignment. Here, we also introduce an optimization of the algorithm to prune large irrelevant parts of the search space (that do not lead to an optimal solution). Section 5 shows which diagnostics and feedback we can provide to the process analyst. Section 6 presents the plug-ins we have implemented and reports some performance results. Here we also validate our approach by applying it to real-life logs of Dutch municipalities. Section 7 concludes the paper highlighting future work.

2 Basic Concepts

Declare is a language that provides both an intuitive graphical representation and a formal semantics for declarative process models. In particular, Declare is grounded in Linear Temporal Logic (LTL) with a finite-trace semantics and every Declare constraint is formally defined through an LTL formula.[2] For instance, the *response* constraint in Fig. 1 can be formally represented using LTL as $\square(Create\ Questionnaire \rightarrow \lozenge Send\ Questionnaire)$. This means that "whenever activity *Create Questionnaire* is executed, eventually activity *Send Questionnaire* is executed". In the following, we refer to a **Declare model** $\mathcal{D} = (A, \Pi)$, where A is a set of activities and Π is a set of Declare constraints defined over activities in A.

To identify potential deviations of a log from a reference Declare model, we need to map each activity in the log onto an activity in the model. Each process instance in a log follows a trace of events and different instances may follow the same trace. Therefore, an event log can be seen as a multi-set of traces, i.e., $\mathcal{L} \in \mathbb{B}({A_L}^*)$[3], where A_L is the set of activities in the log. Since Declare is an "open" language, it allows for the execution of any activity which is not in the model. Therefore, the set of activities in the log may include all activities in the model and more, i.e., $A \subseteq A_L$.

For conformance checking, we do not need to distinguish the activities in the set $A_L \setminus A$ as they do not appear in the model. This allows us to reduce the space of the allowed behaviors. Note that we cannot completely abstract from the activities in $A_L \setminus A$ because some constraints use LTL's next operator (e.g., the chain response and chain precedence constraints). Therefore, we map all events referring to some activity in $A_L \setminus A$ to $\checkmark$. Afterwards, the log $\mathcal{L} \in \mathbb{B}({A_L}^*)$ is converted into $\mathcal{L}' \in \mathbb{B}(\Sigma^*)$ with $\Sigma = A \cup \{\checkmark\}$. We use function $\chi \in A_L \rightarrow \Sigma$ that maps each activity in A_L onto Σ. $\forall a \in A.\ \chi(a) = a$ and $\forall a \in (A_L \setminus A).\ \chi(a) = \checkmark$, i.e., every occurrence of an activity in a log trace that is also defined in the model is mapped onto the activity itself, whereas other events are mapped onto $\checkmark$.

Example 1 (cont.). *The set of activities of the Declare model in Fig. 1 is*

$$A = \langle \textit{Low Insurance Check}, \textit{Low Medical History}, \textit{High Insurance Check}, \textit{High Medical History}, \textit{Contact Hospital}, \textit{Create Questionnaire}, \textit{Send Questionnaire}, \textit{Receive Questionnaire Response}\rangle$$

Let us assume to have a log that contains the following trace:

$$\sigma_L = \langle \textit{Register}, \textit{Low Insurance Check}, \textit{Create Questionnaire}, \textit{Prepare Notification Content}, \textit{Create Questionnaire}, \textit{Send Notification by e-mail}, \textit{Send Notification by Post}, \textit{Archive}\rangle$$

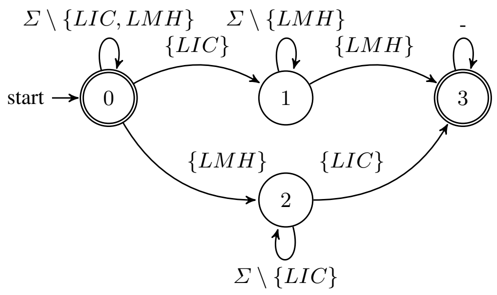
This log trace is defined over an activity set A_L that includes A. Using the mapping function χ, σ_L can be transformed into σ'_L.

$$\sigma'_L = \langle \checkmark, \textit{Low Insurance Check}, \textit{Create Questionnaire}, \checkmark, \textit{Create Questionnaire}, \checkmark, \checkmark, \checkmark\rangle$$

Note that Register, Send Notification by e-mail, Send Notification by Post, Archive are mapped onto $\checkmark$.

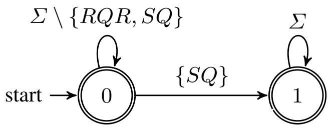
In the remainder, we only consider event logs after mapping unconstrained activities onto $\checkmark$, i.e., $\mathcal{L} \in \mathbb{B}(\Sigma^*)$. To check whether a log trace $\sigma_L \in \mathcal{L}$ is compliant with

[2] For compactness, in the following we will use the LTL acronym to denote LTL on finite traces.
[3] $\mathbb{B}(X)$ the set of all multi-sets over X and X^* is the set of all finite sequences over X.



$\Sigma \setminus \{LIC, LMH\}$
$\Sigma \setminus \{LMH\}$
-
$\{LIC\}$
$\{LMH\}$
start
0
1
3
$\{LMH\}$
$\{LIC\}$
2
$\Sigma \setminus \{LIC\}$

(a) Constraint automaton for the co-existence constraint



(b) Constraint automaton for the precedence constraint

Fig. 2. Constraint automata for two Declare constraints in Fig. 1

a Declare constraint $\pi \in \Pi$, using the technique described in [19], we translate the corresponding LTL formula into a final-state automaton that accepts all traces that do not violate constraint π.

Definition 1 (Constraint Automaton). *Let $\mathcal{D} = (A, \Pi)$ be a Declare model, $\pi \in \Pi$ and $\Sigma = A \cup \{\checkmark\}$. The* constraint automaton $\mathcal{A}_\pi = (\Sigma, \Psi_\pi, \psi_{0_\pi}, \delta_\pi, F_\pi)$ *is the final-state automaton which accepts precisely those traces $\sigma \in \Sigma^*$ satisfying π, where:*

- *$\Sigma = A \cup \{\checkmark\}$ is the input alphabet;*
- *Ψ_π is a finite, non-empty set of states;*
- *$\psi_{0_\pi} \in \Psi_\pi$ is an initial state;*
- *$\delta_\pi \in \Psi_\pi \times \Sigma \rightarrow \Psi_\pi$ is the state-transition function;*
- *$F_\pi \subseteq \Psi_\pi$ is the set of final states.*

We create a constraint automaton $\mathcal{A}_\pi$ for every constraint $\pi \in \Pi$. These automata can be used to check the conformance of a log trace with respect to each constraint in $\mathcal{D}$.

Example 1 (cont.). *For the* co-existence *constraint and the* precedence *constraint in Fig. 1, we obtain the automata depicted in Fig. 2a and 2b. In both cases, state* 0 *is the initial state and accepting states are indicated using a double outline. A transition is labeled with the set of the activities triggering it (we use the initial letters to denote an activity, e.g., we use LIC to indicate* Low Insurance Check*). This indicates that we can follow the transition for any event included in the set (e.g., we can execute event* High Insurance Check *from state* 0 *of the* precedence *automaton and remain in the same state).*

The **process behavior set** $\mathcal{P}_\mathcal{D} \subseteq \Sigma^*$ of a Declare model $\mathcal{D} = (A, \Pi)$ is the set of traces that are accepted by all automata $\mathcal{A}_\pi$ with $\pi \in \Pi$, i.e., all process executions that comply the model $\mathcal{D}$.

3 The Conformance Checking Framework

To check the conformance of an event log $\mathcal{L}$ with respect to a Declare model $\mathcal{D}$, we adopt an approach where we search for an *alignment* of the log and the model. Such an alignment shows how the event log can be replayed on the Declare model.

An alignment relates *moves in log* and to *moves in model* as explained in the following definition. Here, we explicitly indicate *no move* with $\gg$. $\Sigma_{\gg} = \Sigma \cup \{\gg\}$, where Σ denotes the input alphabet of each constraint automaton in $\mathcal{D}$.

Definition 2 (Alignment). *A pair* $(s', s'') \in (\Sigma_{\gg} \times \Sigma_{\gg}) \setminus \{(\gg, \gg)\}$ *is*

- *a* move in log *if* $s' \in \Sigma$ *and* $s'' = \gg$*,*
- *a* move in model *if* $s' = \gg$ *and* $s'' \in \Sigma$*,*
- *a* move in both *if* $s' \in \Sigma$*,* $s'' \in \Sigma$ *and* $s' = s''$*.*

$\Sigma_A = (\Sigma_{\gg} \times \Sigma_{\gg}) \setminus \{(\gg, \gg)\}$ *is the set of the* legal moves.

The alignment *of two execution traces* $\sigma', \sigma'' \in \Sigma^*$ *is a sequence* $\gamma \in {\Sigma_A}^*$ *such that the projection on the first element (ignoring* $\gg$*) yields* σ' *and the projection on the second element yields* σ''*.*

In particular, if $\sigma' = \sigma_L \in \mathcal{L}$ and $\sigma'' \in \mathcal{P}_{\mathcal{D}}$, we refer to the alignment γ as a *complete alignment* of σ_L and $\mathcal{D}$. An alignment of the event log $\mathcal{L}$ and the Declare model $\mathcal{D}$ is a multi-set $\mathcal{A} \in \mathbb{B}({\Sigma_A}^*)$ of alignments such that, for each log trace σ_L, there exists an alignment $\gamma \in \mathcal{A}$ of σ_L and $\mathcal{D}$. The definition of $\mathcal{A}$ as a multi-set is motivated by the fact that an event log can contain the same log trace σ_L multiple times and, hence, the same alignment can be given for all its occurrences.

Example 1 (cont.). *Given the log trace* $\sigma_L = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$*, there are many possible complete alignments of* σ_L *and the* Declare *model in Fig. 1. For instance, the following are valid complete alignments:*

$$\gamma_1 = \begin{array}{l|c|c|c|c|c|c|c|c|}\textbf{L:}\ \checkmark & LIC & CQ & \checkmark & CQ & \gg & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:}\ \checkmark & \gg & CQ & \checkmark & CQ & SQ & \checkmark & \checkmark & \checkmark\end{array} \qquad \gamma_2 = \begin{array}{l|c|c|c|c|c|c|c|c|}\textbf{L:}\ \checkmark & LIC & \gg & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:}\ \checkmark & LIC & LMH & \gg & \checkmark & \gg & \checkmark & \checkmark & \checkmark\end{array}$$

$$\gamma_3 = \begin{array}{l|c|c|c|c|c|c|c|c|c|}\textbf{L:}\ \checkmark & LIC & \gg & CQ & \checkmark & CQ & \gg & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:}\ \checkmark & LIC & LMH & CQ & \checkmark & CQ & SQ & \checkmark & \checkmark & \checkmark\end{array} \qquad \gamma_4 = \begin{array}{l|c|c|c|c|c|c|c|}\textbf{L:}\ \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:}\ \checkmark & \gg & \gg & \checkmark & \gg & \checkmark & \checkmark & \checkmark\end{array}$$

Conversely, $\gamma_0 = \begin{array}{l|c|c|c|c|c|c|c|}\textbf{L:}\ \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:}\ \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark\end{array}$ *is not a complete alignments since* σ_L *is not in the process behavior set of the* Declare *model. Indeed, the* co-existence *constraint is violated, because* Low Insurance Check *occurs in the log trace and* Low Medical History *does not. Moreover, two occurrences of* Create Questionnaire *are not followed by* Send Questionnaire*, as prescribed by the* response *constraint.*

In order to quantify the severity of a deviation, we introduce a cost function on the legal moves $\kappa \in \Sigma_A \to \mathbb{R}_0^+$. One can use a standard cost function with unit costs for moves in log or in model. However, the costs may also depend on the specific characteristics of the process, e.g., it may be more costly to skip an insurance check for high claims than for low claims. Therefore, a different cost function κ needs to be defined for individual processes. The cost of an alignment γ is defined as the sum of the costs of the individual moves in the alignment, $\mathcal{K}(\gamma) = \sum_{(s',s'') \in \gamma} \kappa(s', s'')$.

Given a log trace $\sigma_L \in \mathcal{L}$, our goal is to find a complete alignment of σ_L and a valid trace $\sigma_M \in \mathcal{P}_{\mathcal{D}}$ that minimizes the cost with respect to all $\sigma'_M \in \mathcal{P}_{\mathcal{D}}$. This complete alignment is referred to as an *optimal alignment*.

Definition 3 (Optimal Alignment). *Let $\sigma_L \in \mathcal{L}$ be a log trace and $\mathcal{D}$ a Declare model. Let $\Gamma_{(\sigma_L,\mathcal{D})}$ be the set of the complete alignments of σ_L and $\mathcal{D}$. A complete alignment $\gamma \in \Gamma_{(\sigma_L,\mathcal{D})}$ is an* optimal alignment *of $\sigma_L \in \mathcal{L}$ and $\mathcal{D}$ iff $\forall \gamma' \in \Gamma_{(\sigma_L,\mathcal{D})}$. $\mathcal{K}(\gamma') \geq \mathcal{K}(\gamma)$. The projection of γ on the second element (ignoring $\gg$) yields $\sigma_M \in \mathcal{P}_{\mathcal{D}}$, i.e., a valid trace of $\mathcal{D}$ that is the closest to σ_L.*

Example 1 (cont.). *In our example, we can suppose that deviations for activity* Send Questionnaire *are less severe than those referring to the other activities, since this activity is automatically performed by a system. Moreover, moves associated to $\checkmark$ can be weighted less than any other, since they refer to activities that are not in the model. Therefore, a reasonable cost function on legal moves can be defined as follows:*

$$\kappa(a', \gg) = \kappa(\gg, a') = \begin{cases} 1 & \textit{if } a' = \checkmark \\ 2 & \textit{if } a' = \text{Send Questionnaire} \\ 4 & \textit{otherwise} \end{cases}$$

$$\kappa(a', a'') = \begin{cases} 0 & \textit{if } a' = a'' \\ \infty & \textit{if } a' \neq a'' \wedge a' \neq \gg \wedge a'' \neq \gg \end{cases}$$

Using this cost function, alignments γ_1, γ_2, γ_3 and γ_4 have the following costs: $\mathcal{K}(\gamma_1) = 6$, $\mathcal{K}(\gamma_2) = 12$, $\mathcal{K}(\gamma_3) = 6$ and $\mathcal{K}(\gamma_4) = 12$. Therefore, γ_1 and γ_3 are better complete alignments. According to the given cost function, γ_1 and γ_3 are, in fact, optimal alignments.

When focusing on the fitness dimension of conformance, we are not only interested in finding the optimal alignment and, hence, diagnosing where a log trace does not conform with a model. We also need to quantify the fitness level of traces and logs. Therefore, we introduce a *fitness function* $\mathcal{F} \in (\Sigma^* \times 2^{\Sigma^*}) \rightarrow [0, 1]$. $\mathcal{F}(\sigma_L, \mathcal{D}) = 1$ if σ_L can be replayed by the model from the beginning to the end with no non-conformance costs. Conversely, $\mathcal{F}(\sigma_L, \mathcal{D}) = 0$ denotes a very poor fitness. $\mathcal{K}(\cdot)$ cannot be used as fitness function directly, as we are interested in expressing the fitness level as a number between 0 and 1. The normalization between 0 and 1 can be done in several ways. In our approach, we divide the cost of the optimal alignment by the maximal possible alignment cost. Typically, the greatest possible cost of an alignment of a log trace $\sigma_L = \langle a_1^L, \ldots, a_n^L \rangle$ and a model trace $\sigma_M = \langle a_1^M, \ldots, a_m^M \rangle \in \mathcal{P}_{\mathcal{D}}$ is obtained for the *reference alignment* in which there are only moves in model and in log:

$$\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)} = \begin{array}{l|l|l|l|l|l} \textbf{L:}\ a_1^L & \ldots & a_n^L & \gg & \gg & \gg \\ \hline \textbf{P:}\ \gg & \gg & \gg & a_1^M & \ldots & a_m^M \end{array} .$$

Therefore, the *fitness level* of a log trace can be defined as follows:

Definition 4 (Fitness Level). *Let $\sigma_L \in \Sigma^*$ be a log trace and let $\mathcal{D}$ be a Declare model. Let $\gamma_O \in {\Sigma_A}^*$ be an optimal alignment of σ_L and $\mathcal{D}$ and $\sigma_M \in \mathcal{P}_{\mathcal{D}}$ the model trace in the optimal alignment. The fitness level of σ_L and $\mathcal{D}$ is defined as follows:*

$$\mathcal{F}(\sigma_L, \mathcal{D}) = 1 - \frac{\mathcal{K}(\gamma_O)}{\mathcal{K}(\gamma^{\textit{ref}}_{(\sigma_L,\sigma_M)})}$$

Therefore, the fitness of σ_L and $\mathcal{D}$ is valued 1 if in the alignment there are only moves in both, i.e., there are no deviations. $\mathcal{F}(\sigma_L, \mathcal{D}) = 0$ if the optimal alignment only contains moves in log and in model. Note that this fitness function always returns a value between 0 and 1: if $\gamma_O \in \Sigma_A{}^*$ is an optimal alignment, any other alignment, including $\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)}$, must have the same or a higher cost.

In the next section, we introduce an approach to create an optimal alignment with respect to a custom cost function κ. The approach is based on the A* algorithm that is intended to find the path with the lowest overall cost between two nodes in a direct graph with costs associated to nodes.

4 The A* Algorithm for Conformance Checking

Let us suppose to have a graph V with costs associated to arcs. The A* algorithm, initially proposed in [8], is a pathfinding search in V. It starts at a given *source* node $v_0 \in V$ and explores adjacent nodes until one node of a given *target set* $V_{Trg} \subset V$ of destination nodes is reached, with the intent of finding the path with the overall lowest cost. Every node $v \in V$ is associated to a cost, which is determined by an *evaluation* function $f(v) = g(v) + h(v)$, where

- $g : V \to \mathbb{R}^+_0$ is a function that returns the smallest path cost from v_0 to v;
- $h : V \to \mathbb{R}^+_0$ is an heuristic function that estimates the smallest path cost from v to any target node $v' \in V_{Trg}$.

Function h is said to be *admissible* if it never underestimates the smallest path cost to reach any target node: for each node $v \in V$ and for each target node $v' \in V_{Trg}$ reachable from v, $h(v) \leq g(v')$. Technical results in [8] shows that if h is admissible, A* finds a path that is guaranteed to have the overall lowest cost.

The A* algorithm keeps a priority queue of nodes to be visited: a higher priority is given to nodes with lower costs so as to traverse those with the lowest costs first. The algorithm works iteratively: at each step, the node v with lowest cost is taken from the priority queue. If v belongs to the target set, the algorithm ends returning v. Otherwise, v is expanded: every successor v' of v is added to the priority queue with cost $f(v')$.

4.1 Usage of A* to Find an Optimal Alignment

We use A* to find any of the optimal alignments of a log trace $\sigma_L \in \Sigma^*$ and a Declare model $\mathcal{D}$. In order to be able to apply A*, an opportune search space needs to be defined. Every node γ of the search space V is associated to a different alignment that is a prefix of some complete alignment of σ_L and $\mathcal{D}$. Since a different alignment is also associated to every node and vice versa, later on we use the alignment to refer to the associated node. The source node is the empty alignment $\gamma_0 = \langle\rangle$ and the set of target nodes includes every complete alignment of σ_L and $\mathcal{D}$. Since the successors of an alignment are obtained by adding a move to it, the search space is, in fact, a tree.

Let us denote the length of a trace σ with $\|\sigma\|$. Given a node/alignment $\gamma \in V$, the search-space successors of γ include all alignments $\gamma' \in V$ obtained from γ by concatenating exactly one move. Let us consider a custom cost function κ and denote with $\kappa^{\min}$ the smallest value returned by κ greater than 0. Given an alignment $\gamma \in V$ of σ'_L and σ'_M, the cost of a path from the initial node to $\gamma \in V$ is defined as:

$$g(\gamma) = \kappa^{\min} \cdot \|\sigma'_L\| + \mathcal{K}(\gamma).$$

It is easy to check that, given two complete alignments γ'_C and γ''_C, $\mathcal{K}(\gamma'_C) < \mathcal{K}(\gamma'_C) \Leftrightarrow g(\gamma'_C) < g(\gamma''_C)$ and $\mathcal{K}(\gamma'_C) = \mathcal{K}(\gamma'_C) \Leftrightarrow g(\gamma'_C) = g(\gamma''_C)$. Therefore, an optimal solution returned by the A* algorithm coincides with an optimal alignment. We have added the term $\kappa^{\min} \cdot \|\sigma'_L\|$ (which does not affect the optimality) to define a more efficient and admissible heuristics. Given an alignment $\gamma \in V$ of σ'_L and σ'_M, we utilize the following heuristic:

$$h(\gamma) = \kappa^{\min} \cdot (\|\sigma_L\| - \|\sigma'_L\|)$$

For an alignment γ, the number of moves to add in order to reach a complete alignment cannot exceed the number of moves of σ_L that have not been included yet in the alignment, i.e., $\|\sigma_L\| - \|\sigma'_L\|$. Since the additional cost to traverse a single node is at least $\kappa^{\min}$, the cost to reach a target node is at least $h(\gamma)$, corresponding to the case in which the part of the log trace that still needs to be included in the alignment (i.e., $\sigma_L \setminus \sigma'_L$) fits in full.

4.2 Search Space Reduction

Declarative models allow for more flexibility and, therefore, for more behavior than procedural models. Hence, the search space in the A* algorithm may be extremely large. Nevertheless, many search-space nodes (i.e., partial alignments) are, in fact, equivalent, i.e., some partial alignments can be extended with the same moves:

Definition 5 (Alignment Equivalence). *Let $\mathcal{D} = (A, \Pi)$ be a Declare model and let $\mathcal{A}_\pi = (\Sigma, \Psi_\pi, \psi_{0\pi}, \delta_\pi, F_\pi)$ be the constraint automaton for $\pi \in \Pi$. Let $\sigma_L \in \Sigma^*$ be a log trace. Let γ' and γ'' be alignments of σ'_L and σ'_M, and of σ''_L and σ''_M, where σ'_L and σ''_L are prefixes of σ_L and σ'_M and σ''_M are prefixes of model traces in $\mathcal{P}_\mathcal{D}$. Let $\psi'_\pi = \delta^*_\pi(\psi_{0\pi}, \sigma'_M)$ and $\psi''_\pi = \delta^*_\pi(\psi_{0\pi}, \sigma''_M)$ be the states reached by $\mathcal{A}_\pi$ when replaying σ'_M and σ''_M on it.[4] Alignments γ' and γ'' are equivalent with respect to $\mathcal{D}$, if $\sigma'_L = \sigma''_L$ and, for all $\overline{\pi} \in \Pi$, $\psi'_{\overline{\pi}} = \psi''_{\overline{\pi}}$. We denote this with $\gamma' \sim_\mathcal{D} \gamma''$.*

If two partial alignments γ' and γ'' are equivalent, the cost of the least expensive path to reach a target node (i.e., a complete alignment) from γ' is the same as from γ''. Indeed, since they are equivalent, they both can be extended with the same sequences of alignment moves. In order to get an optimal alignment, it is only necessary to visit one of them, specifically the one with lowest g cost. Therefore, it is possible to prune the sub-trees with roots in the nodes/alignments that do not have to be visited.

[4] Given a state-transition function δ and a symbol sequence $\sigma = \langle s_1, \ldots, s_n \rangle$, $\delta^*(\psi_0, \sigma)$ denotes the recursive application of the state-transition function over a symbol sequence σ starting from state ψ_0, i.e., $\delta^*(\psi_0, \sigma) = \psi_n$ where, for all $0 < i \leq n$, ψ_i is recursively defined as $\psi_i = \delta(\psi_{i-1}, s_i)$.

Theorem 1. *Let $\mathcal{D}$ be a* Declare *model. Let σ_L be a log trace to be aligned. Let $\Gamma_{(\sigma_L,\mathcal{D})}$ be the set of complete alignments of σ_L and $\mathcal{D}$. Let γ' and γ'' be two alignments such that $\gamma' \sim_{\mathcal{D}} \gamma''$ and $g(\gamma') > g(\gamma'')$. For all complete alignments $\gamma' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$, there exists a complete alignment $\gamma'' \oplus \widetilde{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$ such that $g(\gamma' \oplus \widehat{\gamma}) \geq g(\gamma'' \oplus \widetilde{\gamma})$.*

Proof. Let σ'_L be the portion of σ_L aligned by γ' and γ''. Let $\overline{\gamma'} = \gamma' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$ be one of the complete alignments with the lowest cost among the ones that can be obtained by extending γ'. Since $\gamma' \sim_{\mathcal{D}} \gamma''$, alignment γ'' can also be extended with $\widehat{\gamma}$, i.e., $\overline{\gamma''} = \gamma'' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$. $g(\overline{\gamma'}) = k^{\min} \cdot \|\sigma_L\| + \mathcal{K}(\gamma' \oplus \widehat{\gamma})$ and $g(\overline{\gamma''}) = k^{\min} \cdot \|\sigma_L\| + \mathcal{K}(\gamma'' \oplus \widehat{\gamma})$. Suppose that $g(\overline{\gamma''}) > g(\overline{\gamma'})$ with $g(\gamma') > g(\gamma'')$. If this holds, $\mathcal{K}(\gamma'' \oplus \widehat{\gamma}) = \mathcal{K}(\gamma'') + \mathcal{K}(\widehat{\gamma}) > \mathcal{K}(\gamma' \oplus \widehat{\gamma}) = \mathcal{K}(\gamma'') + \mathcal{K}(\widehat{\gamma})$. Therefore, $\mathcal{K}(\gamma'') > \mathcal{K}(\gamma')$ and, hence, $g(\gamma'') > g(\gamma')$, which is a contradiction. Therefore, there is a complete alignment $\overline{\gamma''}$ obtained by extending with cost lower or equal to $g(\overline{\gamma'})$, i.e., the lower bound of the costs of all complete alignments obtained by extending γ'.

We maintain a set Γ_V of nodes $\overline{\gamma}$ that have already been visited and their costs $g(\overline{\gamma})$. When a new node-alignment γ' is encountered, we check whether it is a candidate node to be visited, i.e., whether its successors need to be added to the priority queue of nodes to be visited. Node γ' is a candidate if for every node $\gamma'' \in \Gamma_V$ equivalent to γ' $g(\gamma'') > g(\gamma')$. It is also a candidate if there is no equivalent node in Γ_V.

5 Provided Diagnostics

This section details some advanced diagnostics that we build on top of the optimal alignments that have been returned for all traces in the event log. First, we indicate why an optimal alignment includes a certain move in log/model: in fact, such a move was introduced to solve a violation of a constraint that occurred in the log trace. Second, we provide a helicopter view that allows one to determine which activities are mostly involved in deviations and which constraints are more often violated. On this concern, we provide metrics to measure the "degree of conformance" of single activities and constraints in a Declare model against the entire event log, in addition to simply evaluating the fitness level (Definition 4) of each log trace against the entire Declare model.

5.1 Why Do I Need This Move?

Let $\gamma = \langle (a_1^L, a_1^P), \ldots, (a_n^L, a_n^P) \rangle$ be an optimal alignment of σ_L and $\mathcal{D}$. Let $\mathcal{A}_\pi$ be the constraint automaton for $\pi \in \Pi$. For each move $(a_i^L, a_i^P) \in \gamma$ in log or in model of an alignment (i.e., s.t. either $a_i^L = \gg$ or $a_i^P = \gg$), we indicate which constraint(s) in the Declare model the move aims to solve. For this purpose, we build an execution trace σ_i obtained from $\langle a_1^P, \ldots, a_{i-1}^P, a_i^L, a_{i+1}^P, \ldots, a_n^P \rangle$ by removing all $\gg$. Then, for each constraint $\pi \in \Pi$, we check whether σ_i is accepted by $\mathcal{A}_\pi$. If it is not accepted, (a_i^L, a_i^P) has been introduced to solve a violation in π. Note that, a move in log or in model always solves at least one violation.

Example 1 (cont.). *Let us again consider the optimal alignment γ_1 (see Page 87). It contains two moves in log or in model: $(LIC, \gg)$ and $(\gg, SQ)$. For $(LIC, \gg)$, we build the execution* *trace* $\sigma_1 = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$*. This sequence is accepted by all the constraint automata in the* Declare *model, apart from the constraint automaton for the* co-existence *constraint (see Fig. 2a). Similarly, for* $(\gg, SQ)$, $\sigma_5 = \langle \checkmark, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$ *is accepted by all the constraint automata, apart from the constraint automaton for the* precedence *constraint (shown in Fig. 2b). Therefore,* $(LIC, \gg)$ *has been introduced to solve a violation in the* co-existence *constraint and* $(\gg, SQ)$ *has been introduced to solve a violation in the* precedence *constraint.*

5.2 Degree of Conformance

We denote with $MC_\gamma(\pi)$ the metric representing the number of moves in model and in log of a complete alignment γ that contribute to solve a violation of π. For $a \in \Sigma$, we denote with $MM_\gamma(a)$ the number of a moves in model, with $ML_\gamma(a)$ the number of a moves in log and with $MB_\gamma(a)$ the number of a moves in both model and log. $MC_\gamma(\pi)$, $MM_\gamma(a)$, $ML_\gamma(a)$ and $MB_\gamma(a)$ can be used to quantify the degree of conformance.

For reliability, we average over all optimal alignments. Let $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ be the set of the optimal alignments of a log $\mathcal{L} = \{\sigma_1, \dots, \sigma_n\}$ and $\mathcal{D}$. The *degree of conformance* of $a \in \Sigma$ with respect to Γ is defined as follows:

$$\mathsf{DConf}_\Gamma(a) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \frac{MM_\gamma(a) + ML_\gamma(a)}{MM_\gamma(a) + ML_\gamma(a) + MB_\gamma(a)}.$$

$\mathsf{DConf}_\Gamma(a) = 1$ if the moves that involve a are only moves in both (i.e., there are no deviations related to a). $\mathsf{DConf}_\Gamma(a)$ decreases with the fraction of moves in model or in log. $\mathsf{DConf}_\Gamma(a) = 0$ if all moves that involve a are only moves in log or moves in model.

Given a constraint $\pi \in \Pi$, the degree of conformance π with respect to Γ is defined as follows:

$$\mathsf{DConf}_\Gamma(\pi) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \frac{MC_\gamma(\pi)}{\|\gamma\|}.$$

$\mathsf{DConf}_\Gamma(\pi) = 1$ if π is never violated. $\mathsf{DConf}_\Gamma(\pi)$ decreases towards 0 as the fraction of moves in model and in log needed to solve violations of π increases.

6 Implementation and Experiments

To check the conformance of Declare models, we have implemented two plug-ins of ProM, a generic open-source framework for implementing process mining functionality [18]. The first plug-in is the *Declare Replayer* that takes as input a Declare model and an event log and, using the algorithm described in Section 4, finds an optimal alignment for each trace in the event log. Starting from the results of the *Declare Replayer*, a second plug-in, the *Declare Diagnoser* generates a map based on the diagnostics described in Section 5.

Section 6.1 reports some experiments to analyze the performance of our approach. Then, Section 6.2 presents our plug-ins and illustrates how diagnostics are graphically

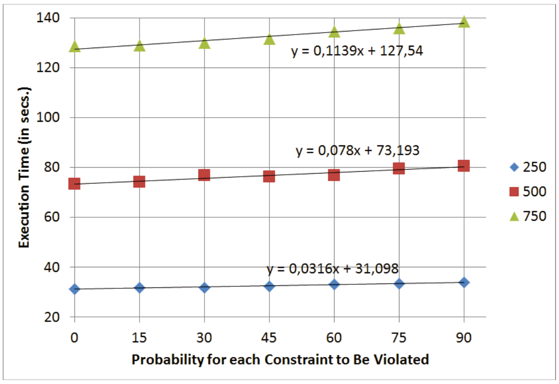


Fig. 3. Results of the experiments conducted on synthetic logs with different combinations of sizes and degrees of non-conformance

visualized in a map. Most of the experiments in Section 6.1 use synthetic logs. Nevertheless, we have also validated our approach on a real case study in the context of the CoSeLoG project[5] involving 10 Dutch municipalities.

6.1 Performance Experiments

To carry out our experiments, we have generated a set of synthetic logs by modeling the process described in Example 1 in CPN Tools (`http://cpntools.org`) and by simulating the model. We use logs with different degrees of non-conformance. A degree of non-conformance of 90% means that each constraint in the log is violated with a probability of 90%. Note that multiple constraints can be violated at the same time in a log trace. In our experiments, the logs have varying degrees of non-conformance: $0\%, 15\%, 30\%, 45\%, 60\%, 75\%$ and 90%. For all degrees of non-conformance, we use randomly generated logs including $250, 500$ and 750 instances to verify the scalability of the approach when varying the log size. The experiments have been conducted on a dual-core CPU at 2.40 GHZ.

Fig. 3 shows the execution times for the *Declare Replayer* plug-in (implementing the A* algorithm) for the different logs. For each combination of log size $(250, 500$ or $750)$ and degree of non-conformance (x-axis), the figure plots the average value of the execution time over 5 runs. The three lines show the trends for the three different log sizes. Fig. 3 illustrates the scalability of the algorithm.

Table 1 shows the effect of pruning the state space and of the heuristics. If we do not prune the search-space employing the technique described in Section 4.2, the *Declare Replayer* has to visit 41% of extra nodes and, consequently, the execution time increases by 33%. Table 1 also shows a dramatic reduction in time and nodes achieved by the heuristics described in Section 4.1. Without using the heuristics, the needed amount of memory increases from 300 MBs to 3.5 GBs.

[5] `http://www.win.tue.nl/coselog`

Table 1. Comparison of the execution time of the *Declare Replayer* when all optimizations are enabled with respect to the cases when they are selectively turned off. The results refer to a log with 250 instances and a degree of non-conformance of 15%.

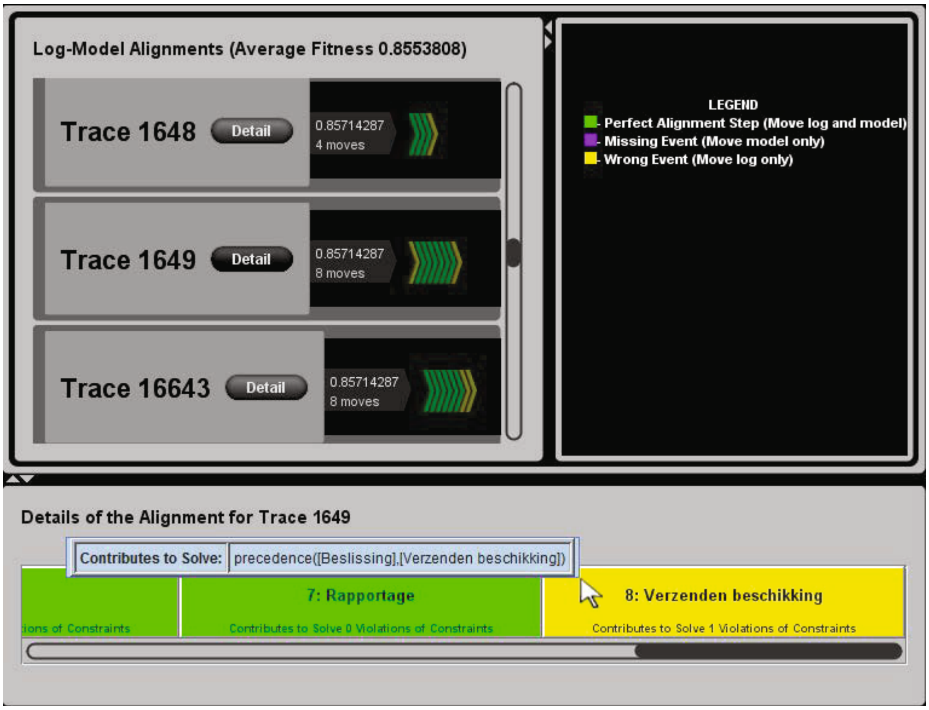
Employed Technique	Visited Nodes	Tree Size	Execution Time
Optimized A*	21	181	30 seconds
Without pruning	30	252	40 seconds
Without heuristics	123419	1204699	ca. 7 hours

We have also performed various experiments using real-life event logs from the CoSeLoG project. For the validation reported here, we have used two logs of processes enacted by two different Dutch municipalities. Process instances in these event logs refer to permissions for building or renovating private houses. We have first discovered a Declare model using an event log of one municipality using the *Declare Miner* plug-in in ProM [14,13]. Then, using the *Declare Replayer*, we have checked the conformance of the mined model with respect to an event log of the second municipality, where every deviation is assigned the same cost/weight. Analysis showed commonalities and interesting differences. From a performance viewpoint the results were also encouraging: 3271 traces with 14338 events could be replayed in roughly 8 minutes, i.e., 137 milliseconds per trace. Section 6.2 provides more details about this experiment.

6.2 User Interface and Diagnostics

Fig. 4 illustrates the output produced by the *Declare Replayer*. The screenshot shows an analysis used in the context of a case study involving two municipalities. An event log of one municipality is compared with a Declare model learned from an event log of another municipality.

Each sequence of triangles in Fig. 4 refers to an alignment of a trace with respect to the model. Each triangle is a different alignment move; the color of the move depends on its type (see the legend on the right-hand side), i.e., move in log (yellow), move in model (purple) or move in both (green). Each sequence is also associated with a number that identifies the fitness level of the specific trace. A button *Detail* is also associated to each trace; it allows us to show the alignment details at the bottom (e.g., for trace 1649 in the screenshot). Each rectangle represents a different move and is annotated with the activity involved in the move. Also here, the color of the move depends on its type, i.e., move in log (yellow), move in model (purple) or move in both (green). In case of moves in log or in model, when moving with the mouse over the rectangle, the *Declare Replayer* shows which constraint violation it aims to solve, in line with the diagnostics described in Section 5.1. In the figure, for trace 1649, the 8th alignment move concerns a move in log for *Verzenden beschikking*. This move has been introduced to solve a violation in a precedence constraint modeling that if activity *Verzenden beschikking* occurs in the log, *Beslissing* must precede, being *Beslissing* not present beforehand. The *Declare Replayer* also provides the average fitness with respect to all log traces (0.8553808). This value indicates that the Declare model mined from the first event log is not fully conforming with the second log, i.e., the two municipalities execute the two processes in a slightly different manner.


Log-Model Alignments (Average Fitness 0.8553808)

Trace 1648 Detail 0.85714287 4 moves

Trace 1649 Detail 0.85714287 8 moves

Trace 16643 Detail 0.85714287 8 moves

LEGEND
🟩 - Perfect Alignment Step (Move log and model)
🟪 - Missing Event (Move model only)
🟨 - Wrong Event (Move log only)

Details of the Alignment for Trace 1649

Contributes to Solve: precedence([Beslissing],[Verzenden beschikking])

ions of Constraints

7: Rapportage
Contributes to Solve 0 Violations of Constraints

8: Verzenden beschikking
Contributes to Solve 1 Violations of Constraints

Fig. 4. A screenshot showing the output of the *Declare Replayer* plug-in. For clarifying, we provide the English translation of the Dutch activity names. *Administratie, Toetsing, Beslissing, Verzenden beschikking* and *Rapportage* can be translated with *Administration, Verification, Judgement, Sending Outcomes* and *Reporting*, respectively.

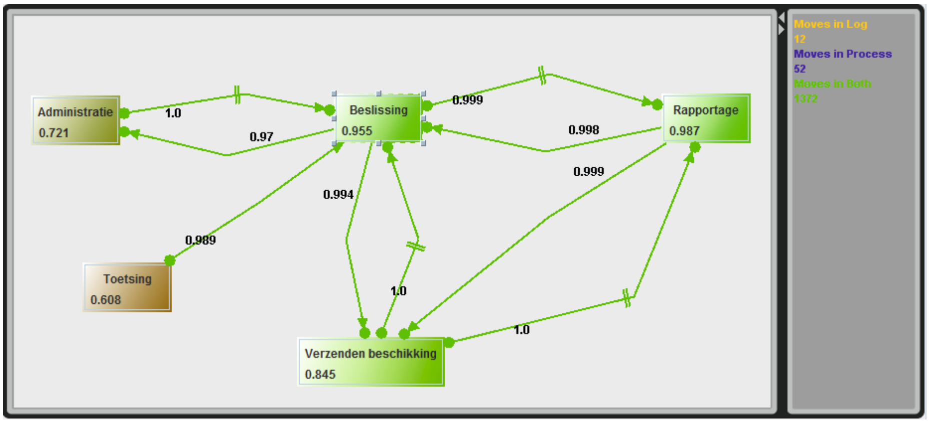


Fig. 5. A screenshot showing the output of the *Declare Diagnoser* plug-in

The results obtained through the *Declare Replayer* can be "projected" on the Declare model by the *Declare Diagnoser* plug-in. The *Declare Diagnoser* plug-in annotates activities and constraints of the Declare model with the degree of conformance. In this way, the process analyst can easily understand where the deviations occur most frequently. Fig. 5 depicts a screenshot of the output of this second plug-in when taking as input the alignments shown in Fig. 4. Activities and constraints are annotated with numbers showing the degree of conformance. To make the visualization more effective also colors are used. Green and red nodes and arcs indicate a degree 1 or 0 of conformance, respectively. Intermediate shades between green and red reflect values in-between these two extremes.

The coloring of activities and constraints in Fig. 5 shows that the level of conformance is reasonable (most parts are close to green). As shown, most of the detected deviations are related to activities *Toetsing* and *Administratie*, which have the lowest degree of conformance ($\mathsf{DConf}_{\Gamma}(Toetsing) = 0.608$ and $\mathsf{DConf}_{\Gamma}(Administratie) = 0.721$). The other activities have degree of conformance close to 1. By selecting an activity, a better insight is provided: in the figure, the selected activity *Beslissing* is involved 12 times in a move in log, 52 times in a move in model and 1372 times in a move in both. The degree of conformance of a constraint indicates whether the constraint is somewhere violated. For instance, $\mathsf{DConf}_{\Gamma}(precedence(Beslissing, Administratie)) = 0.97$ highlights that moves in log and/or in model have been included in some alignments to solve a violation in this constraint. $\mathsf{DConf}_{\Gamma}(not\ succession(Administratie, Beslissing)) = 1$ indicates that this constraint is never violated.

7 Conclusion

This paper presents a novel conformance checking approach tailored towards declarative models. The many conformance checking techniques defined for procedural models (e.g., Petri nets) are not directly applicable to declarative models. Moreover, these techniques tend to provide poor diagnostics, e.g., just reporting the fraction of fitting cases. We adapted alignment-based approaches to be able to deal with the large search spaces induced by the inherent flexibility of declarative models. Based on such alignments we provide novel diagnostics, at the trace level, showing why events need to be inserted/removed in a trace, and at the model level, coloring constraints and activities in the model based on their degree of conformance. As future work, we plan to extend our approach in order to incorporate in our analysis data and resource perspectives.

References

1. 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.) CoopIS/DOA/ODBASE 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D, 99–113 (2009)
3. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. In: IEEE International Enterprise Distributed Object Computing Conference, pp. 55–64. IEEE Computer Society (2011)
6. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
7. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (2011)
8. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM (JACM) 32, 505–536 (1985)
9. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance Checking Between Business Processes and Business Contracts. In: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 221–232. IEEE Computer Society (2006)
10. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Trans. on Knowl. and Data Eng. 18(8), 1010–1027 (2006)
11. Hulstijn, J., Gordijn, J.: Risk analysis for inter-organizational controls. In: Proceedings of the 12th International Conference on Enterprise Information Systems. SciTePress (2010)
12. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and Resource-Aware Conformance Checking of Business Processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 48–59. Springer, Heidelberg (2012)
13. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient Discovery of Understandable Declarative Process Models from Event Logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
14. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, pp. 192–199. IEEE (2011)
15. Munoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: Stability, confidence and severity. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), pp. 184–191 (April 2011)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
17. Vasarhelyi, M., Alles, M., Kogan, A.: Principles of Analytic Monitoring for Continuous Assurance. Journal of Emerging Technologies in Accounting 1(1), 1–21 (2004)
18. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, xESame, and proM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
19. Westergaard, M.: Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 83–98. Springer, Heidelberg (2011)
20. Westergaard, M., Maggi, F.M.: Declare: A tool suite for declarative workflow modeling and enactment. CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)

Context-Aware Compliance Checking

Jan Martijn E.M. van der Werf, H.M.W. Verbeek, and Wil M.P. van der Aalst

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{j.m.e.m.v.d.werf,h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl

Abstract. Organizations face more and more the burden to show that their business is compliant with respect to many different boundaries. The activity of compliance checking is commonly referred to as auditing. As information systems supporting the organization's business record their usage, process mining techniques such as conformance checking offer the auditor novel tools to automate the auditing activity. However, these techniques tend to look at process instances (i.e., cases) in isolation, whereas many compliance rules can only be evaluated when considering interactions between cases and contextual information. For example, a rule like "a paper should not be reviewed by a reviewer that has been a co-author" cannot be checked without considering the corresponding context (i.e., other papers, other issues, other journals, etc.). To check such compliance rules, we link event logs to the context. Events modify a pre-existing context and constraints can be checked on the resulting context. The approach has been implemented in ProM. The resulting context is represented as an ontology, and the semantic web rule language is used to formalize constraints.

Keywords: auditing, compliance checking, process mining, business rules, ontologies.

1 Introduction

Organizations need to ensure that their business stays within boundaries. A *boundary* restricts the organization in its operation. Boundaries may be imposed by all kinds of external sources, like legislation, regulatory bodies, best practices, but also from sources within the organization itself, e.g., based on its corporate culture or management style. These boundaries are often expressed in terms of the business environment, called the *context*. A boundary may restrict the business in *behavior*, in *structure*, or in *data*.

As a consequence, organizations need to constantly monitor their business. The activity of checking whether the business execution adheres to the defined boundaries is called *auditing*. Traditionally, an audit can only provide reasonable assurance that the business is *compliant*, i.e., that the business is executed within the given boundaries. Auditors can only assess the operating effectiveness by checking samples of factual data.

The advance of information systems supporting organizations in their business enables a new form of auditing. The large amount of data recorded by these systems allow to constantly monitor the business execution. *Continuous auditing* [21,25] focuses to bring auditing closer to the operational process, and away from the traditional backward-looking once-a-year examination of financial statements [6].

Continuous auditing is an automated form of auditing that can take all execution data into account [6,11,25]. As such, this form of auditing can provide more assurance. However in order to automate the process of auditing [4,7,16], it first requires the *formalization of the boundaries* in terms of the execution data, and secondly, it can only use context data if an automated link is provided *from the execution data to the context data*. As the boundaries are typically formalized in terms of the context data, and as the automated link from execution data to context data is typically not available, continuous auditing is restricted in its use.

Consider a journal that publishes papers. To support the journal's operation, it may use a system that supports the entire process starting from the submission of a paper to the final publication or rejection. This system typically records everything that happens to a submitted paper into a so-called *event log*. Next to this *Process-Aware Information System* (PAIS) [10], other systems may also collect and store data about this paper. In the remainder of this paper, this other data is referred to as *context data*, while the PAIS data is referred to as *execution data*.

To insure the quality of published papers, a journal needs to discourage undesirable behavior. Therefore, most journals impose a number of *boundaries* on its own operation to guarantee this high quality:

1. For each submitted paper a notification is sent;
2. Each paper needs to be reviewed by at least three different reviewers;
3. The author of a paper cannot be a reviewer of that paper;
4. A reviewer should work at a different affiliation than any of the authors;
5. A reviewer has never been a co-author with any of the authors;

To check whether the journal indeed respects these boundaries, *audits* are used. However, in practical situations, auditing can only be done on a small part of the available data. Hence auditing is incomplete. This may lead, for example, to a paper being rejected by a reviewer which has a conflict of interest, while this conflict does not get noticed.

This paper proposes a link from execution data to context data, which removes some of these restrictions on continuous auditing. As a result of this link, we can bring the execution data to the context data, which allows automated checking of the boundaries and automated linking with other context data.

Please note that although we use a journal to explain the issues at hand, the proposed approach can be used in any other compliance setting. It offers a generic approach for the problem that continuous auditing is restricted to the available execution data.

This paper is organized as follows. Sec. 2 introduces the necessary concepts on which our approach is based. Next, Sec. 3 presents our approach how to link the context data with execution data. Sec. 4 shows the applicability of the approach with a proof of concept, using both existing tools as developing our own. Last, Sec. 5 concludes this paper and hints at possible future directions.

2 Basic Notions

2.1 Business Context

The environment in which an organization like the journal operates is called its *context*. The context is "the combination of all situational circumstances that impact process design and execution" [18], and describes the concepts that influence and bound the business of an organization and the relations between these concepts.

In this paper, we use the notion of a *context model*, which can be viewed as a data model, e.g., using UML class diagrams, the relational database schema or Entity-Relationship diagrams (ERD). A context model defines the *concepts*, their *relationships*, and their *attributes*. Concepts can inherit from other concepts, have different relationships and different attributes. A relationship is from a *source* concept to a *target* concept, where a *cardinality* can be associated to both these concepts. Let $\mathcal{A}$ denote the *attribute name universe*.

Definition 1 (Context model). *A* context model *is a 6-tuple* $(\mathcal{O}, \alpha, \iota, \mathcal{R}, \sigma, \tau)$ *where*

- $\mathcal{O}$ *is a set of* concepts*;*
- $\alpha : \mathcal{O} \rightarrow \mathcal{P}(\mathcal{A})$ *is a function defining for each concept the set of attributes;*
- $\iota : \mathcal{O} \rightharpoonup \mathcal{P}(\mathcal{O})$ *is a partial function defining the inheritance relation. If for a concept* $A \in \mathcal{O}$ *a* $B \in \mathcal{O}$ *exists such that* $B \in \iota(A)$*, we say* B *is a* parent *of* A*. The transitive closure of* ι *has to be irreflexive;*
- $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A} \times \mathcal{O}$ *is a set of* relationships *between concepts such that if* $\{(A, l, B_1), (A, l, B_2)\} \subseteq \mathcal{R}$ *implies* $B_1 = B_2$*. Given a relationship* $r = (A, l, B) \in \mathcal{R}$*,* A *is called the* source *of* r*,* B *is called the* target *of* r*, and* l *is called the* name *of* r*;*
- $\sigma : \mathcal{R} \rightarrow (\mathbb{N} \times (\mathbb{N} \cup \{*\}))$ *defines the* source cardinality *(lower bound and upper bound) of each relationship;*
- $\tau : \mathcal{R} \rightarrow (\mathbb{N} \times (\mathbb{N} \cup \{*\}))$ *defines the* target cardinality *of each relationship.*

For inheritance, we do not allow overriding of attributes and relationships, i.e., for all $B \in \iota^+(A)$ *and* $l \in \mathcal{A}$ *that* $\alpha(A) \cap \alpha(B) = \emptyset$ *and if* $(B, l, \cdot) \in \mathcal{R}$*, then* $(A, l, \cdot) \notin \mathcal{R}$*, where* ι^+ *is the transitive closure of* ι*.*

An example context model for the journal (using an UML class diagram) is depicted in Fig. 1. In this example context model there are six concepts (Affiliation, Author, Journal, Paper, Researcher, and Reviewer), six relationships (authors, is_submitted_to, reviews, works_at, and two unnamed inheritance relations), and nine attributes (abstract, notificationdate, name, number, publicationdate, submissiondate, title, volume, and year). The works_at relation indicates that each

Researcher is connected to exactly one Affiliation and that an Affiliation is connected to multiple (possibly none) Researchers. Likewise, the reviews relation indicates that a Paper is connected to multiple (possibly none, if not under review) Reviewers, and that a Reviewer is connected to at least one Paper, and the authors relation indicates that an Author is connected to at least one Paper, and a Paper to at least one Author. A researcher can both write papers as well as review those. Therefore, the Author and Reviewer concepts inherit from the Researcher concept.

A context model can be *instantiated*, i.e., the context model can be populated with instances of each concept, and associations between these instances, reflecting the relationships defined on the concepts in the model. Let $\mathcal{U}$ denote the *concept instance universe*, i.e., the set of all possible concept instances, and let $\mathcal{V}$ be the *attribute value universe.*

Definition 2 (Instance of a context model). *An instance of a context model* $M = (\mathcal{O}, \alpha, \iota, \mathcal{R}, \sigma, \tau)$ *is a 3-tuple* (I_O, I_A, I_R) *where*

- $I_O : \mathcal{O} \to \mathcal{P}(\mathcal{U})$ *defines the available* concept instances*. Each concept has a, possibly empty, set of instances;*
- $I_A : (\mathcal{U} \times \mathcal{A}) \to \mathcal{P}(\mathcal{V})$ *defines for each concept instance and attribute the corresponding* set of attribute values*;*
- $I_R : \mathcal{R} \to \mathcal{P}(\mathcal{U} \times \mathcal{U})$ *is the set of* associations *between concept instances.*

Let $A, B \in \mathcal{O}$ *be two concepts, let* $l \in \alpha(A)$ *be an attribute of* A*, and let* $(A, r, B) \in \mathcal{R}$ *be a relationship. Given an instance* $I = (I_0, I_A, I_R)$ *of* M*, we write* $a \in A$ *for* $a \in I_O(A)$*,* $a.l = v$ *for* $((a, l), v) \in I_A$ *and* $(a, b) \in r$ *for* $(a, b) \in I_R((A, r, B))$*. The set of all instances of* M *is denoted by* $\mathcal{I}(M)$*.*

Given two instances $I_1 = (I_{O,1}, I_{A,1}, I_{R,1})$ *and* $I_1 = (I_{O,2}, I_{A,2}, I_{R,2})$*, we define their union by* $I_1 \oplus I_2 = (I_{O,1} \oplus I_{O,2}, I_{A,1} \oplus I_{A,2}, I_{R,1} \oplus I_{R,2})$ *where* $f \oplus g : V \to \mathcal{P}(O)$ *with* $(f \oplus g)(v) = f(v) \cup g(v)$ *for all sets* V*,*O*, functions* $f, g : V \to \mathcal{P}(O)$*, and* $v \in V$*.*

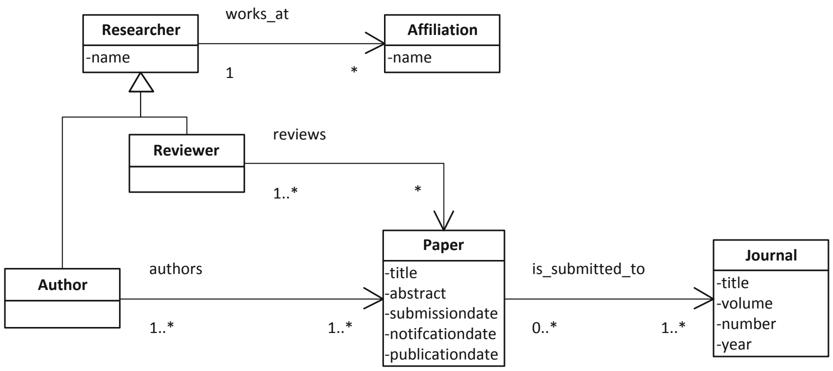


Fig. 1. Context model of the review process context

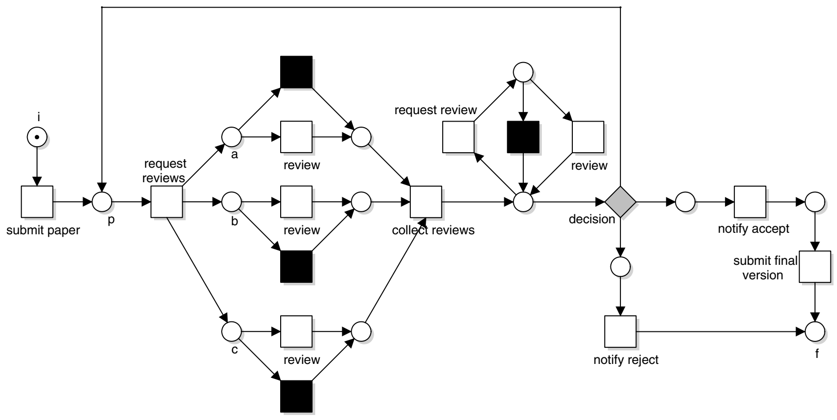


Fig. 2. The review process

Please note that, in principle, an attribute has a set of values, instead of just a single value. For example, a paper will have a set of authors, and not just a single author. However, many attributes will have a singleton set as value. In such cases, we allow the value of the attribute to be the single element of that set instead of the set itself. As a consequence, for a concept instance a and an attribute label l, $a.l$ can either be a set of values ($a.l \in \mathcal{P}(\mathcal{V})$), or the value of its only element in case of a singleton set ($a.l \in \mathcal{V}$). From the context, it will typically be clear which interpretation is used.

An instantiation of a context model is called *consistent* if all the constraints enforced by the cardinalities and inheritance relations are satisfied.

2.2 Process-Aware Information Systems

Business processes form the heart of any organization. A business process can be seen as a set of interdependent tasks and resources needed to produce some product or to deliver some service. In modeling and describing a business process, context plays a natural role. As an example, the names of the activities and resources in the business process are typically aligned with the context in which the process operates. In our example, the journal has a review process that defines for a paper the tasks and resources needed in order to publish the paper in the journal, as depicted in Fig. 2.

In this process model, authors may submit a paper, which will be reviewed by three researchers. Based on the review outcomes, the editor decides whether the paper is accepted, a revision is needed, or whether the paper is rejected, and notifies any accept/reject decision to the authors. Some reviewers may forget to review, or are too late, which is modeled as a skip, using the black transitions. The editor can decide to request an additional review, up to the point that he has sufficient information to make a proper decision.

More and more information systems are developed and implemented to support an organization like the journal in executing their business. A Process Aware Information System (PAIS) [10] assists an organization in the execution of its business process. The business process is then used to configure the system.

A PAIS records each and every step in the business process. For example, the PAIS of the journal records when a paper is submitted, when the reviewers are invited, and whether and when the reviews have been returned. As a result, a PAIS generates enormous amounts of execution data.

For each user action on the system, an *event* is raised. An event records its type, for which activity it has been raised, for which *case* or business process instance, when it was raised, by whom, and the data inserted by the user. Such a recording is called an *event log* [1]. Let $\mathcal{E}$ be the *event universe* and let $\mathcal{C}$ be the *case universe.*

Definition 3 (Event log). *An* event log *is a 3-tuple* $L = (C, E, \#)$ *where*

- $C \subseteq \mathcal{C}$ *is a set of* case identifiers *in the event log;*
- $E \subseteq \mathcal{E}$ *is a set of* event identifiers *in the log;*
- $\# : \mathcal{A} \times (C \cup E) \to \mathcal{P}(\mathcal{V})$ *is an attribute mapping.*

For an attribute $n \in \mathcal{A}$ *we write* $\#_n(\cdot)$ *as a shorthand for* $\#(n, \cdot)$.

Each event belongs to exactly one case, denoted by the mandatory attribute $case \in \mathcal{A}$, *i.e.,* $\#_{case} : E \to \mathcal{P}(C)$ *such that* $|\#_{case}(e)| = 1$ *for all* $e \in E$. *Each case has at least one event, i.e., for all cases* $c \in C$ *an event* $e \in E$ *exists such that* $\#_{case}(e) = c$. *An event may have a successor, denoted by* $next \in \mathcal{A}$, *i.e.,* $\#_{next} : E \to \mathcal{P}(E)$ *such that* $|\#_{next}(e)| \leq 1$ *for all* $e \in E$. *The transitive closure of* $\#_{next}$ *is irreflexive. Each case has exactly one start event, denoted by* $first \in \mathcal{A}$, *and exactly one end event, denoted by* $last \in \mathcal{A}$, *i.e.,* $\#_{first}, \#_{last} : C \to \mathcal{P}(E)$, *such that* $|\#_{first}(c)| = |\#_{last}(c)| = 1$ *for all* $c \in C$, *if* $\#_{next}(e) = \emptyset$ *then* $\#_{last}(\#_{case}(e)) = e$, *and if no event* $p \in E$ *exists with* $\#_{next}(p) = e$, *then* $\#_{first}(\#_{case}(e)) = e$ *for an event* $e \in E$. *The set of all possible event logs is denoted by* $\mathcal{L}$.

An event log represents a period of business execution. Event logs representing consecutive periods of business execution may be concatenated. Suppose event log L' represents the initial business execution, and L'' represents the consecutive business execution. When a case identifier, say c occurs in both L' and L'', it means that this case was not finished in L'. Consequently, all events for c in L'' follow after the last event for case c in L'. In the concatenation of two event logs representing two consecutive periods of business execution, these cases are concatenated: the first event of the second log is the next event of the last event for this case in the first log. Note that by definition of event logs, the sets of event identifiers of two consecutive event logs need to be disjoint. Further, remark that the functions $\#_{\text{first}}$ and $\#_{\text{last}}$ return singleton sets, i.e., these functions are always defined and return a single event.

Definition 4 (Concatenation of event logs). *Let* $L' = (E', C', \#')$ *and* $L'' = (E'', C'', \#'')$ *be two event logs such that* E' *and* E'' *are disjoint. The* concatenation *of* L' *with* L'', *denoted by* $L'; L''$, *results in a new event log* $(C, E, \#)$ *with*

Table 1. Event log of journal reviewing process

Case	Activity	Resource	Time stamp	Data
118	submit paper	system	24-12-2011 17:00:12	title: "Title paper 118", author: 192 author: 193
119	submit paper	system	24-12-2011 17:05:49	title: "Title paper 119", author: 194
...	...	...	...	...
118	request reviews	editor	29-12-2011 10:19:23	reviewer: 112 reviewer: 149 reviewer: 195
119	request reviews	editor	29-12-2011 10:22:43	reviewer: 112 reviewer: 149 reviewer: 195
118	review	149	07-01-2012 16:39:21	verdict: accept
...	...	...	...	...

$C = C' \cup C''$, $E = E' \cup E''$ and $\# = (\#' \setminus \{((\mathit{last}, c), e) \mid c \in C' \cap C'', e \in E'\}) \cup (\#'' \setminus \{((\mathit{first}, c), e) \mid c \in C' \cap C'', e \in E''\}) \cup \{((\mathit{next}, \#'(\mathit{last}, c)), \#''(\mathit{first}, c)) \mid c \in C' \cap C''\}$.

An example of an event log is shown in Tbl. 1. This example represents a small part of an event log generated by the journal review system. It shows for example the submission of papers 118 and 119 with one and respectively two authors, the editor who is requesting reviews for these papers, and a returned review for paper 118 by reviewer 149.

In an event log, we may group the events per case or business process instance. In this way, the events form *execution traces* per case. For example, from the partial event log in Tbl. 1, we have for case 118 the partial execution trace "submit paper", "request reviews" and "review".

The example event log in Tbl. 1 already shows the tight relation between the context and the business execution. For example, solely based on the event log, resource 149 is meaningless, whereas in an instance of the context model, it can be related to some researcher and his affiliation.

2.3 Auditing

Organizations like the journal need to constantly monitor their business to assure that they stay within their boundaries. The activity of checking whether the business execution adheres to the defined boundaries is called auditing [4,7].

Traditionally, an audit can only provide reasonable assurance that the business is *compliant*, i.e., that the business is executed within the given boundaries. As an audit is typically a manual chore, auditors can only assess the operating effectiveness by checking samples of factual data [7].

To audit a system, the auditors may place *process controls* to assess the boundaries, such that the control reports the violation of a boundary [9,14,19,20]. Only if a control is not in place or if it is not functioning as expected, the auditor will typically check factual data. In order to audit a PAIS that supports the review process, a control may be placed counting the number of different reviews per submitted paper. A second control may monitor the time between submission and notification, in order to check the boundary on the time to review a paper.

It is important to realize that a process control may not be sufficient for checking a boundary. Consider, for example, boundary 5, which states that a reviewer may not be a co-author. A control could check, at the moment of the review, whether the reviewer is a co-author, but it will not check this again in the future. Thus, if in the very near future some paper appears in some journal from which it is clear that the reviewer actually was co-authoring a paper, then the control would miss this fact.

Placing controls already increases the assurance an auditor can give. However, adding controls changes the business processes implemented in a PAIS. Hence, if either the business process or a boundary changes, both systems need to be changed. Therefore, process analysis techniques can help in separating process controls from the control flow [17].

3 Context-Aware Continuous Auditing

Since information systems more and more record their usage in event logs, the execution data can be used to check for compliance. Whereas traditional audit only relies on sample-based checks, *continuous auditing* [7, 21] focuses on the analysis of the execution data to perform an audit.

The omnipresence of execution data, coupled with process mining [1] techniques, allows the auditor to perform an audit on the whole business execution, rather than only sample-based [3,4,7,16]. However, boundaries are typically defined on the context data, and not solely based on the execution data [5, 17]. As a result, to be able to check these boundaries, continuous auditing requires both execution data as well as context data. Only if the boundaries are formalized, algorithms and techniques can be developed to automate the process of the auditor. Consequently, not only the boundaries need to be formalized, also the context needs to be modelled.

3.1 Formalization of Boundaries

One way to formalize a boundary is by constraining the concepts in the context model by using the right cardinalities between concepts. In this way, one can easily formalize the boundary that a Paper has at least one Author. It is enforced by the cardinalities of the relationship authors. However, not every boundary can be formalized using only the cardinalities.

Dependencies over different concepts cannot be expressed by local constraints, like the cardinalities, only. For example, the boundary that a Reviewer and Author should be of a different Affiliation cannot be expressed by local constraints. For this reason, we use a constraint language on the context model in first-order logic. The aforementioned boundary can then be expressed by the following non-local constraint:

$$\begin{aligned}
&\forall p \in \text{Paper}, a \in \text{Author}, r \in \text{Reviewer}, w_1, w_2 \in \text{Affiliation} : \\
&\quad (\ (a,p) \in \text{authors} \wedge (r,p) \in \text{reviews} \wedge \\
&\quad\quad (a,w_1) \in \text{works\_at} \wedge (a,w_2) \in \text{works\_at} \\
&\quad)\ \implies w_1 \neq w_2
\end{aligned}$$

This constraint formalizes that in any instance of the context model, the affiliations of an author and a reviewer of the same paper should be different.

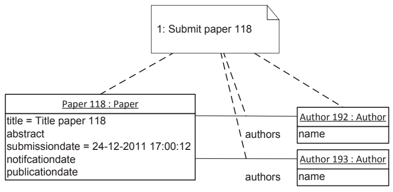
3.2 Relating the Process Execution with the Context

By formalizing the boundaries, compliance checking becomes checking whether each constraint is satisfied by the business execution. However, the constraints are expressed using the context data, while the execution is expressed using the execution data. As a consequence, we need to link both data: Either we transform the constraints in terms of the execution data, or the event log needs to be transformed in terms of the the context data. For example, the rule that a paper has been reviewed by three different reviewers, can be expressed like "Each execution trace contains at least three activities named 'review' executed by different resources". However, many constraints, like the one that corresponds to boundary 5 ("A reviewer has never been a co-author of any of the authors"), are global constraints, i.e., constraints over a set of traces, rather than local constraints over a single execution trace. As a consequence, replay techniques [2] cannot be used, as these only consider single execution traces. Such constraints cannot be transformed into a constraint in terms of the execution data, as the co-author relation does not only depend on the execution data, as authors may decide to submit papers to other journals as well. Clearly, this influences the co-author relation, while these submission will not be stored in the PAIS of the journal at hand.

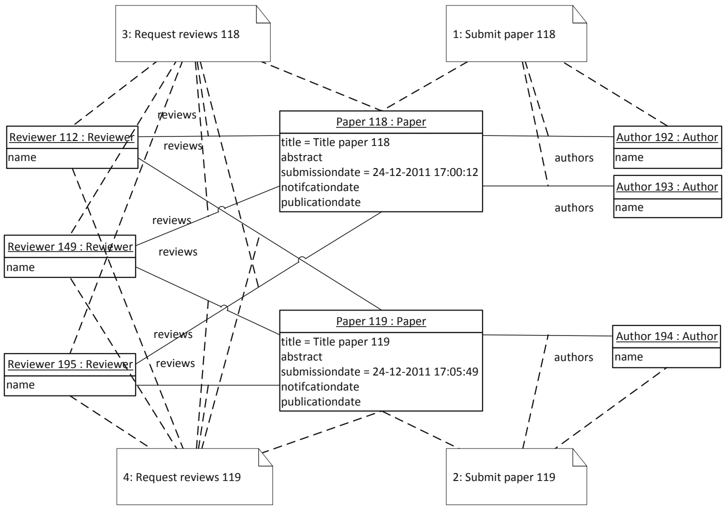
The context model describes the concepts and their relationships. An instance of the context model describes a state of the business. By execution a business process, the state of the business changes, and hence, the instance of the context model. For example, executing the "submit paper" activity corresponds to the insertion of a Paper concept in the current instance. Likewise, the "request reviews" activity corresponds in the insertion of new associations between one instance of Paper and several instances of Reviewer.

Each event in an event log corresponds to an update of the current state of business. As a consequence, executing an event can be seen as a function from one instance of the context model to a new instance. We denote with $\mathcal{I}(M)$ the universe of instances of context model M.

Definition 5 (Transformation function). *Given a context model M, a* transformation function *transforms a given context model instance and an event log into a new instance of the context model, i.e, a function $f : \mathcal{I}(M) \times \mathcal{L} \to \mathcal{I}(M)$ such that for $I \in \mathcal{I}(M)$ and $L_1, L_2 \in \mathcal{L}$ we have $f(I, \emptyset) = I$ and $f(I, L_1; L_2) = f(f(I, L_1), L_2)$.*



(a) After first step



(b) After four steps

Fig. 3. Replaying the event log of Tbl. 1 on the context

In this way, we are able to replay the event log in terms of the context. As each event updates the context, we define an additional function that transforms a context model instance and an event log into a new context model instance. Let us consider the journal example again. Let I be a consistent instance of the context model depicted in Fig. 1, let L be the current event log, and let e be the event that corresponds to the submission of paper 118. The event causes the creation of the Paper 118 object, the Author 192 and Author 193 objects, and it creates the author relation from these Author objects to the Paper object. We can formalize this to an update of the context model instance:

$$f(I, (\{c\}, \{e\}, \#)) = \begin{cases} I \oplus (e_O, e_A, e_R) & \text{if } \#_{\text{name}}(e) = \text{`}\textit{submit paper}\text{'}; \\ I & \text{otherwise,} \end{cases}$$

where

$$\begin{aligned} e_O &= \{\mathit{Paper} \mapsto \{c\}, \mathit{Author}, \mathit{Researcher} \mapsto \#_{\text{author}}(e)\}, \\ e_A &= \{(c, \mathit{title}) \mapsto \#_{\text{title}}(e), (c, \mathit{submissiondate}) \mapsto \#_{\text{timestamp}}(e)\}, \\ e_R &= \{(\mathit{Paper}, \mathit{authors}) \mapsto \{(c, a) \mid a \in \#_{\text{author}}(e)\}\}. \end{aligned}$$

Note that the creation of an object is only necessary if the object does not already exist. If, for example, the object for Author 192 would have already existed, then the existing object would be reused. Likewise, the event that corresponds to the other submission creates the Paper 119 object, the Author 194 object, and the authors relation between both. Fig. 3 shows the result of replaying the first events from Tbl. 1 on an empty journal context model instance.

Similar functions can be created for the other events. The first request review event creates the Reviewer 112, Reviewer 149, and Reviewer 195 objects, and the reviews relations to the Paper 118 object, whereas the second request review reuses the Reviewer objects and creates reviews relations from these objects to the Paper 119 object. As a result of replaying the event log in the context model, we have now extended the context model with the execution data. Hence, we can use the execution data as if it were context data.

3.3 Compliance Checking

The first step in checking compliance, is the check whether the initial instance of the context satisfies each of the constraints. After this check, we can start to replay the event log. For this, we need to sort all events in the event log based on their time stamp, so that the context is updated in the same order as the business has been executed.

While replaying the events on the context model, after each update the current instance needs to be checked to see whether the constraints are still satisfied. If in a step a constraint is violated, we need to report this event and the violation. An auditor then needs to test the severity of the violation. For example, if after completing the trace, the constraint is repaired, the impact of the violation may be less than if the constraint remains violated.

In this approach, we only allow the event log to change the context. In reality, not only the business execution changes the context, also external events, like researchers that change affiliation, or people that write papers for different journals, may change the context. These external changes to the context are (for now) ignored by our approach.

4 Implementation

As a proof of concept, we implemented the approach in the process mining toolkit ProM 6 [22]. In literature, ontologies are often used to capture context (cf. [12,13,18]). An ontology can be seen as a collection of concepts with associations between these concepts [23]. Further, ontologies allow for modularization using the import concept. In this way, different ontologies can be combined into a single context model. For example the context of the journal could be split into an ontology for the publications and reviews, and a separate ontology for the researchers and their affiliation.

4.1 Event Logs as Ontologies

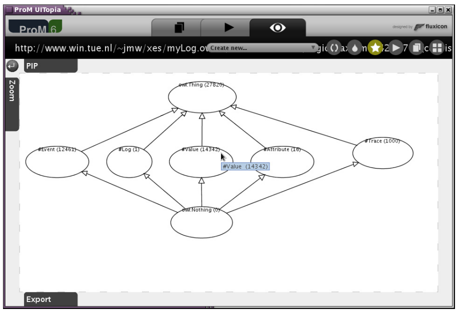
Not only the context model can be represented in an ontology. Also event logs can be represented in an ontology. This allows us to reason over both the context as well as the process execution in a single formal representation. The XES standard [22] defines the important concepts in an event log, and their relationships. In the XES standard, a Log uses zero or more Extensions that each define a set of Attributes. An Attribute has a set of values, represented by the Value concept in the ontology. The concepts Log, Event, Trace and Value are subclasses of the Attributable concept, meaning that they can have Values attached. Each Trace belongs to a Log, and has a start event and an end event. Events belong to a Trace, and each can have a predecessor and a successor Event. The ontology also stores the transitive closure of the predecessor and successor relation, i.e., the set of all predecessors or successors, respectively, of an Event.

The *Ontologies* package of ProM 6 provides an automatic transformation from event logs into the ontology format. For a log of the journal with 1000 cases and 12461 events, the transformation results in an ontology with 455.990 axioms, i.e., concept, individuals, and relations between these individuals.

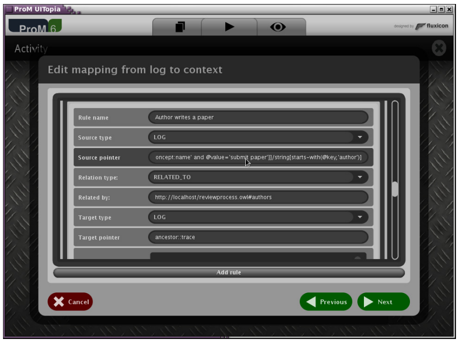
4.2 Linking Events to the Ontology

As seen in the previous section, each event belongs to an update of the context model. For example, the event 'submit paper' creates a paper. In the formalization, we represented this by stating that each trace in the event log creates an instance of the concept Paper. In terms of ontologies, this means that each trace of this event log is not only an individual of the concept Trace, but also of the concept Paper. Hence, we want to add this relation between the ontology of the event log and the ontology of the context model. To automate the process of relating the different individuals, we introduce the notion of annotation rules.

An *annotation rule* relates individuals between the ontologies of the event log and of the context model. It defines a source element, a target element and the



(a) Event log as ontology in ProM 6



Rule name	Author writes a paper
Source type	LOG
Source pointer	oncept:name' and @value='submit paper']]/string[starts-with(@key,'author')]
Relation type:	RELATED_TO
Related by:	http://localhost/reviewprocess.owl#authors
Target type	LOG
Target pointer	ancestor::trace

(b) Annotation rule in ProM 6

Fig. 4. Implementation of context models as ontologies in ProM 6

relation between the source and target, like inheritance relation, instantiation, and an ontology object or data property. If the relation between source and target is an ontology property, we also need to specify which property is used.

To select elements from the event log, we use XPath, which allows us to select multiple elements in an event log. For example, the rule that each trace is a paper, can be expressed by the following annotation rule, assuming that the Paper ontology has the namespace `http://localhost/publication.owl`:

AR 1 (A paper) Creates an instance of the concept 'Paper' for each trace

source	`//trace`
relation	instance of
target	`http://localhost/publicatizon.owl#Paper`

This rule selects each trace (`//trace`) in the event log, and adds the corresponding Trace individual to the concept Paper. As the transformation also specifies that a paper has a title, we need another annotation rule to express this.

AR 2 (A paper has a title) The title is specified in the activity 'submit paper' as an attribute named 'title'

source	`//trace`
relation	has data property: `http://localhost/publication.owl#title`
target	`./event[/string[@key='concept:name' and @value='submit paper']]/string[@key='title']@value`

This rule iterates over each trace, as specified in the source of the annotation rule, and gets the corresponding title by retrieving the attribute 'title' of its 'submit paper' event. In case multiple elements are selected by the target XPath query, then a relation is created for each of the selected elements. In this way, we need to create five annotation rules for the transformation of the 'submit paper' event: one to add the Trace to the Paper concept (*AR 1*), two to add the title (*AR 2*) and a similar rule for the submission date of the paper, one to create the authors, and one to relate the authors to the paper.

Fig. 4(b) depicts an annotation rule as it is implemented in ProM 6. Similar annotation rules can be created for each of the events. In this way, we get an ontology that connects the ontology of the event log with the ontology of the context model. This connecting ontology serves as input for the compliance checking.

4.3 Compliance Checking

The approach results in three ontologies: one ontology being the initial context model instance, one ontology representing the event log and one that provides the connection between these two. In order to check whether the organization stayed within its boundaries, we need to check whether all three ontologies together satisfy these boundaries. The previous section showed how each of the boundaries can be formalized into a set of constraints expressed in first order logic. Next step is to express these constraints in a language that can be used on ontologies.

One such language is the Semantic Web Rule Language (SWRL) [8]. SWRL rules are expressed in terms of an ontology. Hence, we can create a separate ontology containing these rules.

For example, to express the rule that states that for each paper no reviewer may be a coauthor of one of the authors of that paper, we first add a data property to each paper, e.g., *violatesBound5*, with domain Paper and the Boolean values as range. Next, we can express when this rule is true: if a paper has been submitted earlier by both a reviewer and an author of the current paper. Expressed in SWRL:

$$\begin{aligned} \mathit{violatesBound5}(?p, \texttt{true}) \leftarrow\ & \mathit{Paper}(?p) \wedge \mathit{reviews}(?r, ?p) \wedge \mathit{authors}(?a, ?p) \\ & \wedge\ \mathit{Paper}(?y) \wedge \mathit{authors}(?a, ?y) \wedge \mathit{authors}(?r, ?y) \\ & \wedge\ \mathit{submitdate}(?y) < \mathit{submitdate}(?p) \end{aligned}$$

As ontologies are being used to express both execution data and context, other formalisms can be exploited to express constraints, like SBVR [15].

Important to realize when working with ontologies is the *open world assumption.* In the open world assumption, a statement is either true, false, or unknown, whereas in a closed world a statement is either true or false. This is an important difference. For example, in the rule above we can only express when the statement is true. If no witness is found, in the closed world assumption the constraint is satisfied, whereas in the open world assumption, the statement results in unknown.

To increase the expressivity of the constraints, the closed world assumption is needed. For example, a logic language like Prolog would be a natural next step, as the translation from ontologies to Prolog is straightforward [24].

5 Conclusions

All kinds of sources enforce boundaries on the way an organization runs its business. Typically, these boundaries are phrased in terms of the environment of the business, called the context. In this paper, we proposed a novel approach to support the auditor in checking whether the organization stays within its boundaries.

Mostly, the work of an auditor is manual. As more and more organizations are supported by information systems that record their usage in event logs, more and more data becomes available to automate the work of the auditor. The enormous amounts of data available allows the auditor to use techniques like process mining.

In order to automate the compliance checking, also the boundaries need to be formalized. In this paper, we propose context models. However, as event logs are in terms of the system, and the boundaries in terms of the context, we argue the need of transforming the information available in event logs into information available in terms of the context.

The approach presented in this paper is a next step towards continuous auditing. To show the applicability of the approach we implemented it in ProM 6 using ontologies as a context model. Although ontologies provide a powerful mechanism to reason over the context, more research is needed to further automate the task of the auditor.

Current replay techniques only visualize control flow related aspects, like conformance checking, of a business execution. The approach proposed in this paper allows to replay the business execution in its context. In this way, business analysts and auditors have the possibility to inspect how business execution changes the business environment.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Kumar, A., Verdonk, M.: Conceptual Model for Online Auditing. Decision Support Systems 50(3), 636–647 (2011)
4. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Verdonk, M.: Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. IEEE Computer 43(3), 102–105 (2010)
5. Accorsi, R., Stocker, T.: On the Exploitation of Process Mining for Security Audits: The Conformance Checking Case. In: ACM Symposium on Applied Computing. ACM (2012)
6. Alles, M.G., Kogan, A., Vasarhelyi, M.A.: Putting Continuous Auditing Theory into Practice: Lessons from Two Pilot Implementations. Journal of Information Systems 22(2), 195–214 (2008)
7. Chan, D.Y., Vasarhelyi, M.A.: Innovation and Practice of Continuous Auditing. International Journal of Accounting Information Systems 12(2), 152–160 (2011)
8. World Wide Web Consortium. SWRL: A Semantic Web Rule Language Combining OWL and RuleML (2011), http://www.w3.org/Submission/SWRL/
9. Haworth, D.A., Pietron, L.R.: Sarbanes-Oxley: Achieving compliance by starting with ISO 17799. Information Systems Management 23(1), 73–87 (2006)
10. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. John Wiley & Sons, Inc. (2005)
11. Elliot, R.K.: Assurance Service Opportunities: Implications for Academia. Accounting Horizons 11(4), 61–74 (1997)
12. Filipowska, A., Kaczmarek, M., Kowalkiewicz, M., Markovic, I., Zhou, X.: Organizational Ontologies to Support Semantic Business Process Management. In: International Workshop on Semantic Business Process Management, pp. 35–42. ACM (2009)
13. Fox, M.S., Barbuceanu, M., Gruninger, M.: An Organisation Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour. Computers in Industry 29(1-2), 123–134 (1996); WET ICE 1995

14. 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)
15. Goedertier, S., Mues, C., Vanthienen, J.: Specifying Process-Aware Access Control Rules in SBVR. In: Paschke, A., Biletskiy, Y. (eds.) RuleML 2007. LNCS, vol. 4824, pp. 39–52. Springer, Heidelberg (2007)
16. Jans, M., van der Werf, J.M.E.M., Lybaert, N., Vanhoof, K.: A Business Process Mining Application for Internal Transaction Fraud Mitigation. Expert Systems with Applications 38(10), 13351–13359 (2011)
17. Ramezani, E., Fahland, D., van der Werf, J.M., Mattheis, P.: Separating Compliance Management and Business Process Management. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part II. LNBIP, vol. 100, pp. 459–464. Springer, Heidelberg (2012)
18. Rosemann, M., Recker, J.C., Flender, C.: Contextualisation of Business Processes. Int. Journal of Business Process Integration and Management 3(1), 47–60 (2008)
19. Green, S.: Manager's Guide to the Sarbanes-Oxley Act: Improving Internal Controls to Prevent Fraud. Wiley (2004)
20. Sadiq, S., 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)
21. Vasarhelyi, M.A., Halper, F.: The Continuous Audit of Online Systems. Auditing: A Journal of Practice & Theory 10(1), 110–125 (1991)
22. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
23. W3C. OWL 2 Web Ontology Language (2009)
24. Wielemaker, J., Schreiber, G., Wielinga, B.: Prolog-Based Infrastructure for RDF: Scalability and Performance. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 644–658. Springer, Heidelberg (2003)
25. Williams, B.C.: Auditing and recent Developments in IT. Managerial Auditing Journal 7(5), 18–25 (1992)

Measuring Privacy Compliance Using Fitness Metrics[⋆]

Sebastian Banescu[1,2], Milan Petković[1,2], and Nicola Zannone[2]

[1] Philips Research Eindhoven
{sebastian.banescu,milan.petkovic}@philips.com
[2] Eindhoven University of Technology
n.zannone@tue.nl

Abstract. Nowadays, repurposing of personal data is a major privacy issue. Detection of data repurposing requires posteriori mechanisms able to determine how data have been processed. However, current a posteriori solutions for privacy compliance are often manual, leading infringements to remain undetected. In this paper, we propose a privacy compliance technique for detecting privacy infringements and measuring their severity. The approach quantifies infringements by considering a number of deviations from specifications (i.e., insertion, suppression, replacement, and re-ordering).

Keywords: Process compliance management, Security in business processes.

1 Introduction

Privacy protection is becoming a major issue in society nowadays. Advances in ICT allows the collection and storage of a huge amount of personal data. Those data are a valuable asset for business corporations. For instance, they can be used to provide customized services and create effective, personalized marketing strategies. However, such practices might be intrusive limiting customers' right to privacy.

The need to protect personal data has spurred the definition of several privacy policy languages. These languages usually extend access control with the concept of purpose to provide a more strict control on access and usage of data. Purpose is used to represent the reason for which data can be collected and processed. Data are usually labeled with the intended purposes and can be accessed only if the purposes for which data are requested is compliant with the label attached to the requested data. However, this approach is preventive and only guarantees policy compliance by preventing infringements to occur. Therefore, it is unacceptable when access to data should be granted to provide critical services such as medical treatment in emergency situations.

Recent privacy legislation recognizes the need of more comprehensive solutions to privacy and has an increased emphasis on accountability. The demand of technical means to enforce legislation requires the development of novel auditing systems which are able to ensure compliance with privacy policies and make users accountable for their actions. The development of such systems requires methods for determining whether the actual usage of personal data is compliant with the intended purpose.

[⋆] This work is funded by the Dutch national program COMMIT through the THeCS project and by the European Commission through the FP7 TClouds project (nr. 257243).

A first step to enable purpose control is to provide semantics to purposes, which defines how data can be used. Business process has been proved to be a suitable formalism for the definition of semantic purpose models [1]. This formalism makes it possible to verify user behavior against the intended purpose of data by verifying whether the user actions recorded in a log correspond to a valid trace of the process model corresponding to the intended purpose [1]. This approach to purpose control is more suitable for critical systems because it does not prohibit access to data in emergency situations in contrary to the preventive approaches. However, the corresponding existing solutions are only able to determine that a deviation occurred, but they are not able to identify particular deviations and to assess their privacy severity.

This paper proposes a conformance metric to detect privacy infringements and quantify their severity. In particular the conformance metrics is able to identify four different types of local deviations (i.e., insertions, suppressions, and replacements) and non-local deviations (i.e., re-ordering). Moreover, it adds a transition in isolation for each log event that has no corresponding transition in the process model. Therefore, the algorithm accepts a large range of logs as input. The detected infringements are then quantified using the privacy metric proposed in [2].

2 Privacy Compliance Checking

Deviations between an event log and a process model can have different types. Each type of deviation has a different privacy severity. In this paper, we consider four types of deviation: *insertion* occurs when an activity is executed, but its execution is not allowed by the process model; *suppression* occurs when an activity whose execution is required by the process model is not executed; *replacement* occurs when an activity is executed instead of another activity; *re-ordering* occurs when the order in which activities are executed is different than the order of the same activities specified in the process model.

To identify deviations from the specification, we propose a conformance checking technique based on [3]. In particular, the event log is replayed over the process model to identify user behavior which is not compliant with the intended behavior. Non compliant behaviors are represented in terms of *remaining* tokens (i.e., tokens that were left in the model after the execution of the process) and *missing* tokens (i.e., tokens created artificially for the successful execution of the process).

Conformance checking in [3], however, suffers from drawbacks in terms of accuracy [4]. First, the metric is sensitive to the process structure and does not consider all paths allowed by the process model. In addition, it does not identify the type of deviation that occurred for the measurement of fitness. Moreover, [3] assumes that all events in the log must have a counterpart in the process model, and every node must be on some path from the start to the end of the process. This assumption is too restrictive for our purposes. Indeed, we have to consider cases, for instance, where a user performed a task which he was not allowed to perform.

To overcome these drawbacks, we revise the technique in [3] in several ways. We associate a counter to tokens to represent the time in which an event occurred. This (together with the distinction between consumable and produced tokens) allows the definition of patterns for the identification of the type of deviation (Section 2.4). In addition, we will add activities in isolation to the process model. Finally, conformance checking is performed on all alternative paths rather than on the entire process model.

The privacy compliance checking proposed in this paper comprises of five phases:

1. *Pre-processing* phase, in which alternative paths in the process model are identified;
2. *Simulation* phase, in which the event log is replayed in the process model;
3. *Pairing* phase, in which behaviors not compliant with the model are detected;
4. *Deviation identification* phase, in which non compliant behaviors are categorized according to the type of deviation;
5. *Quantification* phase, in which deviations are analyzed and quantified.

2.1 Pre-processing Phase

The pre-processing phase takes as input a business process represented as a CPN model (we refer to [5] for the notation for CPN models) and identifies alternative paths in the model. In particular, this phase consists of a non-disjoint "partitioning" of the process model into sub-processes such that each sub-process contains only paths defined in the original model and no two sub-processes contain the same path. The following phases of the algorithm are executed on each identified sub-process.

2.2 Simulation Phase

The goal of the simulation phase is to replay an event log λ over a CPN model N. To facilitate the pairing phase (Section 2.3) and the identification of the type of deviation (Section 2.4), we distinguish the set of consumable tokens (denoted by $Cons$) and the set of produced tokens (denoted by $Prod$). Tokens have the form of $(p, c)@\ell$, where p is the place containing the token and c is the token color. Each token has a counter ℓ associated to it, which is used to represent the sequence in which events occur.

The replay of λ starts with the initial marking. Then, logged events are fired one after another in N. Differently from [3], in our simulation approach, tokens are not consumed by transitions. Instead, every event e in λ leads to the generation of one consumable token in every place in $\bullet t$ and one produced token in every place in $t\bullet$, where t is the transition in e. If t does not belong to N, it is not performed by the intended role, or accessed data are not as intended, a new activity representing the executed event is added to the model in *isolation* (i.e., unconnected from the rest of the model). The output of the simulation phase is a new process model obtained by replaying λ over N together with marking $Cons \cup Prod$.

2.3 Pairing Phase

The aim of the pairing phase is to detect non conformant behaviors that occurred in the execution of the process model. The basic idea is to identify proper information flow by pairing produced and consumed tokens generated in the simulation phase. Intuitively, a consumable and a produced token match if (1) they are located in the same place, (2) they have equal token color, and (3) the counter of the produced token is lower or equal than the one of the consumable token. Paired tokens are removed from the marking. The set of tokens that do not have a counterpart (hereafter called *unpaired tokens*) corresponds to non conformant behaviors. Unpaired tokens are used in the next phases to identify the type of deviation and quantify its severity.

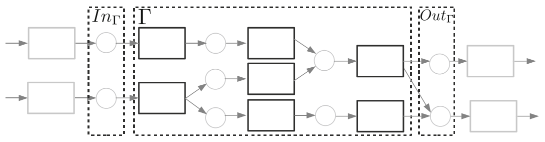


Fig. 1. Graphical representation of grouping

2.4 Deviation Identification Phase

The aim of this phase is to identify the type of deviations that occurred. In particular, the deviation type is identified using patterns based on unpaired tokens. Before presenting the patterns and the identification process, we introduce the following notation.

Definition 1. *Let $N = (P, T, A, C, E, \mathbb{M}, \mathbb{Y}, M_0)$ be a CPN and $\Gamma \subset T$ a set of transitions. The* input boundary *of Γ, denoted by In_Γ, is the set of places $\Pi \subset P$ s.t. for every $p \in \Pi$, $(p, t) \in A$ iff $t \in \Gamma$, and $(t, p) \in A$ iff $t \notin \Gamma$.*

Definition 2. *Let $N = (P, T, A, C, E, \mathbb{M}, \mathbb{Y}, M_0)$ be a CPN and $\Gamma \subset T$ a set of transitions. The* output boundary *of Γ, denoted by Out_Γ, is the set of places $\Pi \subset P$ s.t. for every $p \in \Pi$, $(t, p) \in A$ iff $t \in \Gamma$, and $(p, t) \in A$ iff $t \notin \Gamma$.*

Definition 3. *Let $N = (P, T, A, C, E, \mathbb{M}, \mathbb{Y}, M_0)$ be a CPN. Given a set of transitions $\Gamma \subseteq T$ with input boundary In_Γ and output boundary Out_Γ, we say that Γ forms a* grouping *on N iff for all $p_x \in In_\Gamma$ and $p_y \in Out_\Gamma$ there exists a path from p_x to p_y s.t. all the transitions on the path belong to Γ.*

Fig. 1 represents a grouping together with its input and output boundaries. The grouping is delimited by a dashed rectangle. The places on the left and right sides of the dashed rectangle form the input and output boundaries of the grouping, respectively. Note that input and output boundaries do not have to be disjoint sets.

Deviation Identification Patterns. In this section, we define four patterns based on the configuration of unpaired tokens for the identification of *insertions*, *suppressions*, *replacements* and *re-orderings*.

Insertion Pattern. A set of transitions $\Gamma \subset T$ is an insertion if Γ is a grouping with input and output boundaries In_Γ and Out_Γ respectively, and the following conditions hold: (1) every place $p_i \in In_\Gamma$ contains a token $(p_i, c_i)@\ell_i \in Cons$; (2) every place $p_j \in Out_\Gamma$ contains a token $(p_j, c_j)@\ell_j \in Prod$; (3) for all i and j, $\ell_i \leq \ell_j$; (4) all places $p \in \{\bullet t \cup t \bullet | t \in \Gamma\} \setminus (In_\Gamma \cup Out_\Gamma)$ do not contain any token.

Suppression Pattern. A set of transitions $\Gamma \subset T$ is a suppression if Γ is a grouping with input and output boundaries In_Γ and Out_Γ respectively, and the following conditions hold: (1) every place $p_i \in In_\Gamma$ contains a token $(p_i, c_i)@\ell_i \in Prod$; (2) every place $p_j \in Out_\Gamma$ contains a token $(p_j, c_j)@\ell_j \in Cons$; (3) all places $p \in \{\bullet t \cup t \bullet | t \in \Gamma\} \setminus (In_\Gamma \cup Out_\Gamma)$ do not contain any token.

Replacement Pattern. A set of transitions $\Gamma_S \subset T$ are replaced by a set of transitions $\Gamma_I \subset T$ if the following conditions hold: (1) Γ_S satisfies the conditions of the suppression pattern with tokens $(p_i, c_i)@\ell_i \in \mathit{Prod}$ s.t. $p_i \in \mathit{In}_{\Gamma_S}$, and tokens $(p_j, c_j)@\ell_j \in \mathit{Cons}$ s.t. $p_j \in \mathit{Out}_{\Gamma_S}$; (2) Γ_I satisfies the conditions of the insertion pattern with tokens $(p_k, c_k)@\ell_k \in \mathit{Cons}$ s.t. $p_k \in \mathit{In}_{\Gamma_I}$ and tokens $(p_l, c_l)@\ell_l \in \mathit{Prod}$ s.t. $p_l \in \mathit{Out}_{\Gamma_I}$; (3) for all i, j, k and l, $\ell_i < \ell_k \leq \ell_l < \ell_j$;

Re-ordering Pattern. A set of transitions $\Upsilon \subset T$ are re-ordered if the order in which transitions are executed is different than the order of the same transitions in the process model. A *re-ordering* is characterized by the existence of places $p \in P$ such that p contains tokens $(p, c)@\ell_i \in \mathit{Prod}$ and $(p, c)@\ell_j \in \mathit{Cons}$ s.t. $\ell_j < \ell_i$.

Pattern Identification Process. Deviations are identified in two steps: first, insertions, suppressions and re-orderings are identified; then, replacements. Pattern detection starts from the token with the lowest counter in the marking. If more than one token have such a counter, a consumable token is chosen. The process model is traversed to identify (a set of) tokens which, in the combination with the selected token, satisfy a pattern. Identified patterns are stored into a *deviation list* Δ. A deviation in Δ is a tuple (δ, G, C, R) where δ is the deviation type associated to the pattern, and G, C and R are the sets of transitions, consumable and produced tokens that satisfy the pattern, respectively.

After an insertion or a re-ordering has been identified, the tokens forming the pattern are removed from the marking. After a suppression has been identified, the consumable tokens associated to it are used to fire the suppressed transitions. Such tokens flow in the process model until they reach a place containing a produced token associated to the currently identified suppression. If the counter of the consumable token is higher or equal to the one of the produced token, then the tokens are removed from the marking; otherwise, they indicate that a re-ordering also occurred, and such a deviation is added to the deviation list. To determine which transitions have been re-ordered, it is necessary to analyze the event log and token counters. Let p be a place that contains two tokens satisfying the re-ordering pattern. The corresponding re-ordered activities are identified by determining the longest path from p in the model such that it only contains transitions corresponding to events which occurred in the interval defined by the counters of the tokens used to identify the deviation.

If the selected token, in combination with (possibly) different sets of tokens, satisfies the conditions of more than one pattern, a copy of the deviation list for each pattern is created together with the corresponding marking; each identified pattern is stored in a separate deviation list. Then, the process continues on each marking and deviation list. This step ends when no insertion, suppression or re-ordering can be found.

Replacements are identified by checking deviations lists. In particular, a replacement is identified if there exist an insertion and a suppression in the deviation list such that the insertion occurred within the suppression. If such a pattern is detected, the corresponding insertions and suppressions are removed from the list, and a replacement is added to the same list.

2.5 Quantification Phase

The quantification phase aims to measure the severity of the identified privacy infringements (if the severity is 0, no infringement occurred). The severity assessment is performed on each deviation list associated to a log using the metric d_L^{Φ} presented in [2]. This metric uses a number of factors (e.g., accessed data, executed action, user role) to measure the privacy distance between the intended and actual user behavior (see [2] for details). Every deviation list Δ is traversed separately, and d_L^{Φ} is applied to each deviation in the list. The *insertion* of a sequence of events $\xi \subset E$ is quantified against the empty sequence ϵ, i.e. $d_L^{\Phi}(\epsilon, \xi)$. The *suppression* of a sequence of activities $\chi \subset \mathcal{A}$ is quantified against an empty sequence ϵ, i.e. $d_L^{\Phi}(\chi, \epsilon)$. The *replacement* of a set of (suppressed) activities $\chi \subset \mathcal{A}$ by a sequence of (inserted) events $\xi \subset E$ is quantified as $d_L^{\Phi}(\chi, \xi)$. The *re-ordering* of a sequence of events Υ is quantified by considering the data accessed by events which occurred before they were supposed to have occurred. Let $\xi \subset \Upsilon$ be the sequence of events which occurred before they were supposed to have occurred. Re-ordering is quantified as $d_L^{\Phi}(\xi, \xi')$ where ξ' is equal to ξ except that the user's knowledge about data is the one specified in the process model.

3 Conclusions

In this paper we have proposed a privacy compliance technique for identifying privacy infringements and quantifying their severity. In particular, we have defined patterns for identifying four types of deviations, namely insertions, suppressions, replacements and re-orderings. The proposed technique has been implemented as a ProM 6 plug-in.

We are planning to extend this work by considering silent transitions, which are necessary for the analysis of OR splits. The variety of deviations considered in this work poses the basis for assisting privacy auditors in the investigation of privacy violations. Future work includes the development of a user-friendly auditing tool which allows the visualization of violations and automated generation of privacy assessments.

References

1. Petković, M., Prandi, D., Zannone, N.: Purpose Control: Did You Process the Data for the Intended Purpose? In: Jonker, W., Petković, M. (eds.) SDM 2011. LNCS, vol. 6933, pp. 145–168. Springer, Heidelberg (2011)
2. Banescu, S., Zannone, N.: Measuring privacy compliance with process specifications. In: Proceedings of Int. Workshop on Security Measurements and Metrics, pp. 41–50. IEEE (2011)
3. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. In: Proc. of EDOC 2011, pp. 55–64. IEEE (2011)
5. Lakos, C.: Composing Abstractions of Coloured Petri Nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 323–342. Springer, Heidelberg (2000)

Event-Driven Manufacturing Process Management Approach

Antonio Estruch and José Antonio Heredia Álvaro

Department of Industrial Systems Engineering and Design, Universitat Jaume I
12071 Castellón, Spain
{estruch,heredia}@uji.es

Abstract. Timely insight into manufacturing processes events can help in improving its efficiency and agility. Events are state change in process execution that can be not only monitored but correlated and managed in order to take immediate action. In this paper a new approach to develop event driven manufacturing process management solutions is presented. The event-driven architecture is considered as the basis to design a unified modeling methodology which enables near real time event stream processing. The approach adapts Business Process Management to manufacturing at the different automation levels (ISA-95 levels 2, 3 and 4). The key issue is to model the logic of complex events in manufacturing processes. The use of BPMN 2.0 is proposed as the standardized modeling language. Through this notation a standardized definition of business logic for monitoring and controlling manufacturing operations can be developed, representing the knowledge to apply in order to increase process performance.

Keywords: Business Process, Complex Event Processing, Business Activity Monitoring, BPMN.

1 Introduction and Motivations

Manufacturing related processes cover from high level plant scheduling, material use, production, delivery and shipping activities, to low level monitoring and control of shop-floor activities. In this context, it is often needed to apply decision logic on the fly to detect particular situations and handle them in consequence. In response to these occurred events, the supporting IT solution must help to carry out appropriate automated actions that can require attention of users at different levels of responsibility according to the specific manufacturing organization. The purpose of this paper is to present the Event Driven Manufacturing Process Management (EDMPM) as a novel approach to manage manufacturing processes in cases where is required a highly integration with existing IT manufacturing solutions.

The structure of this paper is as follows: In the section 2 the proposed approach is explained. The proposed solution general schema following the presented approach is shown in section 3. Section 4 illustrates the approach applied to a sample manufacturing scenarios including process models. Finally, section 5 concludes and gives an outlook of findings and future work.

2 Proposed Approach

The EDMPM approach is elaborated from the analysis of complex events occurring in the manufacturing systems, and the identification of the need for a method to express the knowledge about how to handle them.

2.1 Manufacturing Events Analysis

As some authors reviewed before [1, 2, 3], in manufacturing operations, events occur constantly. From sales order creation, ordered products production starts and finishes, materials arrive at the loading dock, machines can reports failures, quality control or statistical process control systems notify about out-of-bounds conditions, and so on. Determining if a manufacturing event is important and how must be handled varies and depend on the complexity of each organization and even on the specific manufacturing industry. Table 1 shows sample situations on the different levels that that are considered in this research.

Simple events normally are handled directly by supervisor systems that monitorize them, but normally these event occurrences are not thrown again to be handled by other upper level systems. However, in others scenarios more elaborated procedures

Table 1. Sample manufacturing complex events application at diferent levels

	Process Control	Manufacturing Execution	Enterprise Management
Quality	• Control Service Level Agreement conformity of machine performance parameters.	• Detect production line failure when consecutive bad finished products are found.	• Detect quality problem when a number of returned products in last month is greater than a specific value.
Production	• Control number operations performed to detect the end of the machining process.	• Control the number of pieces processed to detect lots finalization. • Detect production rescheduling need when some machines fail.	• Control the number of finished lots to detect production end. • Detect delayed needed materials for manufacturing.
Maintenance	• Monitorize machine working parameters to detect out of normal limits.	• Predictive maintenance from product line machine workload. • Control corrective maintenance SLA	• Detect not performed preventive maintenance tasks. • Detect increasing maintenance operations to consider machine renewal.
Energy	• Monitorize energy consumption peaks to predict machining tool changes. • Monitorize energy consumption to turn off machine components when idle too much time.	• Control the energy consumption deviation of machining processes. • Automated turn on/off manufacturing equipment when it is not used.	• Control that Plant energy savings plans are executed on time. • Detect unexpected increases in the organization energy consumption.

for complex event correlation detection logic could be implemented using previously simple event occurrences to detect particular situations that in other way will happen unnoticed. This case use to be more relevant when involved events are originated at different manufacturing levels. For example, a process that manage a scheduled production with time constraints must take attention of any situation that could alter the expected production plan to take special actions to ensure expected delivery times. The occurrence of undesired situations could be inferred from product lots bad finished information, which could be inferred from particular out of control low level process measures.

All of this is part of the manufacturing knowledge to be managed, including particular data acquisition requirements, simple event detection and complex event correlation logic, and finally procedures about how to operate as consequence. This involves people involvement and automated actions beyond manufacturing plant scope including business level activities and sometimes decisions taken by high level staff.

2.2 EDMPM Based Manufacturing IT Solution

The use of manufacturing related IT solutions could represent many benefits with reduction of failures and related costs, obtaining better resources usage and response time. Standardization efforts like ANSI ISA-95 standard [4] for enterprise control systems provides standard terminology and object models used by manufacturing related IT solutions to decide which information, should be managed and exchanged among different levels of the proposed automation pyramid. We are interested on addressing scenarios where is required to detect and correlate events occurrences that can come from different ISA-95 levels. Therefore, the required IT solution conform a new kind of EMI (Enterprise Manufacturing Intelligence) platform supporting communication to existing systems and being able to automate process models representing the knowledge about how each manufacturing scenario must be handled in response to incoming events.

In this context, the EDBPM (Event-Driven Business Process Management) paradigm concepts presented by R. von Ammon and others [5, 6, 7] can be applied. This concept is presented as an enhancement of BPM (Business Process Management) including concepts of SOA (Service Oriented Architecture), EDA (Event Driven Architecture), SaaS (Software as a Service), BAM (Business Activity Monitoring) and CEP (Complex Event Processing). In this paper, inspired by this paradigm we are proposing a new approach, EDMPM, emphasizing the usage of process modeling techniques to express into process models the main concepts of EDA, CEP and BAM without needing to use of several parallel supporting platforms. Furthermore, while a BPM engine can execute these processes models, complex events are detected on the fly and thrown through the BPM system in order to be caught by existing running processes to handle them. At the same time, defined Performance Measures are calculated, emphasizing the streaming processing aspect of this approach without waiting to post process analysis.

The main component of the proposed IT solution will be then the BPM system. Traditionally, BPM system means a software platform, which provides companies the ability to model, manage, and optimize processes. The usage of standards to model how each manufacturing scenario must be handled, describing it in a formally way, helps to communicate across all the organization involved staff how manufacturing operations are done and their implications. This way, common understanding is achieved and future improvement proposals could be obtained from more people, not only particular operation specialists. Different modeling techniques could be applied to model manufacturing related activities, like IDEF0/3 (Integrated Definition Methods) [8], UML (Unified Modeling Language) [9], BPMN (Business Process Modeling Notation) [10], etc. Over time IDEF have been taken over by UML for systems modeling and BPMN for business process modeling. Nowadays, the most used graphical representation for specifying business process in a business process model seems to be the BPMN developed by the BPMI (Business Process Management Initiative), which is maintained by the OMG (Object Management Group) to develop the standard further. The aim of this notation is to provide a modeling language for the business analyst community, which is simple and provide a standard way of process elicitation. Thus, we explored the BPMN capabilities to model manufacturing processes involving complex event handling. The main advantage of using BPMN models would be that these are directly executed by existing BPM engines or indirectly executed being previously mapped to BPEL (Business Process Execution language). Not less important is the BPMN simplicity and familiar flow chart notation that enables both business and IT process modeling participants to understand it.

Modeling CEP with BPMN

Handling scenarios, like manufacturing where a huge number of events are generated and must be processed, require applying more event focused techniques. AS defined by D. Luckham [11], an event is something that happens or is thought of happening in a system and involves information about its origin, timestamp, and other specific event related data. Even though simple events are useful by themselves, in many scenarios, some interesting events are inferred only from a combination of simple events occurrence.

Traditionally, to perform this inference, CEP approaches use techniques like event-pattern detection or running query analysis against a mass of events previously stored. Different Event Processing Languages (EPL) to define or query events can be found in the literature: There are event pattern description languages like Rapide [12] for specifying patterns of events in causal event executions (posets); Other languages are composition-operator-based (COB) to define simple events and a set of operators that are recursively defined on them, conforming a so called “event algebra”. The result of applying these COB operators is a complex event. An example of these languages is SASE [13]; Data stream languages focused on data operations like TelegraphCQ [14] developed at the UC Berkeley, implements a dataflow system for processing continuous queries over data streams that could be used to discover complex event occurrences. Engines for data streaming with a SQL language variant for writing event queries are implemented in open source libraries like ESPER [15]; Production rule languages that were not created originally for event query but use a condition-action rule of the form “WHEN condition THEN action” can also be used to express event queries. An example of these languages is DROLS [16].

Almost all CEP oriented languages only come with textual representations and for that reason are not used by process modelers in graphical languages. At the contrary, modeling complex events evaluation logic expressed using the BPMN modeling notation offers more visibility about the correlation logic behind it and enables best understanding from non-technical users. A. Barros et al. [17] discusses about the requirements for handling complex events in process models and how difficult is to represent some CEP patterns in flow oriented languages like BPMN because its event matching and consumption mechanism. However BPMN 2.0 can avoid, partially, these limitations. This last BPMN version includes new event types like parallel multiple event to match multiple events at the same time, signals for broadcasting, or the conditional event that could include rule based EPLs to match more complex event correlations.

Event-Driven Architecture (EDA) with BPM

EDA is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. In our approach, following the typical EDA event flow, the BPM acts as "event generator" receiving events from the already implemented adapters, and as "event channel" storing these events in event queues to be ready to be consumed by processes. The other typical flow layers of EDA, "event processing engine" and "downstream event-driven activity" are played by the running processes themselves, which incorporate into the models the complex event detection logic and the tasks to be executed in consequence of the event.

How to organize all the complex event management logic into BPMN models is not trivial. Conceptually, the process models can be classified attending to its events usage as event producer, complex event evaluator or event consumer. At the same time, one process could play more than one of these roles.

- *Event Producer Processes*: All BPM running processes potentially could be an event producer process. But not all events, required to detect a complex event, come from other executing process because often these events come from external sources (mainly at ISA-95 level 2).
- *Complex Event Evaluator Processes*: The complex event detection logic should be designed as an isolated process, although not mandatory, that attends incoming events and decides if a complex event occurs and then a message is sent to the interested consumer processes.
- *Complex Event consumers*: These processes attends complex event occurrence and operate in consequence.

The following figure (Fig. 1) shows a picture to clarify these concepts using BPMN symbols and depicts our proposed EDA modeling guideline. For each manufacturing process, which can play one or more of the possible roles, a modeled phase identifies the tasks, gates and caught/thrown events involved. This way, publisher/subscription event propagation mechanism is explicit in the BPMN diagrams.

At runtime, running instances of these process models and/or external adapters can throw events which must be caught by other running instances of complex event evaluator processes. Complex event logic is applied then, and when a complex event occurrence is detected other existing process instances are notified or new ones are created to operate in consequence. In a full event-driven monitoring, manufacturing related KPIs are then updated accordingly and refreshed the result measures dashboards.

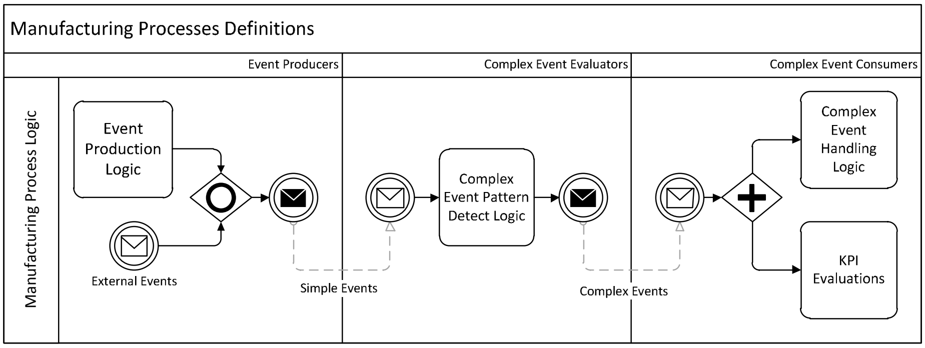


Fig. 1. Event driven process roles

Modeling BAM Related Performance Measures

Manufacturing processes performance measurement is required for its management. Choosing the right Key Performance Indicators (KPI) of a manufacturing processes requires good understanding of what is important to the organization. To help in task some BPM based solutions offers the capability of store logs and information persistence about each performed task or event occurred that can be exploited. BAM discipline attends this topic in depth, using Business Intelligence (BI) related techniques, like data warehousing and data mining that proved its value applied to enterprise data sources, to handle BPM processes tracking data. Not all KPIs can be evaluated directly from this persisted data and often requires access to external data sources and maybe perform special logic that must be implemented separately on external tools or included into the processes models themselves. Over the years much work has been done in field of BAM and related intelligence management tools. For instance, D. Gregori et al. [18] presented a tool suite providing features for analysis, monitoring, prediction, control and optimization of business processes using BI techniques like data mining and data warehouse. This is a sample approach that uses an externally full BI post process execution data processing cycle that penalizes rapid response times.

Our approach proposes to evaluate manufacturing KPIs explicitly, embedding KPIs evaluation logic inside the manufacturing processes logic. In this respect, CEP technologies can be used as a foundation for BAM as some authors had proposed [11]. This way, all the needed information to evaluate real time KPIs could be forwarded by the business process itself, throwing and catching events, forcing KPIs measures update while its execution, instead of being obtained from the BPMS database or other systems. In this context, some authors [19, 20] discuss about implementing workflows oriented to obtain operational measures.

There are clear differences between process oriented BPMN models and classical Extract, Transformation and Loading (ETL) data warehouse cycle. The advantage of the proposed BPMN models approach is that instead of using a post process analysis, the KPI update logic is executed synchronously with the related activities; near real time updates for critical measures are possible. The basic steps required to monitorize a KPI must be expressed in BPMN: data arrivals, filtering, time series addition, computation and analysis.

Figure 2 shows an example BPMN subprocess diagram to monitorize a simple KPI computed from a sensor values. Incoming sensor data are modeled as input messages containing measured data and timestamp information, which must be filtered comparing the sensor value to valid boundary limits, throwing and event in case of out of bounds. The valid data is added to a data sequence and then the KPI is computed using aggregation functions on this data sequence. In parallel, both the KPI trend and the KPI semaphore state are evaluated. If there is a change in one of them, a trend or semaphore change event occurs. These events are thrown to be handled by consumer processes and/or other evaluator processes. Infinite variations of this model can be designed depending on each specific KPI evaluation requirements.

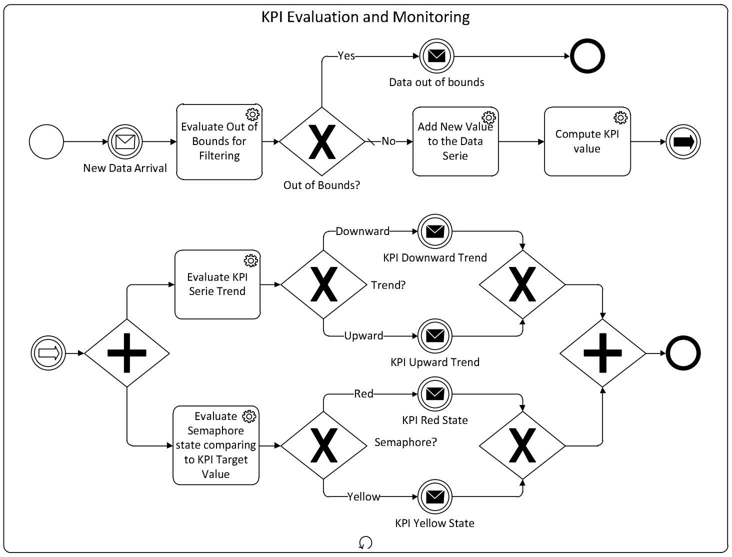


Fig. 2. Sample subprocess diagram to monitorize a simple KPI explicitly

BPMN Extensions for Modeling CEP and KPIs Evaluation

Some other authors propose to extend BPMN standard with special symbols with semantics related to CEP and KPIs concepts and use them instead of BPMN standard symbols. S. Kunz et al. [21] presented an approach for integrating complex events, described in ESPER EPL [15], extending attributes of standard symbols, but requires that ESPER must be integrated into the BPM solution as a rule engine and some special diagram artifacts information must be understood by the BPM execution engine. G. Decker et al. [22] introduce a graphical language for modeling composite events in business processes, called BEMN that could be used to define event rules. In the field of KPIs evaluation models, J. Friedenstab et al. [23] presents a set of graphical symbols extending BPMN to represent BAM relevant concepts like measures, aggregations, targets, filtering, etc., but it is not clear how BPMS engines will understand these symbols and implement the required logic. Further research and standardization efforts seem to be required in this direction.

3 EDMPM Based IT Solution Architecture

Following some of the previously presented CEP techniques some authors [1, 2, 3] propose frameworks to manage manufacturing processes, using the event publisher / consumer mechanism in conjunction with EDA architecture as a basis for an extensible MES (Manufacturing Execution System) solution. The problems with these query based approaches are the mass event storage required, EPL queries engine performance, and mainly its poor extensibility comparing to a BPM based solution where process models can be really easily modified.

In this paper, we propose to exploit EDMPM paradigm that promotes the use process modeling languages to design the manufacturing process models. But the execution of this process models must be supported by an IT solution, which must offer a set of features that must be discussed. The following figure (Fig. 3) shows the proposed approach generic architecture schema for an EDMPM based solution. Three main layers are proposed in this architecture: connectivity layer to enabled communication with existing manufacturing IT solutions using available technology standards in the manufacturing field; an enhanced BPM engine to support direct execution of manufacturing processes modeled with BPMN, enabling an enhanced time window event handling to help CEP correlations evaluation and offering some special predefined tasks to enable data series statistical processing to help KPIs evaluation; and finally the user interface layer that must provide the BPMS user interaction required support plus customizable KPIs visualization and analysis support.

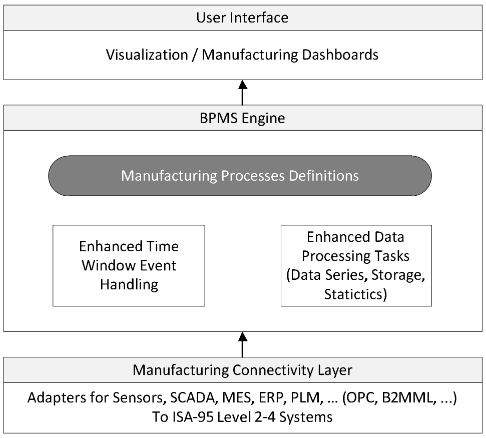

User Interface
Visualization / Manufacturing Dashboards
BPMS Engine
Manufacturing Processes Definitions
Enhanced Time Window Event Handling
Enhanced Data Processing Tasks (Data Series, Storage, Statictics)
Manufacturing Connectivity Layer
Adapters for Sensors, SCADA, MES, ERP, PLM, ... (OPC, B2MML, ...) To ISA-95 Level 2-4 Systems

Fig. 3. EDMPM general architecture schema

3.1 Manufacturing Connectivity Layer

The IT solutions found in the manufacturing field are highly specialized on each level and the use of specific communication standards increase the complexity of the management of all the manufacturing related activities in an integrated way. As manufacturers changed to a global business model this integration is more obviously needed.

Now, manufacturing IT solutions are no longer an island unto itself, but an important component in the organization wheel. This way the solution accommodates the current business environment requiring higher integration and openness.

Therefore, the IT solution must provide adapters implementing standards like OPC (OLE for Process Control) to read process measures or alarms from machine automation controllers at level 2, or B2MML (Business to Manufacturing Markup Language) defined by the WBF (World Batch Forum) according to the ISA-95 standard to interchange information between level 3 and level 4 about production performance results, produced and consumed material, product tracking and tracing information, etc. These adapters transform specialized existing standard protocols messages to BPM ready to consume messages. BPM vendors typically have their own event representation format, but the Workflow Management Coalition had published XML Business Process Analytics Format (BPAF) standard that defines event information representation that could be used for this purpose. BPM based solutions are not EAI (Enterprise Application Integration) tools but they can be used to solve the manufacturing systems integration problems because its SOA nature, but this overhead could impact on system response time and make it unusable for severe time restrictions scenarios like ISA level 2 process control (milliseconds), etc.

3.2 BPM Engine Enhancements

In addition to normal business process execution support including management of all the running processes, notifying users pending tasks, etc., the BPM engine must offer some enhancements to help the normal execution of EDMPM process models. The connectivity adapters and running processes are generating simple events that must remain in the BPM event queues, at least for a period of time that could be specified by an event attribute, avoiding to be lost if some processes, which could be waiting for them, are not waiting at the moment that they were thrown. In addition broadcasting events functionality is a requirement in these scenarios when a single event could be consumed by many processes.

In order to support basic data processing to enable process related data to be stored, filtered, aggregated, etc., some specialized tasks could be required to be offered by the BPM engine itself. In addition, more complex processing tasks like computational intelligence algorithms could help to update prediction models that could be critical in some scenarios to prevent failures that could be identified as warning events and be processed. If all these features are implemented, natively embedded into the BPM engine, overall response time will be reduced, increasing performance.

3.3 Customizable User Interface

In addition to common user interaction with BPM oriented solutions where the execution of pending running process tasks are managed, a more manufacturing management oriented interface is required. In the business contexts, strategic management solutions like Balanced Scorecard (BSC) solution offers real time visualization update especially to show critical business KPIs. The user interface must be able to visualize dashboards including charts, data tables, gauges, semaphores, etc., to visualize KPIs and its related information. Highly customizable dashboard visualization and KPIs analysis is necessary to accommodate user information needs to take the appropriate decisions inside critical processes. This feature could be partially offered by many BPM vendors using their BAM visualization tools or rely on thirdy party solutions for these purposes.

4 Use Case

To illustrate the proposed EDMPM approach, we applied its concepts to a simple manufacturing scenario, considering a production line with various machines. Products must be processed on each machine consecutively. In this fully automated scenario, all the product line is sensor equipped and it is possible to detect some basic situations. Some complex events applications identified on section 1, using events correlation logic implemented in BPMN to detect complex event occurrence, is applied to illustrate how to use our approach in order to manage the control production quality, detect machine failures, maintenance tasks and manufacturing energy consumption.

Several events have to be considered in order to decide if a CEP pattern occurs, considering time window restrictions when it is needed. In order to simplify the BPMN diagrams, the required CEP logic are represented as loop subprocesses where on each iteration a new CEP event is detected and thrown. To simplify, all incoming events are represented as intermediate messages despite there are a lot of event types in BPMN and it is not necessary to include all possible event type combinations in the diagrams.

- *Quality related complex event sample*: The objective is to advise from a possible production line problem where consecutive bad finished products are found. It follows the *Event cardinality pattern*: A specified number of events of the same type that are all subject to the same constraints have to have occurred. As shown in figure 4, initially a counter is zero initialized and two parallel flows are executed. On one hand a reset event is expected to enable the counter to be reset externally. On other hand, every time a quality error event occurs the corresponding counter is increased and its value is compared to a max value. If this max value is reached the CEP event is thrown, otherwise a new quality error event must be expected.

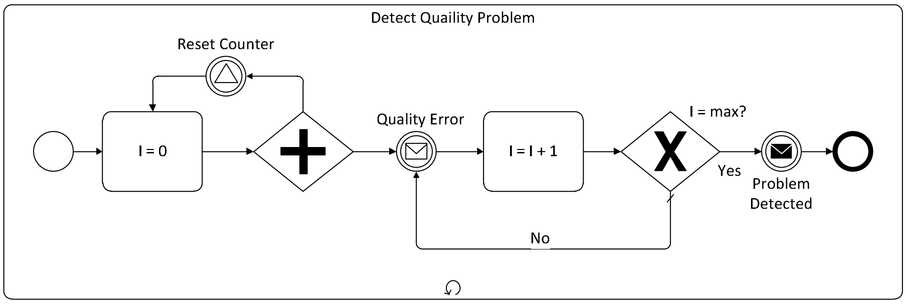


Fig. 4. Detect quality problem expanded subprocess loop

- *Maintenance related complex event sample*: Detection of not performed corrective maintenance tasks according to the maintenance Service Level Agreement (SLA). It follows the *Event – Event Time Relation pattern*: This is a special case where the event of one type always has to have occurred after the other. As shown in figure 5, once a machine problem happens, the maintenance team must apply corrective maintenance under the previously signed SLA. Only if this maintenance is not done in the SLA interval the success event wait subprocess is interrupted and the CEP event is thrown.

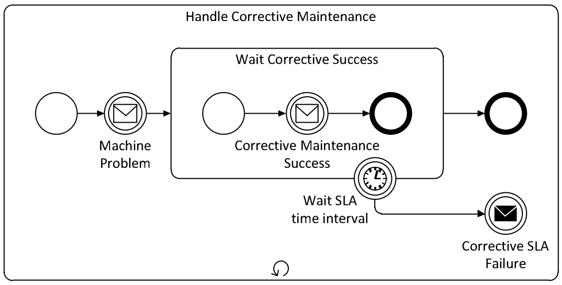


Fig. 5. Detect maintenance SLA failure expanded subprocess loop

- *Energy related complex event sample*: Detection of machine idle state for a period of time to propose turn off components for energy savings. It follows the *Event – Event Time Relation pattern*: This is a special case where the occurrence of a third event cancels the pattern. As shown in figure 6, every time an energy power measure arrives from the energy monitoring device, its value is checked to identify idle state (low values). If idle state is identified and the subprocess "waiting period of time" hasn't started previously then it starts. In the case of the incoming measure is identified as a regular operation value and the waiting subprocess was started previously a signal is thrown to interrupt it. Only in the case of the idle values remains constant on the predefined period of time the CEP turn off proposal event is thrown.

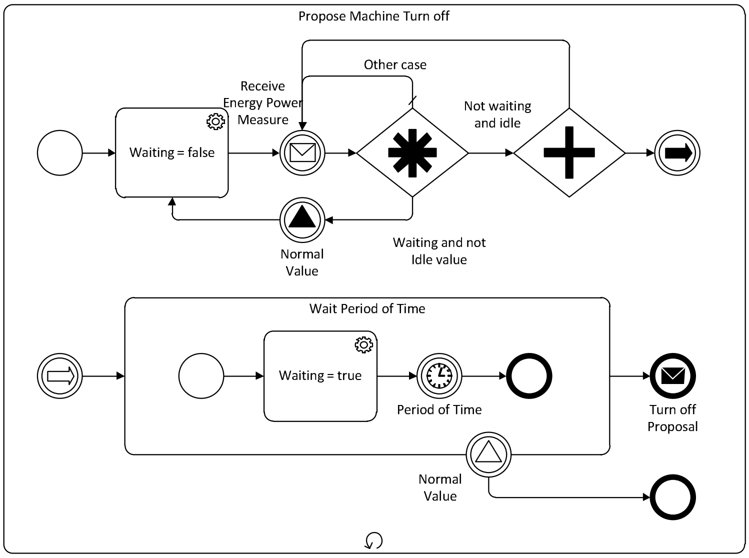


Fig. 6. Propose machine turn off expanded subprocess loop

5 Conclusions and Future Work

Following the proposed EDMPM approach, it is possible to use the widely extended BPMN to express at the same time the main manufacturing process activities, the logic to detect and handle complex events and the related KPIs evaluation. This way, a new level of manufacturing process intelligence is possible, representing the knowledge about how to manage different manufacturing situations, enabling it to be understood by more people from the manufacturing organization.

Solutions following the EDMPM approach can be implemented as a BPM based solution, implementing adapters to manufacturing communication standards to interact with existing manufacturing IT solutions and technologies like OPC and B2MML.

The events managed by the BPM engine must remain in the BPM event queues, at least for a period of time that could be specified in an event attribute, avoiding to be lost if some processes, which could be waiting for them, are not waiting at the moment that they were thrown. In addition broadcasting events functionality is a requirement in these scenarios when a single event could be consumed by many processes. BPMN 2.0 execution capability could help to avoid some of these problems because its more rich catalog of event symbols.

In order to support basic data processing to enable process related data to be stored, filtered, aggregated, etc., some specialized tasks must be offered by the BPM engine itself to increase performance. In addition, more complex processing tasks like data mining or optimization related algorithms could help to update prediction models that could be critical in some scenarios to prevent failures that could be identified as warning events and be processed.

The EDMPM approach could be used to define more integrated manufacturing processes in highly automated scenarios which are promoted by future manufacturing efforts to increase efficiency, flexibility, agility and quality like FuTMaN [24] or ManuFuture [25]. These efforts propose to increase manufacturing automation related activities requiring capturing and managing of an also increasing amount of data and events in contexts where near real time analysis and response could be critical.

In order to validate the proposed approach, some manufacturing scenarios like identified in section 1 could be simulated into plant simulator software capable to interact with a BPM companion engine, generating and receiving events to/from deployed process models to carry out the manufacturing management logic including CEP correlation to identify new complex events and BAM measures update logic to update KPIs values.

As future work, more manufacturing related patterns will be identified and specialized set of tasks will be well defined and classified in the context of EDMPM. It is also the will to identify more useful patterns for CEP and KPI evaluation logic to be embedded into the manufacturing process models. Finally, the proposed approach could be applied to distributed modern enterprise manufacturing environments where processes covering supply chain aspects which require greater communication logic must be defined.

References

1. Zhang, Y.H., Dai, Q.Y., Zhong, R.Y.: An Extensible Event-Driven Manufacturing Management with Complex Event Processing Approach. Int. Journal of Control and Automation 2(3), 1–12 (2009)
2. Walzer, K., Rode, J., Wünsch, D., Groch, M.: Event-Driven Manufacturing: Unified Management of Primitive and Complex Events for Manufacturing Monitoring and Control. In: Workshop on Factory Communication Systems (2008)
3. Janiesch, C., Matzner, M., Müller, O.: A Blueprint for Event-Driven Business Activity Management. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 17–28. Springer, Heidelberg (2011)
4. The Instrumentation, Systems and Automation Society. ANSI/ISA-95.00.01-2000: Enterprise Control System Integration (2000)
5. von Ammon, R.: Event-Driven Business Process Management. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems. Springer, Heidelberg (2008)
6. von Ammon, R., Emmersberger, C., Springer, F.: Event-Driven Business Process Management and its Practical Application Taking the Example of DHL. In: Future Internet Symposium, Vienna (2008)
7. von Ammon, R., Emmersberger, C., Greiner, T., Springer, F., Wolff, C.: Event-Driven Business Process Management. In: Second International Conference on Distributed Event-Based Systems, DEBS 2008 (2008)
8. IDEF, IDEF Family of Methods (2012), `http://www.idef.com`
9. Object Management Group, Unified Modeling Language (UML), OMG (2012), `http://www.uml.org`
10. Object Management Group, Business Process Modeling Notation (BPMN), OMG. Tech. Rep. 2011-01-03 (2012), `http://www.omg.org/spec/BPMN/2.0/PDF`
11. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Munich (2007)
12. Luckham, D., Frasca, B.: Complex Event Processing in Distributed System. Program Analysis and Verification Group Computer Systems Lab Stanford University (1998)
13. Wu, E., Diao, Y., Rizvi, S.: High-performance Complex Event Processing over streams. In: Proc. Int. ACM Conf. on Management of Data (SIGMOD), pp. 407–418. ACM (2006)
14. Chandrasekaran, S., et al.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proc. of the 2003 Conf. on Innovative Data Systems Research, pp. 269–280 (2003)
15. EsperTech Inc., Event Stream and Complex Event Processing platform (ESPER) (2012), `http://www.esper.codehaus.org`
16. JBoss.org, Drools (2012), `http://www.jboss.org/drools`
17. Barros, A., Decker, G., Grosskopf, A.: Complex Events in Business Processes. In: Abramowicz, W. (ed.) BIS 2007. LNCS, vol. 4439, pp. 29–40. Springer, Heidelberg (2007)
18. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business Process Intelligence. Journal of Computers in Industry 53(3), 321–343 (2004)
19. El Akkaoui, Z., Zimányi, E.: Defining ETL Workflows using BPMN and BPEL. In: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP (DOLAP), pp. 41–48 (2009)
20. Dayal, U., Wilkinson, K., Simitsis, A., Castellanos, M.: Business Processes Meet Operational Business Intelligence. IEEE Data Eng. Bull. 32(3), 35–41 (2009)

21. Kunz, S., Fickinger, T., Prescher, J., Spengler, K.: Managing Complex Event Processes with Business Process Modeling Notation. In: Mendling, J., Weidlich, M., Weske, M. (eds.) BPMN 2010. LNBIP, vol. 67, pp. 78–90. Springer, Heidelberg (2010)
22. Decker, G., Grosskopf, A., Barros, A.: A Graphical Notation for Modeling Complex Events in Business Processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference, pp. 27–36 (2007)
23. Friedenstab, J., Janiesch, C., Matzner, M., Müller, O.: Extending BPMN for Business Activity Monitoring. In: Paper Presented at the Hawai'i Int. Conf. on System Sciences, Hawai'i, USA (2012)
24. Rosselli, F.: FuTMaN: The Future of Manufacturing in Europe 2015-2020. The challenge for sustainability. In: Final Report for the European Commission. Directorate-General Joint Research Center (2003)
25. European Commission. Manufuture: A vision for 2020, Office for Official Publications of the European Communities (2004)

Process-Based Design and Integration of Wireless Sensor Network Applications

Stefano Tranquillini[1], Patrik Spieß[2], Florian Daniel[1], Stamatis Karnouskos[2], Fabio Casati[1], Nina Oertel[2], Luca Mottola[3], Felix Jonathan Oppermann[4], Gian Pietro Picco[1], Kay Römer[4], and Thiemo Voigt[3]

[1] University of Trento, Via Sommarive 5, 38123 Povo (TN), Italy
{tranquillini,daniel,casati,picco}@disi.unitn.it
[2] SAP AG, Vincenz-Prießnitz-Straße 1, 76131 Karlsruhe, Germany
{parik.spiess,stamatis.karnouskos,nina.oertel}@sap.com
[3] Swedish Institute of Computer Science, Isafjordsgatan 22, Kista, Sweden
{luca,thiemo}@sics.se
[4] University of Lübeck, Ratzeburger Allee 160 23562 Lübeck, Germany
{oppermann,romer}@iti.uni-luebeck.de

Abstract. Wireless Sensor and Actuator Networks (WSNs) are distributed sensor and actuator networks that monitor and control real-world phenomena, enabling the integration of the physical with the virtual world. They are used in domains like building automation, control systems, remote healthcare, etc., which are all highly process-driven. Today, tools and insights of Business Process Modeling (BPM) are not used to model WSN logic, as BPM focuses mostly on the coordination of people and IT systems and neglects the integration of embedded IT. WSN development still requires significant special-purpose, low-level, and manual coding of process logic. By exploiting similarities between WSN applications and business processes, this work aims to create a holistic system enabling the modeling and execution of executable processes that integrate, coordinate, and control WSNs. Concretely, we present a WSN-specific extension for Business Process Modeling Notation (BPMN) and a compiler that transforms the extended BPMN models into WSN-specific code to distribute process execution over both a WSN and a standard business process engine. The developed tool-chain allows modeling of an independent control loop for the WSN.

1 Introduction

Today there is still lack of high-level, model-driven programming tools for Wireless Sensor and Actuator Network (WSN) applications and the integration with enterprise services requires significant effort and expertise in embedded programming of WSNs. Organizations are reluctant to install large-scale WSNs, as this still requires significant, costly, low-level programming of sensing and actuation logic for the WSN, in addition to the physical deployment of the WSN nodes (e.g., inside a building). Additionally, setting up the communication channel between a WSN and an enterprise's information system requires an even larger set of technologies and manually writing of custom code. Domain experts typically lack the necessary low-level programming skills.

To foster widespread adoption and more efficient use of sensor networks for enterprise information systems, a need for a specifically tailored integration technique that is able to bring together sensor networks and business applications [1] is perceived. The aim is to drastically improve the ease of programming of WSNs by enabling the graphical *modeling* of WSN applications, leaving low-level details to a model compiler and a run-time system. WSN programming should be accessible to domain experts, such as business process modelers. They should further be empowered to design the WSN's interaction with enterprise information systems using the methods of business process modeling they are familiar with. Our approach aims to:

– Provide a *conceptual model* that abstracts typical WSN programming knowledge into reusable tasks that can be integrated into modeling notations, such as the Business Process Modeling Notation (BPMN).
– Develop an *extension of BPMN* [2], BPMN4WSN, that enables the graphical modeling of WSN applications and their integration with BPs based on an abstraction layer that hides low-level details of the sensor network.
– Introduce *tools* that enable the design, deployment, and execution of integrated WSN/BP applications. We do not reuse existing APIs toward the WSN; we *program* the WSN and automatically generate the necessary APIs.
– Evaluate our approach with a realistic *prototype* deployment, including a self-optimizing run-time system layer, and a report on the first experiences with its usage in the context of the EU project make*Sense*.

In order to create applications that span both a BPMN process and a WSN application, knowledge in both fields is required. We do not expect the *application developer* (the domain expert) to model an executable process. Rather, we suggest a two-phase approach, where a descriptive process model is created by the developer, which is then refined by a more technical *system developer* using the WSN extension integrated in the process diagram.

In the following, we outline an application scenario to better describe our approach. Then, in Section 3, the typical characteristics and components of WSNs are analyzed. In Section 4 it is outlined how the challenges identified in the scenario are approached conceptually, and in Section 5 the according extension of BPMN is described. Subsequently in Section 6 the implementation of the prototype, including the code generation logic for WSNs is described. Section 7 critically discusses the results achieved so far. Section 8 reviews related work before concluding the paper.

2 Scenario: Convention Center HVAC Management

Our application scenario showcases the operation of a convention center (see Figure 1) that has a variety of meeting rooms, which can be booked for

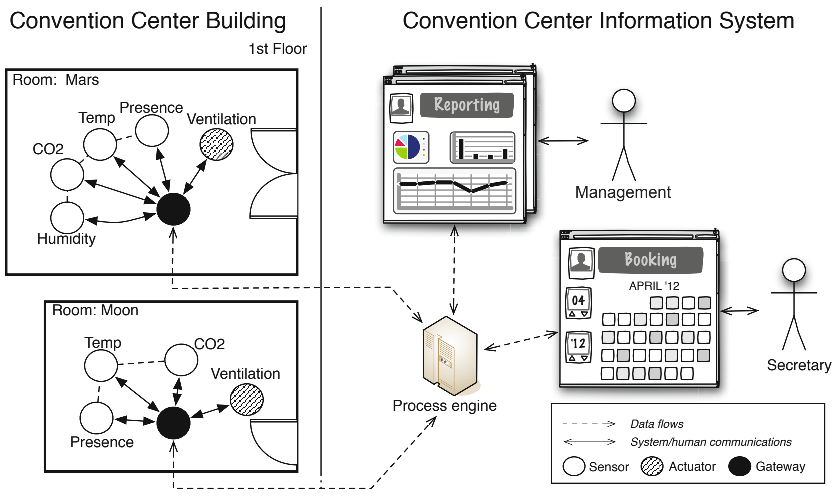


Fig. 1. Integration of a convention center's BP engine with a WSN for HVAC

various events. Each room can be booked at a rate that partly depends on room characteristics (e.g., its size) and partly on the energy consumption of the event organized in the room. For this purpose, the convention center is equipped with a Heating, Ventilation, and Air Conditioning (HVAC) system including a WSN, which ensures comfortable levels of temperature, humidity, and CO_2 for each individual room for the booked duration of the respective event. In order to do so, the HVAC system must be instructed automatically by the convention center's information system about when to activate the ventilation and how long to control the room's temperature and CO_2 concentration for each room. Room conditions are only maintained during the booked times to save energy and only if presence of people is detected by presence sensing; air conditioning is shut off when a meeting is not attended at all or ends prematurely. In turn, the HVAC system feeds back sensor data to the information system, which allow the information system to precisely compute the HVAC cost for each individual event. The information system is used for the booking of rooms, the reporting on energy consumption, and the billing of customers. This mode of operation is more energy efficient that today's common practice, where one would simply run the HVAC system at a fixed rate, independently of room occupation or environmental conditions — a practice that wastes much energy.

Technically, it is necessary to develop (i) the BP logic running inside the BP engine, (ii) the code running on the nodes inside the WSN, and (iii) a suitable set of communication endpoints supporting the interaction of the BPM with the WSN and vice versa. Note that it is *not* the goal of this work to optimize convention center operation or more generally building automation, but to provide a basic set of abstractions, tools, and methodologies that can be used in all scenarios where also WSNs are used. We use it merely as a device to depict a concrete application of our approach.

3 Relevant Properties of Wireless Sensor Networks

Before going into the details of the approach, the special properties of WSNs that are relevant at the application layer and that therefore underpin our model of the system are explained. A WSN is a distributed system, namely a network of wireless, battery-powered, autonomous, small-scale devices, so called nodes, each of which is equipped with one or more sensors or actuators or both. Nodes are battery-powered and replacing the battery is mostly not intended or not feasible from a Total Cost of Ownership (TCO) perspective. Therefore, they make use of ultra-low-power hardware, that is drastically limited in processing power, memory, and transmission bandwidth and the application software running on the nodes, including wireless communication protocols, needs to be optimized for low power consumption to extend network lifespan. These limits typically prevent executing a regular BPMN engine on the devices that interprets BPMN models serialized as XML.

Sensors are used to sense information from the real world (e.g., temperature) while actuators perform actions that change the state of the environment (e.g., control a motor or a lamp). The typical number of nodes inside a network can vary from a few to hundreds or even thousands. Via radio links, a node can generally communicate with all other nodes in its transmission range and with nodes further away by multi-hop, routed communication. WSNs are able to self-organize, overcome network failure, and execute distributed computation logic, such as computing the average of sensor values while those are routed to a destination node. Often, WSNs are composed of heterogeneous nodes, each equipped with a custom set of sensors, so that, for example, one type of node can sense CO_2 and humidity while another type of node is able to control an automatic door, while a third has enough special hardware to compute complex arithmetics.

As a basis for modeling WSN application logic, a very simple model of the physical set-up that is sensed and acted upon is assumed: a given WSN monitors real world entities, each is referred to as an Entity of Interest (EoI) which can be a *location* or a *thing.* A thing is any physical object, while a location is a space that the sensor network is monitoring, e.g., a room or a building. A domain expert is usually only interested in the EoIs and the operations that can be applied to them, but not in the technical layer of sensors that sense or the actuators that influence them.

To overcome the limitations of WSN hardware and to maximize efficiency of operations, the research community has introduced a large number of programming abstractions to program wireless sensor networks [3]. By abstracting existing programming concept into high-level constructs [4] (described in Section 5.2) and assuming that all existing functionality can be expressed using them, one can use high-level constructs as basic building blocks for graphical modeling. Usually, a sensor network will perform some or all of these tasks:

Sensing: measuring one or more environmental parameters of an EoI, such as temperature or humidity, making use of the sensing equipment of the nodes.

Actuation: enacting operations physically affecting an EoI, e.g., controlling or moving it or flashing a LED. WSNs are often used to actuate or control the environment in reaction to sensed parameters, creating a control loop (as the actuation eventually triggers changes in the sensed values).

Task distribution: distributing operations that coordinate a subset of nodes, e.g., any in-network aggregation on the input values or the election of a controller node based on certain criteria. As WSNs consist of several nodes, several of which can monitor the same EoI, especially data aggregation operations are often required, e.g., to compute the average temperature of a room observed by many sensor nodes.

From the perspective of a domain expert, it is irrelevant which part of a WSN performs a task, e.g., whether an operation is carried out by a single node or the network as a whole as long as the operations are addressable by an EoI.

4 Requirements and Approach

In the convention center scenario, there is a need for collaboration between the reservations and billing systems in the back-end and the sensor network that executes the sensing and actuating operations. Thus, the application runs on different types of systems which can be seen as two distinct participants in the process. This raises the need to model both the intra-WSN logic and its interactions with back-end systems as a collaboration of two process participants. While the back-end part is orchestrated using classical Business Process Management (BPM), modeling the process logic to be executed inside the WSN needs certain provisions (e.g., model extensions) to enable the specification of WSN logic in a high-level fashion and the creation of code that can be executed in the network.

Typically, the integration of WSNs into BPs is based on the invocation of services exposed by the network [5,6,7]. This results in a modeling approach that uses the network as set of available operations on which a process can be constructed, but that prohibits the programming of the WSN itself. This limits the possibility to define custom WSN logic to be carried out by the network as part of the process. Instead, the key idea of our approach is to develop a business process modeling notation that allows a domain expert to program both the BP *and* the actual network logic, without the need to know and specify all the low-level details. The created process model is later used to derive the code that will be executed by the WSN. In this way, the WSN logic is fully specified at the process level.

The specific requirements we identify can be divided into supporting *modeling*, *deployment* and *runtime*. Supporting modeling means defining a modeling paradigm that fits the needs of a domain expert and integrates back-end business processes and WSN logic using a single modeling language. This requires to:

– Provide an easy to understand and familiar way of expressing WSN logic; enable integrating WSN processes into back-end processes, coupling them and allowing for easy data sharing.

- Define a set of concepts to describe the logic and operations that can be combined for creating reusable, high-level WSN modeling constructs. We have to supply the modeler with the possibility to specify operations like sense, actuate and aggregate for measurements over EoIs.
- Model WSN capabilities and details. WSNs are usually heterogeneous regarding the type of sensors and actuators. Knowing the characteristics of the network is fundamental to have an overview of which things and locations can be controlled and monitored by the WSN as well as which operations the WSN is able to perform. Having such a model will give the domain expert the ability to express the desired processes in the familiar terms of EoIs and irrespective of technical systems.
- Supporting the modeler in designing only feasible processes by restricting the available modeling constructs to him to what the WSN is capable of executing.

Supporting the deployment of the process requires to:

- Split the process model into an intra-WSN part and a WSN-aware part (back-end). The process is divided between two actors that participate in the execution. These two parts of the process have to be separated and handled differently.
- Create WSN binary code. The intra-WSN part of the process has to be translated to binary code and injected into the nodes. This code is generated based on the flow of the process model and tasks that describe the operations.
- Create the endpoints and communication channels to handle the messages from and to the network. After having split the process in two parts and after having translated the WSN part into binary code the communication between these two participants has to be guaranteed. To do so, the endpoints and the communication channels through which the messages will be sent/received need to be available.

Supporting the execution requires to:

- Provide a process engine to execute the WSN-aware business process part. The process engine also handles the communication with the WSN.
- Run the code in the WSN. Part of the process actually runs inside the network without the need for external communication and control. The process is executed on the gateways and the actions are distributed on the nodes, guaranteeing the correctness of the process depicted by the modeler.

Figure 2 illustrates the conceptual model of how we approach WSN programming. The model is not meant to be an extension of the BPMN meta-model. Only part of it is related to BPMN4WSN, the other part is related to our own modeling formalism for the definition of low-level WSN logic. The two entities on the top represent the physical WSN, which we abstract as composed of a set of *Nodes* (sensor or actuator nodes) supporting a set of native operations, the so-called *WSN Operations*, such as *sense* CO_2 for a sensor or *open* for a valve

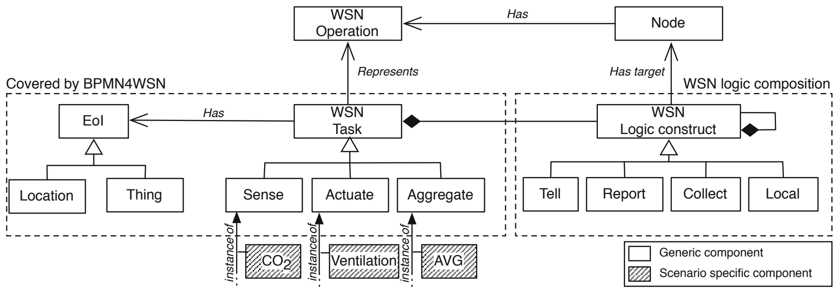


Fig. 2. Conceptual model of WSN operations

actuator. We allow the domain expert to use WSN operations by abstracting away from the network topology, i.e., nodes, and instead allowing him to reason in terms of EoIs via a dedicated task type, the *WSN Tasks*. A *WSN Task* is a generic action that can be used to express *sense*, *actuate*, and *aggregate* operations and that can be executed by the network. The *WSN Task* is logically connected to an EoI, which allows the modelers to scope the action. That is, the EoI specifies *where* the action will be executed; it could be a *thing* or a *location*. WSN Task and EoI represent the high-level constructs used to model WSN logic in BPMN4WSN. This level of abstraction is however not enough to describe all the needed details to generate binary code that runs on the nodes, which instead requires taking into account the topology of the network. The detailed specification is based on *WSN logic constructs*, which abstract operations that can be configured (e.g., by adding a concrete target node resolving a logical EoI) and translated into binary code. The composition of WSN logic constructs (the *WSN logic composition* box) allows the system developer to refine the process model designed by the domain expert and to fill WSN tasks with concrete logic.

Figure 3 shows the architecture of the tool chain for developing WSN/BP applications containing an extended BPMN editor in which the process is modeled, and a compiler for translating the high-level specifications into low-level executable binary code for the sensor network and for the process engine. Next, the *modeling* and *deployment* part are discussed in more detail; a first prototype of the tool is discussed afterwards.

5 BPMN4WSN

As illustrated in Figure 3, two types of developers jointly develop a process model: the application developer and the system developer. The application developer is the person who models the coarse process; he is an expert of the domain with experience in business process modeling and has some WSN background. The system developer is a WSN expert and has the task of creating the refined, XML-formated model of the system (see the bottom left corner of Figure 3). This model contains information of the network such as the EOIs, nodes and available sense and actuate operations. The two roles collaborate mainly in the design of WSN Tasks. The application developer creates a process that crosses the system boundary between

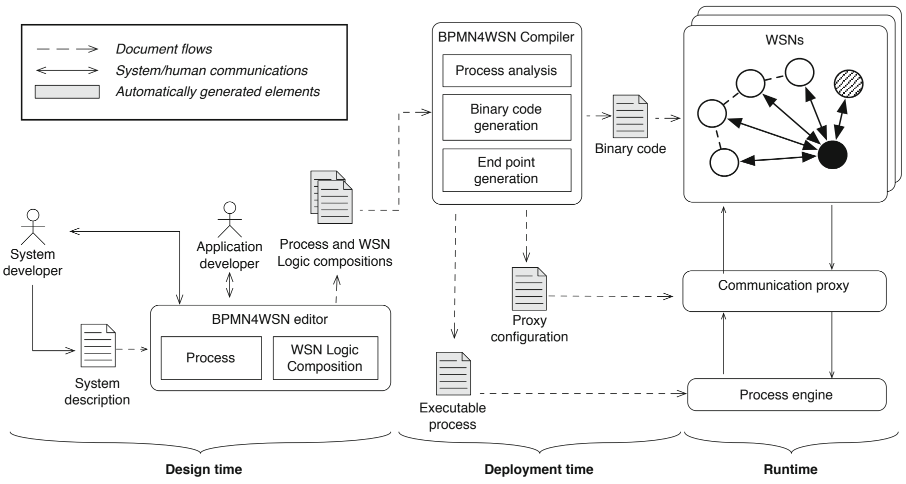


Fig. 3. Architecture

standard IT and WSN including the specification of the behavior of the latter. He defines a descriptive, not yet executable version of the process. For instance, in the convention center use case, the application developer would specify a task for reading the latest sensor values or driving an actuator based on the system descriptor model. Later, the system developer would refine this model by adding WSN logic components to make the tasks that involve the WSN executable.

5.1 Process Logic

In our solution the design of the business process is mainly carried out by the application developer, who uses BPMN [8] with some additions based on the extension points defined in the standard (without touching the BPMN meta model), designed to model the salient characteristics of the WSN. The extended language is referred to as *BPMN4WSN*. This extended version comprises both new components and modeling rules.

A BPMN4WSN process must be composed of at least two pools: an *intra-WSN* pool and *WSN-aware* pool; Figure 4 contains a minimal example. The intra-WSN pool is the part where the WSN logic is specified, while the WSN-aware pool is a classical BPMN process. The splitting into process logic executed inside and outside the WSN forces the modeler to explicitly model interactions between the two parts as messages, directly mapping the run-time behavior (where messages are the only way of interaction between the parts) to the model. This separation also enables the clean generation of code.

In the intra-WSN pool, constructs that directly orchestrate WSN functionality (made available through high-level abstractions) are needed. This need is addressed by introducing a new activity type: the *WSN Task*, that can only be used in the intra-WSN part. It has two properties: a reference to a set of *WSN logic construct* definitions and the *EoI* to which the respective operations should

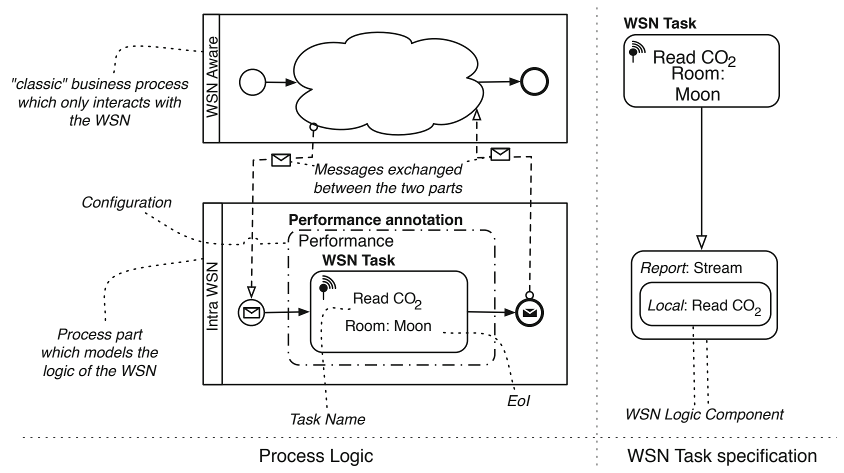


Fig. 4. WSN-specific modeling constructs in BPMN4WSN

be applied (see Figures 4 and 2). It has an antenna on the top-left corner to distinguish it from other tasks; if specified, the *EoI* value is written below the task name. For example, setting the EoI value to "room Moon" will execute the task on those nodes that belong to the "room Moon". In a nutshell it specifies *where* (i.e., by which subset of nodes) each WSN Task is executed.

The referenced *WSN logic construct* definition is the set of operations that have to be performed by the network. For simplifying the modeling of such low-level programming specification, a set of *WSN logic constructs* that describe the common operations and the way they can be combined is created.

In addition to the WSN Task, a *Performance Annotation* element, i.e., an extension of the BPMN *group* element which shows the chosen performance configuration on the top-left corner, is introduced. It is used for describing the network behavior from a performance point of view. This new component allows the application developer and system developer to decide when the network performance goal has to be changed (e.g., to optimize battery lifetime). For example, when a room is empty, the network will be set to *low energy consumption* mode in order to save battery and prolong node network lifetime at the cost of lower reactiveness and possibly less reliable message transfer. In cases where high performance is needed (at the cost of battery power), other performance annotations are used. At run-time the execution semantics of these annotations is that one performance mode is set for the whole WSN, depending on the number of the tasks in each performance group. The group that contains the most tasks to be executed sets the performance mode.

5.2 WSN Task Specification

WSN Tasks are modeled in two steps: (i) the *process design* and (ii) the *process refinement*. The process design is generally carried out by the application developer. He just specifies a *WSN Tasks* with a speaking name, which can be a *sense*, *actuate*, or *aggregate* operation and the *EoI* on which the operation has to be executed. This part of the modeling is represented in Figure 2 by the items inside the *BPMN4WSN* dashed rectangle.

The process refinement (an example is shown in Figure 4), instead, is generally performed by the system developer. Its goal is to transform all high-level WSN Tasks into executable operations by combining *WSN logic constructs* which model the network behaviors. As shown in Figure 2, each WSN Task represents *WSN logic constructs* that are the basic functionality and instances of so called meta abstractions [4] that must be configured and instantiated:

Local actions are executed locally on each sensor node.

- The *tell/report actions* represent one-to-many/many-to-one communication.
- The *tell action* enables a node to delegate an embedded action to a set of other nodes.
- The *report action* enables the gathering of information from many nodes.
- *Collective actions* enable distributed, many-to-many collaborations.

Each of these distributed actions has a *target*, which is used to select the subset of nodes the action refers to (obtained by resolving logical EoIs into physical nodes, based on the system description). In addition there is also the possibility to specify *data operators* useful to perform mathematical operations during transmission of data (e.g., to compute the average).

Each specific WSN deployment has its unique system-description, which is the starting point for modeling. It describes the details of the network and it is used as configuration for the model editor. The document provides a high-level description of application-specific details of the concrete WSN deployment to the business process editor and to the model compiler. It is used by the editor to list only those attributes to the system developer that are actually available in a concrete deployment, such as the list of EoIs (simple or composed ones like "First Floor" comprising "room1" and "room2") and to restrict the selectable operations (e.g., CO_2 sensing can only be selected if EoI "room2" has been selected, because only that room is equipped with CO_2 sensors).

6 Prototype

The approach described in the previous sections has been implemented as a proof-of-concept prototype. Figure 3 depicts the architecture of the prototype, showing the document flow and the actors involved. The modeling process, defined by our tool chain, is divided into three phases: *modeling*, *translation*, and *execution*.

Modeling. For the modeling part of the prototype, a well-known web-based BPMN editor called *Signavio Core Components* (`http://code.google.com/p/signavio-core-components`) has been extended. The editor has been modified by adding a start page for scenario selection and a model editor for the *WSN logic constructs*.

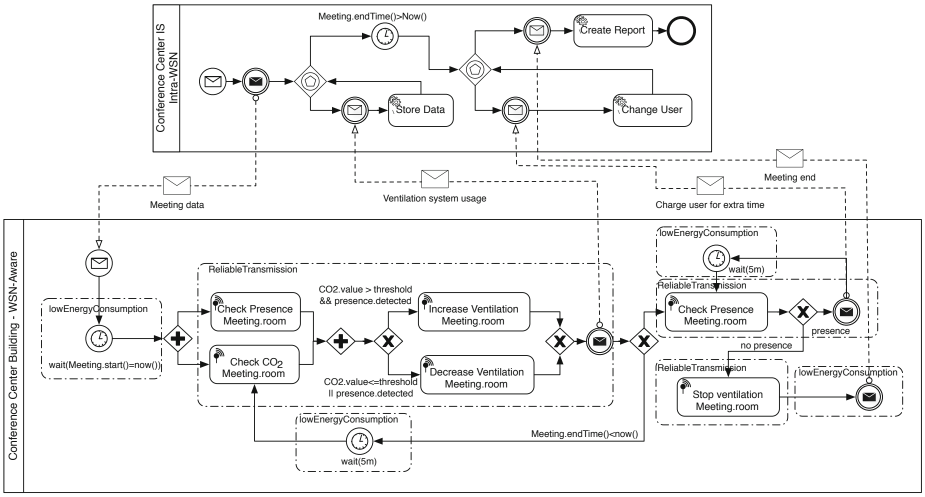

Conference Center IS
Intra-WSN
Meeting.endTime()>Now()
Create Report
Store Data
Change User
Meeting data
Ventilation system usage
Meeting end
Charge user for extra time
Conference Center Building - WSN-Aware
lowEnergyConsumption
wait(Meeting.start()=now())
ReliableTransmission
Check Presence Meeting.room
Check CO2 Meeting.room
CO2.value > threshold && presence.detected
CO2.value<=threshold || presence.detected
Increase Ventilation Meeting.room
Decrease Ventilation Meeting.room
wait(5m)
Meeting.endTime()<now()
presence
no presence
Stop ventilation Meeting.room

Fig. 5. The HVAC process of the convention center. For high-res screen shots, visit `http://goo.gl/4Roc2`; last check 23 March 2012.

The start page is used to select or create a separate workspace for each scenario to enable development for distinct WSN set-ups, each with its on system-description. In each workspace, only operations that can actually be executed inside the corresponding network are enabled, helping the modeler in creating correct executable processes. For instance, in our example scenario there would be the possibility to sense CO_2 and presence but no other environmental parameters as the WSN is only equipped with these sensors.

BPMN extension points have been used to realize WSN Tasks and performance annotations as explained in Section 5 and in Figure 4. To support the modeling, the modeling tool has been extended with these two components.

The *WSN logic construct* composition has been enabled by creating a new meta model inside the tool. By doing so, the modeler is given the possibility to compose *WSN logic construct* blocks by dragging and dropping and nesting them according to predefined composition rules that are checked by the tool. The composition is later translated into an internal format, and the files are used by the compiler to create the binary code for sensors.

Example. In Figure 5 there is a screen shot of the process that models the scenario explained in Section 2. For the sake of clarity, in the intra-WSN process only CO_2 measurement and presence detection are modeled.

A new process instance is started when a new meeting is scheduled. The WSN will be set to *low energy consumption mode* until the actual meeting starts. Throughout the duration of the meeting, the network checks the room conditions, increasing the ventilation when sensor values exceed a given threshold and a human presence is detected. After the scheduled meeting end time, the network checks if someone is still in the room, in which case the information system is informed, charging the user for *extra time*.

Translation and Execution. The WSN-aware part of the process is a standard BPMN model that can be executed by a process engine exterior to the WSN. The intra-WSN part, instead, is translated into executable code. A tool for translation called *model compiler* takes this part of the process and generates code implementing a custom execution engine. The executable program hence behaves similar to a regular BPMN engine interpreting the given BPMN model. The generated program implements a finite state machine, realizing the execution semantics of the translated process model including instance management and message correlation, and of course keeps track of all execution tokens in each process instance as specified in the BPMN 2.0 specification.

For example, an exclusive diverging gateway will be translated into a series of if statements (mapping the conditions on the outgoing flows) in the "main loop" of the program. Each WSN Task is translated using the *WSN logic construct* composition describing sensor logic. This is the most extensive generation step, as these sub-models need to be mapped to an API for instantiating, managing, and using those programming abstractions. The system-description describes the characteristic of each node of the network and it is used as input for the translator. The EoI of a WSN Task is mapped to attribute matching at run-time, e.g. if a WSN Task has been configured to operate on EoI "Floor 1" and the system developer contains information which room ids belong to that floor, this could be mapped to the expression `location='room1.1' or location='room1.2'`.

The two parts of the process can now be executed separately. To make them communicate, the model compiler maps the message flows between intra-WSN and WSN-aware process to communication endpoints that are created automatically on either side, enabling each part to receive and send messages. As the message format and transmission encoding are out of scope of the BPMN specification, a simple message format and an efficient transmission encoding are defined and implemented in both the generated intra-WSN executable and as an extension to a regular BPMN execution engine. In order to support the coordination of multiple instances, each message contains a field that is used for instance correlation and the execution of message start events creates instance IDs that need to be used by either side of a same process instance.

7 Discussion and Future Work

Our approach was guided by the core requirement presented in Section 4, i.e., to integrate WSN programming into business process modeling. We address this requirement by offering unified modeling in one model editor, hiding model artifacts that are not relevant in a given modeling context, splitting work between application developer and system developer, and providing model compilation and execution as a custom engine in the WSN.

The work described in this paper is a first iteration towards integrating WSNs with BPs, combining classical business process modeling with ad-hoc extensions for WSNs that hide low-level network details. This integration allows an application developer to design process logic both inside and outside the sensor network, without requiring intimate knowledge of how to program distributed computations inside a WSN; an intuitive understanding of EoIs and sensing and actuating actions is enough. The system developer instead only focuses on the refinement of WSN Tasks. The described tool-chain takes care of splitting the two logics (intra-WSN and WSN-aware) and of the binary code generation. Endpoints for communication between the business process and the network are created following the model of the process. Yet, there is still space for improvement, space that we are going to cover in our future work.

Patterns. For instance, WSNs are characterized by hardware constraints that an application developer is typically not familiar with. Thus, providing patterns as best practices for modeling WSN logics is a necessity that is not yet implemented but that will be covered in the future, in order to further simplify the creation of WSN applications.

Two Different Meta Models. Our current solution uses a different meta-model for modeling the lowest level of WSN logic, the *WSN logic construct.* WSN Tasks internally follow a logic that is different from BPMN, in order to compose WSN logic constructs. We could therefore not model them via simple sub-processes. A new logic and meta-model is needed, which is however seamlessly integrated in the same model editor. Opting for pure BPMN modeling, also for the *WSN logic construct* would have complicated the modeling as they have strict composition rules that could not be expressed easily in BPMN. Instead, we use a simple box model to compose *WSN logic constructs,* supporting the construction of correct models. We will evaluate to what degree we can use more BPMN and less custom modeling constructs in the future.

Modeling Two Separate Pools. Our current prototype forces the modeler to model at least two pools, one for the intra-WSN part and one for the WSN-aware part. We opted for this solution as it makes communication from and to the WSN explicit and the compilation and creation of communication endpoints easier. Modeling all logic in a single pool would be less burdensome to the modeler, but it requires non-trivial data and control analyses of the process, which for the sake of simplicity we did not implement yet.

Events. Although the current model does not explicitly use special BPMN events for the WSN inside the process model, the approach is strongly based on asynchronous communication under the hood. On the one hand, modeling two explicit pools with asynchronous message exchanges has similar semantics as BPMN message events. On the other hand, for example, the current implementation of receiving a sensor value is realized asynchronously: rather than querying the sensor directly, the buffered last value of a stream of sensor values is read. This behavior is however hidden behind the WSN logic constructs. Introducing WSN-specific events in the process model would however allow for more flexible control flow logic and is therefore part of the future development of the system.

Multiple Processes Interacting with One WSN. In our current prototype, the generated intra-WSN logic only supports conversations with multiple instances of one process model. In the future, we intend to support multiple process models by merging the intra-WSN parts of all models to a combined model or by running several of the generated engines concurrently at operating system level and dispatching messages based on model identifiers. In practice, we could single out the WSN-internal logic as an own pool and allow the application developer to define multiple (WSN-aware) processes interacting with this WSN-internal process. This would enable the generation of WSN code that natively supports multiple different WSN-aware processes.

8 Related Work

Building commercially relevant applications on resource-constrained, networked embedded systems (the front-end) such as WSNs while integrating them into business processes of an enterprise (the back-end) is a complex, challenging task that has to be repeated for each combination of front-end and back-end. Numerous efforts have been made, aiming also at demonstrating the business benefit.

Approaching the problem bottom-up, i.e., from the WSNs, several solutions have been proposed to simplify programming. Although many programming abstractions have been introduced, most of them aim at simplifying the activities of skilled WSN programmers [9] and cannot be used directly to specify high-level process constructs by domain experts without WSN expertise.

The COBIS project (`www.cobis-online.de`) aimed at integrating heterogeneous WSNs with back-end systems by providing a web service facade to the WSN's functionality. The proof of concept was trialled in an environment, health, and safety application scenario, more specifically by enforcing physical storage rules for hazardous goods managed in an enterprise system [5,10].

The SOCRADES project (`www.socrades.eu`) targeted industrial automation with the goal to almost eliminate the need for any proprietary intermediate layers between embedded services and the business back-end by directly service-enabling devices themselves [1]. The approach was based on the WS-* family of web service standards and only for very resource-constrained and legacy devices a gateway/service-mediator concept was developed to enable those to participate in service orchestrations.

Other proposed solutions for modeling sensor network applications using a process-based design include the Graphic Workflow Execution Language for Sensor Network (GWELS) [6], which enables the design of data-flow as workflow, and an ad-hoc architecture for handling the communication. Similarly, [7] uses a process paradigm for defining WSN applications, easing the configuration for non-experts of the field. Mash-up composition is also promising; in [11], the authors wrap smart-objects with web services, introducing an architecture and a web-based mash-up tool for composition and execution. These solutions enable the modeling of WSN logic in a model-driven fashion but without deriving the executable logic of the network.

Recently, BPMN has gained interest as method to *program* WSNs. Caracas et al. [12,13] presented studies on the expressiveness of the language and its potential to be compiled into source code for WSN nodes. As results they produce a system that creates WSN applications by compiling BPMN processes. The outcomes highlight that, as it is, BPMN is powerful enough for specifying the high-level behavior (if modeled with correct patterns) more than low-level one. At the same time they prove how a process can be compiled into native source code for WSNs, without losing too much performance compared to hand-written code. These preliminary works show the possibility to compile the BPMN for creating binary code. However, the example shown in this work users a higher-level API, that does not allow one to fine-tune communication in the WSN as it is possible with our approach.

In the past months, extensions of BPMN for modeling smart objects have been proposed as outcome of the IoT-A (`www.iot-a.eu`) project [14,15], an idea that shares some common ground with our approach. The idea is to extend the BPMN language to model Internet of Things (IoT) aspects. However, this approach differs as they propose modeling extensions that affect the language at a high level of abstraction; in fact their goal is to use this language to model IoT services instead of creating the logic from the process.

Approaches like SysML [16] are only remotely related to our approach. This modeling framework, derived from UML, allows the modeling of low-level details of a WSN system. Yet, SysML models are graphical models without a standard serialization, therefore they are not directly usable for process-based integration.

9 Conclusion

In the era of the IoT, collaboration and integration of non-conventional IT devices, such as entertainment and automotive equipment, RFID devices and tags, or WSNs, with Enterprise services is of paramount importance [1]. In this paper, we focused on one relevant representative of this need, i.e., WSNs, which typically still represent isolated and impenetrable realities from a business IT point of view. We proposed a layered approach for developing, deploying and managing WSN applications that natively interact with enterprise information systems, such as a business process engine and the processes running therein. We did not try to crack the whole problem at once, e.g., by aiming at a business-view-only approach to WSN application development, and rather foster current practice, equipping both the application developer (holding the process knowledge) and the system developer (holding the WSN knowledge) with effective languages and instruments to co-develop advanced, process-based WSN applications with non-trivial distributed sensing and actuation logics.

Acknowledgments. This work was supported by the European Commission funded project make*Sense* (`www.project-makesense.eu`).

References

1. Karnouskos, S., Savio, D., Spiess, P., Guinard, D., Trifa, V., Baecker, O.: Real world service interaction with enterprise systems in dynamic manufacturing environments. In: Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management. Springer (2010)
2. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011), http://www.omg.org/spec/BPMN/2.0
3. Mottola, L., Picco, G.: Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Computing Surveys (CSUR) 43(3), 19 (2011)
4. Casati, F., Daniel, F., Dantchev, G., Eriksson, J., Finne, N., Karnouskos, S., Montero, P.M., Mottola, L., Oppermann, F., Picco, G., Quartulli, A., Römer, K., Spiess, P., Tranquillini, S., Voigt, T.: Towards business processes orchestrating the physical enterprise with wireless sensor networks. In: ICSE 2012 (June 2012)
5. Spiess, P., Vogt, H., Jutting, H.: Integrating sensor networks with business processes. In: Real-World Sensor Networks Workshop at ACM MobiSys (2006)
6. Glombitza, N., Lipphardt, M., Werner, C., Fischer, S.: Using graphical process modeling for realizing SOA programming paradigms in sensor networks. In: WONS 2009, pp. 61–70 (2009)
7. Amundson, I., Kushwaha, M., Koutsoukos, X., Neema, S., Sztipanovits, J.: Efficient integration of web services in ambient-aware sensor network applications. In: BaseNets 2006 (October 2006)
8. White, S.: Introduction to BPMN. IBM Cooperation (2004)
9. Mottola, L., Picco, G.P.: Programming wireless sensor networks: Fundamental concepts and state of the art. Technical report (2008)
10. Spiess, P., Karnouskos, S.: Maximizing the business value of networked embedded systems through process-level integration into enterprise software. In: ICPCA 2007, pp. 536–541 (July 2007)
11. Guinard, D., Trifa, V., Wilde, E.: Architecting a mashable open world wide web of things. Technical Report 663, Institute for Pervasive Computing, ETH Zurich (2010)
12. Caracaş, A., Kramp, T.: On the Expressiveness of BPMN for Modeling Wireless Sensor Networks Applications. In: Dijkman, R., Hofstetter, J., Koehler, J. (eds.) BPMN 2011. LNBIP, vol. 95, pp. 16–30. Springer, Heidelberg (2011)
13. Caracas, A., Bernauer, A.: Compiling business process models for sensor networks. In: DCOSS, pp. 1–8. IEEE (2011)
14. Sperner, K., Meyer, S., Magerkurth, C.: Introducing Entity-Based Concepts to Business Process Modeling. In: Dijkman, R., Hofstetter, J., Koehler, J. (eds.) BPMN 2011. LNBIP, vol. 95, pp. 166–171. Springer, Heidelberg (2011)
15. Meyer, S., Sperner, K., Magerkurth, C., Pasquier, J.: Towards modeling real-world aware business processes. In: Proceedings of the Second International Workshop on Web of Things, WoT 2011, pp. 8:1–8:6. ACM, New York (2011)
16. Weilkiens, T.: Systems engineering with SysML/UML: modeling, analysis, design. Morgan Kaufmann (2007)

Modeling Rewards and Incentive Mechanisms for Social BPM

Ognjen Scekic, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology, Vienna, Austria
{oscekic,truong,dustdar}@infosys.tuwien.ac.at

Abstract. Social computing is actively shaping Internet-based business models. Scalability and effectiveness of collective intelligence are becoming increasingly attractive to investors. However, to fully exploit this potential we still have to develop crowd-management frameworks capable of supporting rich collaboration models, smart task division and virtual careers. An important step in this direction is the development of models of rewarding/incentivizing processes. In this paper, we conceptualize and represent rewarding and incentive mechanisms for social business processes. Our techniques enable definition, composition, execution and monitoring of rewarding mechanisms in a generic way.

Keywords: rewards, incentives, socially-enhanced BPM.

1 Introduction

Incentives and rewarding are inseparable parts of business processes today. Their main purpose is to align the interests of workers and employers. By stimulating workers with various monetary, material and psychological rewards the employer can enhance productivity, quality, knowledge, collaboration, leadership, and other positive qualities in the company. Even more beneficial are the selective effects of the incentives [4]. Each particular incentive usually targets to enhance a single aspect of worker's performance. This can lead to workers starting to exhibit various dysfunctional types of behavior, meant to increase productivity only in segments targeted by the incentive while neglecting the others. This is why in practice often a number of simple incentives are combined together targeting each other's unwanted consequences.

Motivation: Social Computing for business is expected to grow substantially in the coming years[1]. Crowdsourcing is already a well-established business model, but it is characterized by exploitation of unstructured crowds of independent workers performing small, simple tasks. Collaboration models on the web are becoming richer, evolving from traditional company to crowdsourcing to social business. In the future, we expect that the development and adoption of novel business models involving social business processes will match or extend the contemporary processes in traditional companies. These business processes (so-called *Social BPs)* will (partially) rely on dynamic, distributed workforces, structured in problem-related, ad-hoc assembled teams of professionals. The new business reality will require advanced organizational and management structures for the workers, intelligent task division and distribution, with the advent of long-lasting “virtual careers”.

These trends inevitably require advanced crowd-management capabilities in future social computing platforms, including novel rewarding/incentive frameworks exploiting the advantages of vast amounts of digital productivity records, cheap peer evaluation, psychological techniques, etc. However, to the best of our knowledge, existing social computing platforms lack techniques for formulating, composing and automatically deploying incentive mechanisms.

Contribution: We identify the basic composing parts and conceptualize a model for representing most real-world incentive mechanisms using rewarding rules and events. Our model supports reasoning and acting along quantitative, structural and temporal data associated with teams of workers, allowing composition of incentive mechanisms into complex schemes.

Related Work: Most related work in the area of rewarding and incentives originates from economics, organizational science, psychology and applied research. It can be used to classify and substantiate the basic rewarding approaches and expected outcomes, and to simulate the responses to incentive strategies. The principal economic theory treating incentives today is the *Agency Theory*[3]. We use many of the basic findings from this theory implicitly in the foundation of our model. The paper [6] presents a comprehensive review and comparison of different incentive strategies. In computer science, the topic has been treated only within application-specific contexts so far, e.g., social networks[8], human microtask platforms[5,7], peer-to-peer networks, etc. However, to the best of our knowledge, the topic has not been previously addressed elsewhere in a comprehensive, general manner.

2 Modeling Rewards and Incentive Mechanisms

Incentive is any activity employed by the system to stimulate or discourage certain worker activities before the actual execution of those activities. **Reward** is any kind of recompense for worthy services rendered or retribution for wrongdoing exerted upon workers after the completion of the activity.

Based on a thorough review of classical economic literature on the topic we identified the three components that every *incentive mechanism* consists of:

(1) **Evaluation methods** serve to assess the quality of worker’s performance from different aspects. They provide inputs for making a decision whether to apply a reward/sanction. (2) **Incentive conditions** represent the business logic behind the incentive mechanism. They contain the rules for application of rewarding actions and take the evaluation results as inputs. (3) **Rewarding actions** are concrete measures taken against individuals or teams to influence their future behavior.

Any concrete incentive mechanism can be expressed as *incentive rules* containing these three components, in a system-independent way. It is is then possible

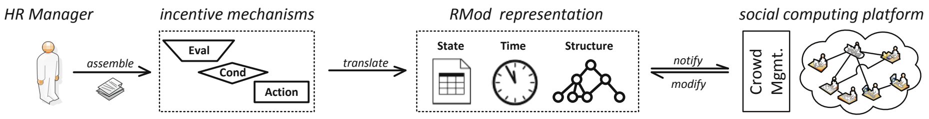


Fig. 1. Supporting incentives in social computing environment

to translate automatically such rules into queries and actions upon a **rewarding model (RMod)**, representing the following aspects of a real-world system:

State represents quantitative state of the system. It includes global attributes and individual worker attributes, representing different performance metrics (QoS). These metrics are part of the business logic of a company, and, as such, represent an input to our model. **Time** is expressed as a collection of time-annotated records of past and future worker interactions, supporting various time conditions and constraints. **Structure** allows representation and manipulation of various types of relationships among workers.

The RMod represents an abstraction layer between an actual real-world platform that manages worker teams and client's system-independent representation of an incentive mechanism (Figure 1). At any time, the RMod must mirror the current state of the external system. RMod must be versatile and general enough to model many different real-world platforms and support application of any incentive mechanism. Therefore, it must stay decoupled of both. This allows for seamless switching between different incentive strategies and application of same strategies on different systems. The aforementioned implies a highly abstract and minimalistic RMod that fits to various underlying systems, and is able to support expressing a range of specific incentive mechanisms by the end-users. We believe that incentive mechanisms should be expressed *declaratively*, i.e. without explicit control flow and data manipulation. As in real life, a client then needs only to specify what incentive actions should be applied and upon which conditions. The condition evaluation and actual scheduling and execution of incentive actions should be encoded *imperatively* at runtime in RMod, transparent to the client. The benefit for the client is the ability to specify human-friendly, portable, scalable, composable and modifiable incentive strategies. In the remainder of this paper we focus only on RMod.

2.1 The Rewarding Model (RMod)

To develop a general-purpose model we adopt simple and abstract representations. An organization, referred to as *principal*, employs a group of *workers* to perform a complex process, consisting of multiple tasks. The principal uses a system that splits, assigns, and in every other aspect manages task lifecycles. A worker is assigned a (sub)task to perform in a given time and agrees to be subject of incentive evaluations. Workers can work individually on assigned tasks, or have a formalized organization or relationship with the principal (be employed, be part of teams, have managers, etc.) Workers can be paid and/or otherwise rewarded for their contribution. Principal's knowledge of the task progress is obtained by periodic *messages* (updates) that he receives from workers and subsequent reasoning over that data. Similarly, his influence over a worker (penalty, promotion, bonus, etc.) is performed via legally-binding messages to the worker. With this assumption, a worker can be represented by a real human, e.g., as a member of a Social Compute Unit (SCU) [2] and via a Web Service interface. Without loss of generality, we can assume that the principal employs a group of humans that perform their work via Web Services, by contracting a third-party human-labor platform that fully takes care of task and worker management. That way, we can focus solely on providing the services of management of incentives and rewards (**RI Management**).

Task is the basic working unit. Workers are rewarded for working on a particular task within the task's timeframe, although the outcome of the evaluation can also depend on the history of previous contributions. Therefore, the lifetime of a worker is not related to the duration of the task. The principal maintains his own view of the workers and the relations between them in a *community graph*. The nodes in the graph represent the workers, while the edges represent different real-world relationships among the workers (e.g., records of past collaborations, trust, dependencies, managerial relations, etc.). In addition, each node is described by a set of attributes. The attributes may represent task-specific (short-lived) or permanent records of worker's performance. This is the most general representation possible. However, in practice we expect this model to be coupled with a real-world system, so the nodes and relations can be mapped to entities in a system that uses, e.g., BPEL4People, SCU or a custom platform for managing tasks and workers.

Each task is performed in iterations. *Iteration* length is measured in clock ticks. *Clock tick* is the basic unit of time measurement. Worker's progress is submitted upon iteration expiry so the system can update the QoS metrics. Iteration is the basic unit for splitting, monitoring and evaluating task execution in runtime. Iteration cycle length is tunable to allow better runtime adaptability, as the iteration length can be a significant factor when evaluating results and can affect the performance of the team. In order to represent history of past behavior, as well as scheduling of future performance evaluations and rewarding actions, we include in the model the notions of *timeline* and *event*. The timeline is a time-stamped collection of past and future event records. An **event** object encapsulates an executable action and a timestamp. Events are interpreted by the system as orders or suggestions to the system itself or particular workers, e.g, to notify a worker to increase QoS level in future iterations, to dissolve a team, invite new workers, terminate contracts, etc. Events can be generated by the system itself or originate from an incentive mechanism. They can target individual workers, groups of workers or global system properties, depending on the query that forms part of the action contained in the event object.

An event can be in two states: *scheduled* and *past*. **Scheduled events** are used to enforce/influence future behavior. They contain information to execute performance measurements, evaluations or concrete rewarding actions in a specified moment in the future. Scheduled events can be canceled or re-scheduled

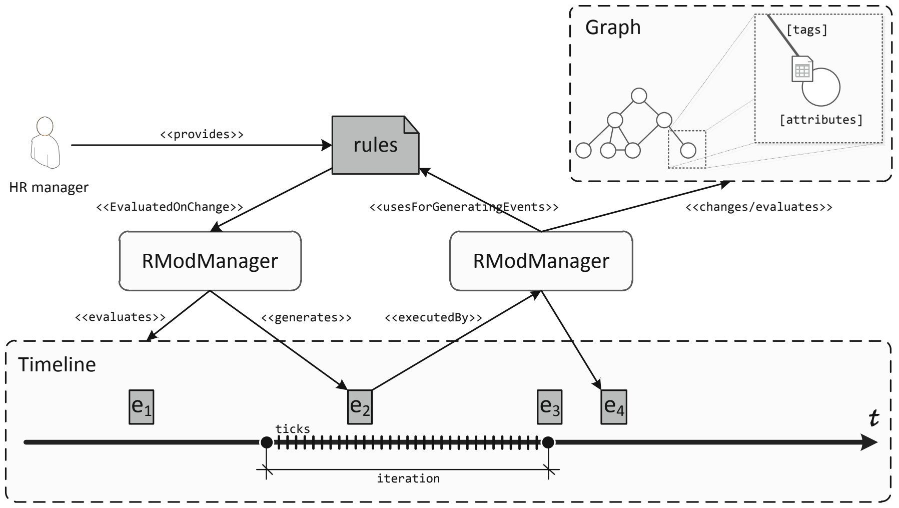


Fig. 2. Components and interactions in RMod

when needed. The timestamp can be expressed either in iterations or clock ticks. Time expressed in clock ticks is fixed, whereas time expressed in iterations is automatically recalculated to an appropriate clock tick if the iteration duration is altered. This can be useful in many real-world situations. For example, we want Christmas bonuses to be paid out on a fixed date, while if a process stage is prolonged due to some unexpected events, we want to reschedule the current iteration and perform the rewarding only at its end. When the time to execute an event is reached, the contained action is executed and the results stored back in the event, which is then archived and put into past state. After that point, the purpose of the **past event** is to serve as a historical reference for future evaluations of workers. An event execution can generate new events, or perform modifications of the team structure and worker attributes. Events are initially generated by executing *rewarding rules*. The rules encode an actual rewarding mechanism provided by the principal. Those rules that fulfill the execution condition generate new event objects to be stored in the timeline. Rules also contain the various bits of logic that get embedded into event objects.

Figure 2 describes a typical working cycle of our Rewarding Model (RMod). Rules provide the necessary logic for performing evaluations and rewarding actions. At every clock tick rules get evaluated. Only the rules that fulfill a logical condition will be triggered to execute. The rules examine the current state of the model and, if an action needs to be performed, produce one or more events. The action contained in the event will include the logic contained in the rule. The events then get stored into the timeline. When the appropriate time comes, the events get executed, modifying the attributes and the graph structure, and possibly spawning new events. The *RModManager* boxes in Figure 2 represent the system that implements the functionalities and manipulates RMod.

The RMod allows us to express and compose different incentive mechanisms. For example:

– *"At the end of iteration, award each contributor who scored better than the average score of his neighbors in that iteration."*
– *"Reward every worker (contributor) who within the last* n *iterations scored a score* t *or greater in at least* k *iterations (*$k \leq n$*)."*
– *"Assign the person with most check-ins at a place a 'Mayor' badge."*
– *"Unless the productivity increases to a level* p *within* n *next iterations, replace team's current manager with the most-trusted of his subordinate workers."*

3 Conclusions and Future Work

Considering the lack of general methods for defining and composing incentive mechanisms for social business process, in this paper we analyze common components of incentive mechanisms and devise novel techniques for modeling and representing incentive schemes suitable for emerging, social-based processes. Our Rewarding Model supports expressing, composing and executing customized, complex incentive schemes. We are currently developing a prototype and intend to illustrate our techniques with real-world scenarios that cover the most important aspects of rewarding. Furthermore, we are integrating our model with the Social Compute Unit[2], a framework for on-demand virtual team provisioning for managing distributed large-scale software systems in clouds.

References

1. Austin, T., Drakos, N., Rozwell, C., Landry, S.: Business Gets Social, `http://www.gartner.com/DisplayDocument?doc_cd=207424&amp;ref=g_noreg`
2. Dustdar, S., Bhattacharya, K.: The Social Compute Unit. IEEE Internet Computing 15(3), 64–69 (2011)
3. Laffont, J.J., Martimort, D.: The Theory of Incentives: The Principal-Agent Model. Princeton University Press, New Jersey (2002)
4. Lazear, E.P., Shaw, K.L.: Personnel economics: The economist's view of human resources. Journal of Economic Perspectives 21(4), 91–114 (2007)
5. Mason, W., Watts, D.J.: Financial incentives and the "performance of crowds". ACM SIGKDD Explor. Newsl. 11(2), 100–108 (2010)
6. Prendergast, C.: The provision of incentives in firms. Journal of Economic Literature 37(1), 7–63 (1999)
7. Shaw, A.D., Horton, J.J., Chen, D.L.: Designing incentives for inexpert human raters. In: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work, CSCW 2011, pp. 275–284. ACM (2011)
8. Yogo, K., Shinkuma, R., Takahashi, T., Konishi, T., Itaya, S., Doi, S., Yamada, K.: Differentiated incentive rewarding for social networking services. In: 10th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), pp. 169–172. IEEE Computer Society (2010)

The Difficulty of Replacing an Inclusive OR-Join

Cédric Favre and Hagen Völzer

IBM Research - Zurich, Switzerland
{ced,hvo}@zurich.ibm.com

Abstract. Some popular modeling languages for business processes, e.g., BPMN, contain inclusive OR-joins (*IOR-joins*), but others, e.g., Petri nets, do not. Various scenarios in Business Process Management require, or benefit from, translating a process model from one language to another. This paper studies whether the control flow of a process containing IOR-joins can be translated into a control flow without IOR-joins.

First we characterize which IOR-joins can be replaced locally and define a local replacement for each replaceable IOR-join. Then, we present examples that cannot be locally replaced but have a more general translation. We give a non-local replacement technique, together with its condition of applicability, which runs in polynomial time. Finally, we show that there exist simple process models with an IOR-join that cannot be replaced – in the sense that its synchronization behavior cannot be obtained by any combination of AND and XOR gateways. The proof reveals an intrinsic limitation on the replaceability of IOR-joins and hence the translatability of BPMN-like control flow into Petri nets.

1 Introduction

Different BPM tools, execution engines, and scientific analysis techniques are based on different modeling languages for business processes. This generates a general interest in translating models from one language into another. In particular, business processes are in practice often modeled in industrial languages such as BPMN and EPCs whereas many analysis techniques, such as control-flow analysis, cost estimation, performance analysis, and process mining are based on Petri nets.

One major obstacle to translate process models between those industrial languages and Petri nets is the presence of gateways with inclusive OR (IOR) logic in the industrial languages. An IOR gateway forks or joins a variable set of threads, thereby supporting various workflow patterns [9]. The IOR-join, which has a *non-local* semantics, is difficult to translate to Petri nets because the semantics of a Petri net transition is *local*. That is, the enablement and effect of a transition in a Petri net relates only to its adjacent places—a small part of the state of the Petri net—whereas the enablement of an IOR-join may depend on the entire state of the process model.

In this paper, we study the question to what extent a process model with IOR-joins can be translated into a process model without IOR-joins. More precisely, we focus on the control flow of a business process, which is modeled by a workflow graph. We ask whether a workflow graph with IOR-joins can be translated into a workflow graph with only XOR and AND gateways. Workflow graphs with XOR and AND gateways can be easily translated into an isomorphic Petri net [8].

We focus on acyclic workflow graphs, for which the IOR-join semantics is simpler. This will allow us to provide replacement strategies for IOR-joins. Conversely, it will already suffice to display simple workflow graphs in which, in some formal sense, an IOR-join cannot be replaced. This will reveal an intrinsic limitation on the replacability of IOR-joins and hence the translatability of the workflow graph of general process models into Petri nets.

The requirements of a translation from a workflow graph with IOR-joins into a workflow graph without IOR-joins can vary for different use cases. To obtain general, yet useful results, we take the following requirements into account:

– The translated workflow graph must have *equivalent* behavior. Many notions of equivalence exist [10]. The adequacy of an equivalence for the translation depends on the use case. We will present the equivalences we use later in the paper. Note that this requirement is by itself not challenging as one can easily ‘unfold’ an acyclic workflow graph into its finite full behavior (i.e., computation tree) and then encode this ‘unfolding’ as a sequential workflow graph. Such a construction would preserve, depending on its precise execution, many popular behavioral equivalences, such as trace equivalence and bisimulation. However, the obtained translation is in general exponentially larger than the original workflow graph. Therefore, we require that
– the size of the obtained workflow graph must be manageable. An exponential blowup is usually not acceptable. We are not aware of any general translation from workflow graphs with IOR-joins into Petri nets that preserves the behavior and does not incur an exponential blowup.
– Furthermore, we are interested to preserve the structure of the workflow graph as much as possible. This is important if we want to map analysis results between the two workflow graphs. For example, in order to return to the user of an analysis technique the results in terms of the original process model or, when monitoring or administrating a process, to understand a trace or a state of the running process in terms of the original process model.

The paper is structured as follows. In Sect. 3, we will first consider *local replacements* of IOR-joins. This translation strategy consists in replacing an IOR-join by a *partial workflow graph* that connects to the edges left dangling by the removal of the IOR-join. Such a local replacement fully maintains, apart from the IOR-join, the original workflow graph and thus makes the mapping to the original workflow graph trivial. Moreover, it leads to a very intuitive notion of equivalence: the partial workflow graph must have the same behavior as the IOR-join. We characterize under which conditions a local replacement is possible in an acyclic workflow graph and define a replacement for these cases. In Sect. 4, we consider a non-local translation strategy and characterize its condition of application. A non-local replacement essentially still retains the structure of the original workflow graph and allows us to replace some IOR-joins that have no local replacement. In Sect. 5, we relax our notion of replacement even more, and we show that even then, there exist simple acyclic workflow graphs that have IOR-joins that cannot be replaced. In Sect. 6, we relate this result and its implications to the translations of workflow graphs containing IOR-joins into Petri nets. Most proofs are omitted in this version but are available in an extended version, which is available as a technical report [5].

2 Workflow Graphs

In this section, we define the necessary fundamental notions, which include workflow graphs and their semantics.

A *directed multi-graph* $G = (N, E, c)$ consists of a set N of nodes, a set E of *edges* and a mapping $c : E \to (N \cup \{\text{null}\}) \times (N \cup \{\text{null}\})$ that maps each edge to an ordered pair of nodes or a null value. If $c(e) = (s, t)$, then s is called the *source* of e, t is called the *target* of e; e is an *outgoing* edge of s, and e is an *incoming* edge of t. If $s =$ null, then we say that e is a *source* of the graph. If $t =$ null, then we say that e is a *sink* of the graph. For a node $n \in N$, the set of incoming edges of n is denoted by $\circ n$. The set of outgoing edges of n is denoted $n\circ$.

Let $G = (N, E, c)$ be an acyclic multi-graph. If x_1, x_2 are two elements in $N \cup E$ such that there is a non trivial path from x_1 to x_2, then we say that x_1 *precedes* x_2, denoted $x_1 < x_2$, and x_2 *follows* x_1.

A *partial workflow graph (pwfg)* $W = (N, E, c, l)$ consists of a multi-graph $G = (N, E, c)$ and a mapping $l : N \to \{\text{AND}, \text{XOR}, \text{IOR}, \text{task}\}$ that associates a *logic* with every node $n \in N$. A *workflow graph* is a partial workflow graph $W = (N, E, c, l)$, such that: 1. W has exactly one source and at least one sink. 2. For each node $n \in N$, there exists a path from the source to one of the sinks that contains n.

Figure 1 depicts an acyclic workflow graph. A rectangle represents a task node. A diamond containing a plus symbol represents a node with AND logic, an empty diamond represents a node with XOR logic, and a diamond with a circle inside represents a node with IOR logic. A node with a single incoming edge and multiple outgoing edges is called a *split*. A node with multiple incoming edges and a single outgoing edge is called a *join*. A node with AND, IOR, or XOR logic is called *gateway*. For the sake of presentation simplicity, we do not use gateways with multiple incoming edges and multiple outgoing edges. It will become clear later that this restriction does not reduce the expressiveness of workflow graphs because such gateway could be represented by a split and a join of the same logic. Nodes with task logic always have a single incoming edge and a single outgoing edge. In the rest of this paper, we only consider acyclic workflow graphs, which simplifies some of our definitions such as the semantics of the IOR-join.

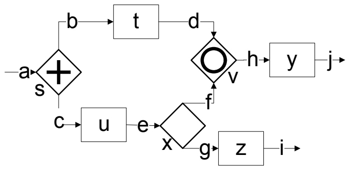


Fig. 1. A workflow graph

The semantics of workflow graphs is, similarly to Petri nets, defined as a token game. Let $W = (N, E, c, l)$ be a workflow graph. A *marking* of W is represented by tokens on the edges of W, i.e., a *marking* is a mapping $m : E \to \mathbb{N}$. We write $m[e]$ instead of $m(e)$. When $m[e] = k$, we say that the edge e *is marked with* k *tokens* in m. When $m[e] > 0$, we say that m *marks* e. The *initial marking* m_s of W is such that the source edge is marked with exactly one token in m_s and m_s does not mark any other edge. If a node n of a workflow graph has AND or task logic, executing n removes one token from each of the incoming edges of n and adds one token to each of the outgoing edges of n. If n has XOR logic, executing n removes one token from one of the incoming edges of n and adds one token to one of the outgoing edges of n. If n has IOR logic, n can be executed if and only if at least one of its incoming edges is marked and there is no marked edge that precedes a non-marked incoming edge of n. When n executes, it removes one token from each of its marked incoming edges and adds one token to a non-empty subset of its outgoing edges. This IOR semantics, which is explained in detail elsewhere [12], complies with the BPMN 2.0 standard and BPEL's dead path elimination [1]. The choice of the set of outgoing edges to which a token is added when executing a node with XOR or IOR logic is non-deterministic. In the following, this semantics is defined formally:

A triple (E_1, n, E_2) is called a *transition* if $n \in N$, and any of the following propositions:

- $l(n) = \text{AND}$ or $l(n) = \text{task}$, $E_1 = \circ n$, and $E_2 = n\circ$.
- $l(n) = \text{XOR}$, there exists an edge $e \in \circ n$ such that $E_1 = \{e\}$, and there exists an edge $e' \in n\circ$ such that $E_2 = \{e'\}$.
- $l(n) = \text{IOR}$, $E_1 \subseteq \circ n$, $E_2 \subseteq n\circ$, and E_1 and E_2 are non-empty.

Let m and m' be two markings of W. A transition (E_1, n, E_2) is *enabled* in a marking m if, for each edge $e \in E_1$, we have $m[e] > 0$ and, if $l(n) = IOR$, $E_1 = \{e \in \circ n \mid m(e) > 0\}$ and for every edge $e \in \circ n \setminus E_1$, there exists no edge e', marked in m, such that $e' < e$. A transition t can be *executed* in a marking m if t is enabled in m. When t is executed in m, a marking m' results such that: $m'[e] = m[e] - 1$ if $e \in E_1$, $m'[e] = m[e] + 1$ if $e \in E_2$, and $m'[e] = m[e]$ otherwise. We write $m_1 \xrightarrow{t} m_2$, when a transition t is enabled in a marking m_1 and its execution results in a marking m_2.

An *execution sequence* of W is a finite alternating sequence $\sigma = \langle m_0,\ t_0,\ m_1,\ ...,\ m_n\rangle$ of markings m_i of W and transitions $t_i = (E_i, n_i, E'_i)$ such that, for each $i \geq 0$, t_i is enabled in m_i and m_{i+1} results from the execution of t_i in m_i. We write $m_1 \xrightarrow{\sigma} m_2$ when m_1 is the first marking and m_2 is the last marking of σ. An execution sequence σ *marks* an edge e if there exists a marking of σ that marks e. An *execution* of W is a (finite) execution sequence $\sigma = \langle m_0, ..., m_n\rangle$ of W such that $m_0 = m_s$ and there is no transition enabled in m_n. As the transition between two markings can be easily deduced, we often omit the transitions when representing an execution or an execution sequence, i.e., we write them as sequence of markings. We only consider finite executions and finite execution sequences because we only discuss acyclic workflow graphs.

Let m be a marking of W, m is *reachable from* a marking m' of W if there exists an execution sequence $\sigma = \langle m_0, ..., m_n \rangle$ of W such that $m_0 = m'$ and $m = m_n$. The marking m is a *reachable marking* of W if m is reachable from m_s. The marking m is a *(local) deadlock* if there exists a non-sink edge $e \in E$ that is marked in m and e is marked in all the markings reachable from m. We say that W is *deadlock-free* if there exists no reachable marking m of W such that m is a deadlock. The marking m is a *lack of synchronization* (or *unsafe*) if there exists an edge $e \in E$ that is marked by more than one token in m. We say that a workflow graph W *contains* a lack of synchronization if there exists a reachable marking m of W such that m is a lack of synchronization. A workflow graph is *sound* if it is deadlock-free and does not contain a lack of synchronization. Note that this notion of soundness is equivalent to the usual notion of soundness used for workflow nets (see for e.g. [8]).

3 Local Replacements

Fig. 2 and Fig. 3 illustrate an example of a *local replacement* of an IOR-join. In this example, the IOR-join j is replaced by the partial workflow graph composed of the nodes v and w, and the edge i. As a graphical convention, we represent the elements of the original workflow graph using plain lines and the elements introduced to replace an IOR-join using dashed lines. In the following, we omit tasks in the workflow graphs when they are not relevant for our discussion.

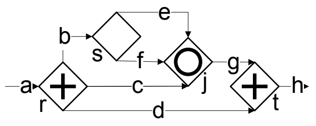


Fig. 2. A workflow graph

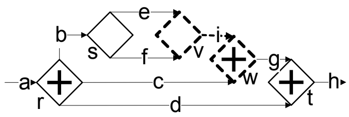


Fig. 3. Local replacement of j by the partial workflow graph composed of the nodes v and w, and the edge i

A local replacement of an IOR-join j is a partial workflow graph R that connects to the edges left dangling by the removal of j. Note that j and R have exactly the same set of incoming and outgoing edges. Such a replacement is a very intuitive way to replace an IOR-join. Apart from the IOR-join, a local replacement preserves the original workflow graph, which makes it very easy to relate the original and the translated workflow graph. We formalize a local replacement as follows:

Definition 1 (Local replacement). *Let $W = (N, E, c, l)$ be a workflow graph and j be a node in N. Let $R = (N'', E'', c'', l'')$ be a partial workflow graph such that for each node $n \in N''$, $l''(n) =$ XOR or $l''(n) =$ AND, $N \cap N'' = \emptyset$, and $E \cap E'' = \emptyset$.*

A local replacement *of j in W by R results in a workflow graph $W' = (N', E', c', l')$ such that:*

- $N' = N \setminus \{j\} \cup N''$,
- $E' = E \cup E''$,

– $c'(e) = c''(e)$ *when* $e \in E''$,
  $c'(e) = c(e) = (s,t)$ *when* $e \in E$ *and* $s \neq j \neq t$,
  $c'(e) = (s,t)$ *such that* $s \in N''$ *and* $t \in N$ *iff* $e \in E$ *and* $c(e) = (j,t)$,
  $c'(e) = (s,t)$ *such that* $t \in N''$ *and* $s \in N$ *iff* $e \in E$ *and* $c(e) = (s,j)$,
– $l'(n) = l(n)$ *when* $n \in N \setminus j$, $l'(n) = l''(n)$ *when* $n \in N''$, *and*
– *each element* $x \in N'' \cup E''$ *is on a path in* W *from an edge* $e_{in} \in E$ *such that* $c(e_{in}) = (s,j)$ *to the edge* $e_{out} \in E$ *such that* $c(e_{out}) = (j,t)$.

Intuitively, the workflow graph W' resulting from the local replacement of an IOR-join j in a workflow graph W by a partial workflow graph R is *equivalent* to W iff R has the same "behaviour" as j. We formalize this as follows: Let, in the rest of this section, $W = (N, E, c, l)$ be a workflow graph containing an IOR-join j and $W' = (N', E', c', l')$ be a workflow graph obtained by local-replacement of j by a partial workflow graph R. A transition (E_1, n, E_2) of W' is a *replacement transition* iff $n \in (N' \setminus N)$. An execution sequence σ of W' is a *replacement execution sequence* iff each transition of σ is a replacement transition and after σ no replacement transition is enabled and no edge $e \in E' \setminus E$ is marked. W and W' are *equivalent* iff the following two conditions are met:

1. Let m_1 and m_2 be two reachable markings of W. For any transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W, there exists a replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W'.
2. Let m_1 and m_2 be two reachable markings of W' such that m_1 and m_2 only mark edges in E. For any replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W', there exists a transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W.

Some IOR-joins can easily be replaced locally: It is clear that we can replace an IOR-join by an AND-join if all its incoming edges are marked everytime it is executed, and by an XOR-join if only one of its incoming edge is marked everytime it is executed [13]. For an acyclic workflow graph, we have shown elsewhere [4] how to compute these properties in quadratic time with respect to the size of the workflow graph. Furthermore, a workflow graph completion heuristic based on the *refined process structure tree* [11] can also provide a local replacement for some IOR-joins.

In the following, we give a local replacement technique which, as we will see later, can provide a local replacement for any IOR-join that can be replaced locally. Then, we characterize under which conditions an IOR-join can be replaced locally, i.e., regardless of the replacement technique. This result is an extension of a technique [11] that completes a workflow graph with multiple sinks to obtain a workflow graph with a single sink.

First, we define the notions of test and cover which will allow us to formulate a replacement based on covers:

Definition 2 (Mutually exclusive edges, test, and cover). *Let $W = (N, E, c, l)$ be a workflow graph.*

Two edges in E are mutually exclusive *iff there exists no execution of W which marks both edges.*

A test of *an edge $e \in E$ is a set $T_e \subseteq E$ of pairwise mutually exclusive edges such that an execution σ of W marks e iff σ marks one of the edges in T_e. If $X \subseteq E$ is a subset of edges such that $T_e \subseteq X$, we say that T_e is a test* over X.

Let $X \subseteq E$ and $e \in E$. A cover of X with respect to e *is a set $\mathcal{C}$ of tests of e over X such that each edge in X belongs to a test in $\mathcal{C}$.*

In Fig. 2, the tests $T_1 = \{e, f\}$ and $T_2 = \{c\}$ of g form a cover $\mathcal{C} = \{T_1, T_2\}$ of $\circ j$ with respect to g. We now describe how to obtain a local replacement of an IOR-join using a cover of its incoming edges with respect to its outgoing edge. We shall see later that such cover does not always exist. Fig. 4 illustrates the structure of the replacement:

Definition 3 (Cover-based replacement). *Let j be an* IOR*-join in a workflow graph W and o be the outgoing edge of j. Let $\mathcal{C}$ be a cover of $\circ j$ with respect to o. Let the $R = (N'', E'', c'', l'')$ be a partial workflow graph defined as follows:*

- *N'' contains: an* AND*-split a_e for each edge $e \in \circ j$, an* XOR*-join x_i for each test $T_i \in \mathcal{C}$, and one* AND*-join f.*
- *For each test $T_i \in \mathcal{C}$, for each edge $e \in T_i$, E'' contains an edge from the* AND*-split a_e to the* XOR*-join x_i. For each* XOR*-join x_i, E'' contains an edge from x_i to the* AND*-join f.*

The Cover-based replacement *(*C-replacement *for short) of j replaces j by R as follows: The target of each (previously) incoming edge e of j is set to be a_e. The source of the (previously) outgoing edge o of j is set to f.*

Note that, when an edge e in $\circ j$ belongs to only one test, the AND-split a_e has a single outgoing edge and can be removed. When a test T_i contains only one edge, the XOR-join x_i has a single incoming edge and can be removed. The local replacement illustrated by Fig. 2 and Fig. 3 is the result of a C-replacement using the cover $\mathcal{C} = \{\{e, f\}, \{c\}\}$ of $\circ j$ with respect to the outgoing edge g of j. Because each edge is used only in one test, there is no AND-split necessary. Moreover, the test $\{c\}$ contains only one edge, therefore it does not require an XOR-join.

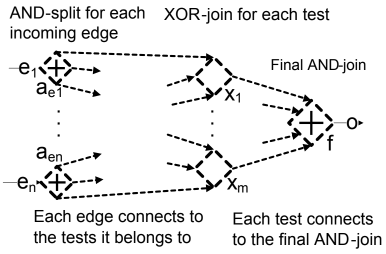

AND-split for each incoming edge
XOR-join for each test
Final AND-join
e1
ae1
aen
en
x1
xm
o
f
Each edge connects to the tests it belongs to
Each test connects to the final AND-join

Fig. 4. Canvas of local replacement based on a cover

Computing tests, including checking that edges are mutually exclusive can be done using state space exploration, which can take exponential time. More efficient heuristics exist for some special cases. For example, it is possible to compute in quadratic time whether a set of edges in an acyclic process is mutually exclusive [4]. Efficient computation of the tests is out of scope of this paper.

We can now characterize the conditions under which an IOR-join has an equivalent local replacement:

Theorem 1 (Equivalent local replacement existence). *Let W be a sound workflow graph containing an* IOR*-join j.*

An IOR*-join j has an equivalent local replacement iff there exists a cover of $\circ j$ with respect to the outgoing edge of j.*

While Thm. 1 applies to any local replacement technique, the proof of the 'if' direction [5] shows that, whenever there exists a cover of $\circ j$ with respect to the outgoing edge of j, the C-replacement of j produces an equivalent workflow graph.

Fig. 5 is a slight variation of Fig. 2 where changing the target of the edge e makes it impossible replace the IOR-join locally. By changing the target of e, e does not belong to $\circ j$ anymore. Therefore, the test $T_1 = \{e, f\}$ cannot be used to build a cover of $\circ j$ anymore and there is no other test of g that contains f.

In Fig. 6, the IOR-join cannot be replaced locally because e does not belong to any test of h contained in $\circ j$, i.e., there exists no cover of h. We will see in the next section how the IOR-joins of these two examples can be replaced using a non-local replacement.

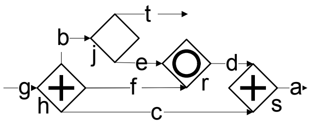


Fig. 5. The edge f does not belong to any test of g that is a subset of $\circ j$

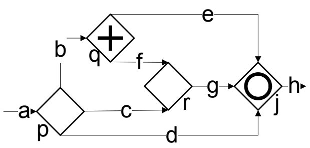


Fig. 6. The edge e does not belong to any test of h that is a subset of $\circ j$

4 Non-local Replacements

Fig. 8 shows an example of a non-local replacement where the IOR-join j of Fig. 7 is replaced by the partial workflow graph composed of the nodes w, x and the edges i, e' where, additionally, the AND-split v was inserted on the edge c which delivers additional (non-local) information to the IOR-join replacement via the edge i.

So, in addition to a local replacement, we allow non-local replacements to insert additional AND-splits in the graph, which can only be connected to the IOR-join replacement. These AND-splits only "copy" tokens to route them to the replacement and do not alter the original behavior of the process. This also preserves the graph structure to a large extent.

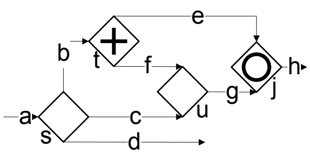
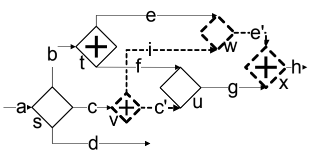


Fig. 7. A workflow graph



Kiepuszewski et al. [6, Proof of Theorem 5.1] give a completion approach to transform a Petri net with

Fig. 8. Non-local replacement of j in Fig. 7

multiple sinks into a Petri net with a single sink. In this section, we will show that one can use a variation of that approach, which we call *K-replacement*, to replace an IOR-join in an acyclic workflow graph. We give a condition that characterizes the IOR-joins for which this replacement produces an equivalent workflow graph. Finally, we show that checking whether replacing the IOR-join produces an equivalent workflow graph can be done in polynomial time and that the replacement itself requires polynomial time.

Non-local Replacement and Equivalence: First, we formalize the concept of non-local replacement. When *inserting* a gateway g on an edge e, we create an additional edge e' such that the source of e' is g and the target of e' is the target of e and the target of e becomes g. We say that the edge e' is the *resulting edge* from the insertion of g on e.

Definition 4 (Non-local replacement). *Let $W = (N, E, c, l)$ be a workflow graph and j be a node in N. Let $R = (N'', E'', c'', l'')$ be a partial workflow graph such that for each node $n \in N''$, $l''(n) = \text{XOR}$ or $l''(n) = \text{AND}$, $N \cap N'' = \emptyset$, and $E \cap E'' = \emptyset$. Let $l_g =< a_0, \dots, a_n >$ be a list of* AND*-splits such that $l_g \cap (N \cup R) = \emptyset$.*

A non-local replacement *of j in W by R and l_g results in a workflow graph $W' = (N', E', c', l')$ obtained by the* insertion *of l_g on some edges $< e_0, \dots, e_n >$ of W resulting in a list l_e of edges and the* insertion *of R such that:*

- *$N' = N \setminus \{j\} \cup N'' \cup l_g$,*
- *$E' = E \cup E'' \cup l_e$,*
- *$c'(e) = c(e) = (s, t)$ when $e \in E$, $s \in N$, $t \in N$, and $s \neq j \neq t$,*
  $c'(e) = c''(e) = (s, t)$ when $e \in E''$, $s \in N''$, and $t \in N''$,
  $c'(e) = (s, t)$ such that $s \in N$ and $t \in N''$ iff $e \in E$ and $c(e) = (s, j)$ or $e \in E''$ and $s \in l_g$,
  $c'(e) = (s, t)$ such that $s \in N''$ and $t \in N$ iff $e \in E$ and $c(e) = (j, t)$,
- *$l'(n) = l(n)$ when $n \in N \setminus j$, $l'(n) = l''(n)$ when $n \in N''$, and*
- *each element $x \in N'' \cup E''$ is on a path in W from an edge $e_{in} \in E$ such that $c(e_{in}) = (s, j)$ or a node $a_i \in l_g$ to the edge $e_{out} \in E$ such that $c(e_{out}) = (j, t)$.*

We now define when a non-local replacement is semantically correct through a notion of equivalence of the two workflow graphs. Let, in the rest of this section, $W = (N, E, c, l)$ be a workflow graph containing an IOR-join j and $W' = (N', E', c', l')$ be a workflow graph obtained by non-local replacement of j. To define equivalence we need to map the markings of W and W'. We first define a mapping $\psi : E' \to E \cup \{null\}$ such that for any edge e' in E':

$$\psi(e') = \begin{cases} e' & \text{if } e' \in E, \\ e & \text{if } e' \text{ is the resulting edge of the insertion of an AND-join on } e, \\ null & \text{otherwise.} \end{cases}$$

We define a mapping ϕ from a marking of W' to a marking of W such that $\phi(m)[e] = \sum_{\psi(e')=e} m[e']$.

We reuse the notion of *replacement execution sequence* defined in Sect. 3.

W and W' are *equivalent* iff for any pair of reachable markings m_1 of W, m'_1 of W' such that $m_1 = \phi(m'_1)$, we have:

1. for any transition $t = (E_1, j, E_2)$ and any marking m_2 such that $m_1 \xrightarrow{t} m_2$ in W, there exists a replacement execution sequence σ and a marking m'_2 such that $m'_1 \xrightarrow{\sigma} m'_2$ in W' and $m_2 = \phi(m'_2)$, and
2. for any replacement execution sequence σ and any marking m'_2 such that $m'_1 \xrightarrow{\sigma} m'_2$ in W', there exists a transition $t = (E_1, j, E_2)$ and a marking m_2 such that $m_1 \xrightarrow{t} m_2$ in W and $m_2 = \phi(m'_2)$.

A Simple Non-local Replacement: As intermediary step, we discuss informally a simple version of the technique, which we call *simple K-replacement*. We will then point out a shortcoming of simple K-replacement and modify it to obtain the *K-replacement*.

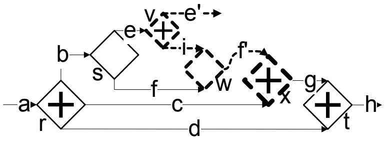


Fig. 9. Non-local replacement of j in Fig. 5

Fig. 9 and Fig. 10 show equivalent non-local replacements for the workflow graphs in Fig. 5 and Fig. 6, respectively. These are two examples of simple K-replacement.

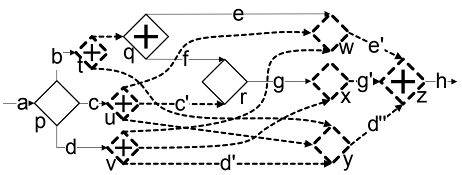


Fig. 10. Non-local replacement of j in Fig. 6

The simple K-replacement uses the notion of a *bridge*: A *bridge* from an edge e to an edge e' is a path from e to e' such that each split on the path is an AND-split and each join on the path is an XOR-join. For example, in Fig. 9, the path $< e, v, j, w, f' >$ is a bridge from e to f'. The existence of a bridge from e to e' implies that every execution which marks e also marks e'.

The key idea of simple K-replacement is to replace an IOR-join by an AND-join and to ensure that every incoming edge of the new AND-join carries a token in every execution by adding suitable bridges. More precisely, for each incoming edge e' of the AND-join and each outgoing edge e of any XOR-split such that e is not on a path to e', we create a bridge from e to e'. Intuitively, for each outgoing edge e of an XOR-split that removes a token from a path to e', we add a bridge that brings an additional token to e' on a different path. This leads to equivalent workflow graphs for the examples in Fig. 5 and Fig. 6.

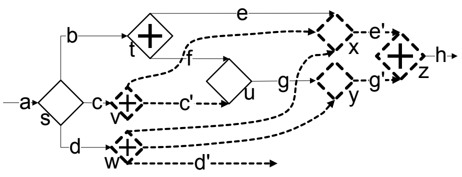


Fig. 11. Non-local replacement of j in Fig. 7

However, consider the workflow graph in Fig. 11 obtained by the same technique for the IOR-join v in the workflow graph shown by Fig. 7. When an execution σ marks the edge d of the workflow graph in Fig. 11, the edge h is also marked by σ which is not the case when an execution marks the edge d in the workflow graph in Fig. 7. Thus, simple K-replacement does not lead to an equivalent workflow graph when applied to an IOR-join that is not executed by every execution of the original workflow graph.

K-Replacement: We now describe our generalized technique, called *K-replacement*. Applying K-replacement to f in Fig. 7 results in the workflow graph in Fig. 8. K-replacement uses the notion of *dominator frontier* to apply the same replacement strategy as the simple K-replacement to a sub-graph of the workflow instead of the complete graph. Applying the K-replacement to a sub-graph of the workflow graph implies that the AND-join replacing the IOR-join only executes during the execution that marks an edge of this sub-graph and thus allows us to produce an equivalent replacement in more cases than with simple K-replacement.

To this end, we use the following notions. A node x_1 *dominates* another node x_2 if each path from the source edge of the workflow graph to x_2 contains x_1. A dominator x_1 of a node x_2 is the *minimal dominator* of x_2 iff there exists no node x_1' such that x_1' dominates x_2, x_1 dominates x_1', and $x_1 \neq x_1' \neq x_2$. For example, in Fig. 3, the nodes r and s dominate the node v and the node s is the minimal dominator of v. A set E_d of edges is the *dominator frontier* of a node x_2 iff a node x_1 is the minimal dominator of x_2, $E_d \subseteq x_1\circ$, for each edge $e \in E_d$, $e < x_2$, and for each edge $e' \in ((x_1\circ) \setminus E_d)$, we have $e' \not< x_2$. For example, in Fig. 7, the dominator frontier of j is the set of edges $\{b, c\}$.

K-replacement replaces an IOR-join j by an AND-join. Furthermore, a bridge from e to e' is created for each incoming edge e' of j and each edge e such that e is the outgoing edge of an XOR-split on a path from an edge of the dominator frontier of j to j, and there is no path from e to e'. K-replacement is detailed further by Algorithm 1. As mentioned earlier, Fig. 8 shows the workflow graph resulting from the application of Algorithm 1 to the IOR-join j in Fig. 7.

The K-replacement implies that the AND-join replacing the IOR-join executes in every execution where an edge of the dominator frontier is marked. Thus, intuitively, the K-replacement produces an equivalent workflow graph if each execution that marks one edge of the dominator frontier also executes the IOR-join. In the following, we formalize this intuition as a necessary and sufficient condition for the K-replacement to produce an equivalent workflow graph:

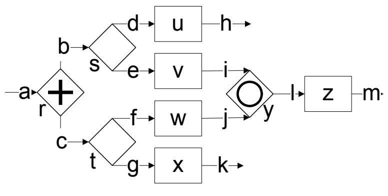


Fig. 12. The IOR-join y cannot be replaced

Theorem 2 (K-replacement applicability). *K-replacement of an* IOR*-join j in a workflow graph W produces a workflow graph that is equivalent with W iff j is executed in each execution of W in which an edge of the dominator frontier of j is marked.*

Algorithm 1. K-replace(j, W) **input**: A workflow graph $W = (N, E, c, l)$ and and IOR-join $j \in N$. **output**: A workflow graph $W' = (N', E', c', l')$ where j has been K-replaced.

```
Create an AND-join $a$ and set the source of the outgoing edge of $j$ to be $a$.
Let $E_I$ be the set of incoming edges of $j$ in $W$.
for each edge $e \in E_i$ do
  Create an XOR-join $x_e$.
  Set the target of $e$ to be $x_e$.
  Create and edge from $x_e$ to $a$.
Let $preset(j)$ be the set of all elements in $E \cup N$ from the minimal dominator of $j$
having a path to $j$
for each edge $e' \in \circ j$ do
  for each decision $d \in preset(j)$ do
    Let $preset(e')$ be the set of all elements in $E \cup N$ from the dominator frontier
    of $j$ having a path to $e'$,
    for each edge $e \in d\circ$ such that $e \notin preset(e'))$ do
      if $d \neq$ minimal dominator of $j$ OR $e \in$ dominator frontier of $j$ then
        if The target of $e$ is not an and split then
          Insert an AND-split $s$ on $e$
        else
          Let $s$ be the target of $e$
        Add an edge from $s$ to $x_{e'}$
```

Thus, K-replacement of the IOR-join j in Fig. 7 produces an equivalent workflow graph shown in Fig. 8 because j executes in an execution σ if and only if an edge of its dominator frontier $\{b, c\}$ is marked by σ. This is not the case for the workflow graph in Fig. 12. The dominator frontier of y in Fig. 12 is the set $\{b, c\}$. The edges b and c are marked by every execution. Thus, applying the K-replacement algorithm to y would thus produce a workflow graph in which the task z is executed during every execution. It is not the case for the workflow graph in Fig. 12 in which the execution where the edges g and d are marked does not execute z.

A Note on Complexity: We now argue that the condition expressed by Thm. 2 can be computed efficiently, i.e., in polynomial time with respect to the size of the workflow graph. Algorithm 1 also runs in polynomial time. This allows us to conclude that we can identify IOR-joins that can be replaced by K-replacement and replace them efficiently:

Theorem 3 (Polynomial time complexity of K-replacement). *Let j be an IOR-join, and W be the workflow graph containing j.*

1. *Computing whether the K-replacement of j produces a workflow graph that is equivalent to W can be done in polynomial time with respect to the size of W.*
2. *The K-replacement of an* IOR*-join j can be computed in polynomial time with respect to the size of W.*

5 The Difficulty of Replacing an IOR-Join

As discussed above, the IOR-join y in Fig. 12 cannot be replaced correctly with K-replacement. In this section, we provide an argument why it is difficult to implement the IOR-join in Fig. 12 with any combination of AND and XOR gateways.

Recalling the discussion in Sect. 1 we have to specify an equivalence and we require some structure to be preserved in order to rule out some 'simple' implementations that incur an exponential blowup.

In this section, we take the view that the IOR-join synchronizes its incoming branches, where these incoming branches have a certain 'forking' logic, depending on the gateway structure 'before' the IOR-join. The forking logic for the example in Fig. 12 is represented by the workflow graph in Fig. 13. We will show that the workflow graph in Fig. 13 cannot be completed with any combination of AND and XOR gateways to produce a behavior equivalent to the behavior of the workflow graph in Fig. 12. In this sense, no combination of AND and XOR gateways can produce the synchronization behavior of the IOR-join in Fig. 12.

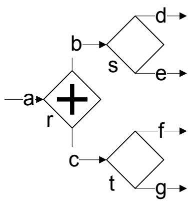


Fig. 13. Prefix that cannot be completed

We consider the workflow graph in Fig. 13 as a *prefix* of a possible implementation. A *prefix* $P = (N_p, E_p, c_p, l_p)$ of a workflow graph $W = (N, E, c, l)$ is a workflow graph that is a subgraph of $N \cup E$ such that for each pair e_1, e_2 of elements of W such that $e_1 < e_2$, we have: If e_2 belongs to P, then e_1 belongs to P.

Rather than picking a concrete behavioral equivalence (cf. discussion in Sect. 1), we formalize properties that a workflow graph must have to be equivalent to the workflow graph in Fig. 12. We allow multiple tasks of the implementing workflow graph to be labeled with z and therefore to correspond to the task z in Fig. 12.

Definition 5 (Equivalent workflow graph properties).
Let W' be a workflow graph which has the prefix illustrated by Fig. 13. Let $t_1 = (\{b\}, s, \{d\})$ and $t_2 = (\{c\}, t, \{g\}$. The workflow graph W' satisfies the following properties:

P1 *There exists no execution where t_1, t_2, and a task labeled with z are executed.*
P2 *There exists an execution during which t_1 and a transition t_z executing a task labeled with z are executed and, for each execution where t_1 and t_z are executed, t_1 is executed before t_z.*

It is easy to see that the workflow graph in Fig. 12 satisfies these properties and that notions of equivalence such as the ones that we defined for local and non-local replacements would ensure that any equivalent workflow graph satisfies them too. We can now enunciate our result:

Theorem 4 (The synchronization role of IOR-joins cannot be implemented using only AND and XOR-logic). *There exists no deadlock-free workflow graph W' such that W' has the workflow graph P illustrated by Fig. 13* *as prefix, W' does not contain any IOR-join, and W' satisfies the properties of Def. 5.*

Proof. This proof rests on the following two lemmas:

Lemma 1. *Let W be a deadlock-free workflow graph, e, e' be two edges of W, m be a reachable marking of W which marks e.*

If there exists a path p from e to e' in W, then there exists a marking m', reachable from m, such that m' marks e'.

Lemma 2. *Let W be a deadlock-free acyclic workflow which does not contain IOR-joins. Let $t = (E_1, n, E_2)$ and $t' = (E'_1, n', E'_2)$ be two transitions of W.*

If, in each execution σ of W where t and t' occur, we have that t occurs before t' during σ, then there exists a path from an edge $e \in E_2$ to an edge $e' \in E'_1$.

Proof of Thm. 4: *We prove this theorem by contradiction: Suppose that there exists a workflow graph W' such that W' has P (illustrated by Fig. 13) as prefix, does not contain an IOR-join, and satisfies P1 and P2.*

By P2, we have that t_1 and a transition $t_z = (E_1, n, E_2)$ such that n is a task labeled z occur together in some execution and that t_1 always occur before t_z when both occur. By Lemma 2, there exists a path p from d to the incoming edge of n. Consider an execution sequence $\sigma = \langle [a], (\{a\}, r, \{b, c\}), [b, c], t_1, [d, c], t_2, [d, g] \rangle$. We can complete σ to obtain the execution σ^* by following p and thus executing t_z (Lemma 1). This contradicts P1 because t_1, t_2, and a task labeled z are executed during σ^*.

6 On the Translation to Petri Nets and Related Work

We have shown that, in some sense, the IOR-join cannot always be replaced by a combination of AND and XOR gateways. Workflow graphs without IOR gateways are essentially equivalent to free-choice workflow nets, a class of Petri nets for which efficient analysis algorithms exist [2]: A workflow graph without IOR gateways can be translated into an isomorphic free-choice workflow net of about the same size [8] but it can also be shown that a free-choice workflow net can be translated into an isomorphic workflow graph without IOR of about the same size. Hence, there is no free-choice workflow net implementing the IOR-join in Fig. 12 in the sense discussed above.

To translate an acyclic workflow graph into a non-free-choice Petri-net, one can use Dead Path Elimination [1,12] to implement the IOR-join with gateways that have a local semantics. Dead Path Elimination uses workflow graphs with individual tokens, where a token can have a value that is either *true* or *false*. Such a workflow graph can be easily translated into a high-level Petri net, which in turn can be unfolded into a non-free-choice Petri net. Alternatively, Mendling, van Dongen, and van der Aalst [7] use the theory of regions to synthesize a Petri net from the reachability graph of the process model. In both approaches, the resulting Petri net is exponentially larger than the original workflow graph and does not preserve the structure of the original process.

A more direct approach to implement the IOR-join from Fig. 12 as a Petri net is shown in Fig. 14, where the IOR-join replacement is delimited by the dashed box. The behavior of this Petri net is equivalent to the workflow graph in Fig. 12. Note that, similar to K-replacement, we provide additional inputs to the IOR-replacement and that this construction preserves most of the structure of the original workflow graph. These additional inputs give the IOR-join replacement information on the edges that have been marked by the execution.

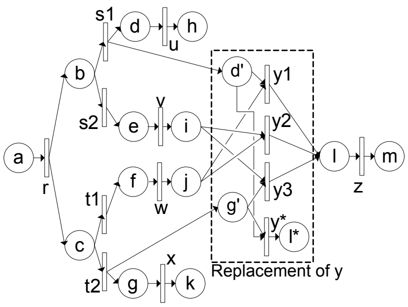


Fig. 14. A non-free-choice Petri net that is equivalent to the one in Fig. 12.

We can then think of the IOR-join as two boolean expressions over these marked edges that fully characterize its executions: the first expression characterizes the executions that lead to a token on the outgoing edge of the IOR-join, the second characterizing all other executions. Both expressions can be easily represented by a non-free-choice Petri net as in the example in Fig. 14, where the transitions y_1, y_2, and y_3 implement the first expression $(d' \wedge j) \vee (i \wedge j) \vee (i \wedge g')$ The transition y^* implements the second expression. The role of y^* is to "purge" the place d' and g' in the second case.

This construction can be defined to implement an IOR-join in any acyclic workflow graph because the full execution history can be provided to the replacement subgraph. However, the Petri net representations of the boolean expressions are exponential in the size of the workflow graph. We leave it as future work to evaluate whether this exponential blowup can be mitigated using simplification techniques for boolean formulas.

To sum up, Theorem 4 gives a strong argument why IOR-joins cannot be easily implemented by a free-choice net, it points to some difficulty when trying to translate to general Petri nets, and no general polynomial-space translation to general Petri nets is known.

7 Conclusion

In this paper, we studied the difficulty of replacing an IOR-join. For acyclic processes, we established a necessary and sufficient condition that characterizes IOR-joins that can be locally replaced. For the IOR-joins that can be locally replaced, we proposed a generic local replacement. We presented a non-local replacement strategy which allows us to translate some of the IOR-joins that cannot be replaced locally. We have shown that computing this replacement and checking its condition of application can be done in polynomial time with respect to the size of the original workflow graph.

While these results have been presented for the replacement of a single IOR-join in a sound acyclic business process model, they can be applied to replace multiple IOR-joins in an acyclic workflow graph. Moreover, it can be shown that both replacement techniques do not alter the soundness of the process, i.e., cannot introduce or fix a control-flow error which makes these replacement strategies applicable to replace IOR-joins in process models of which the soundness is unknown such as, for example, when performing a control-flow analysis. We have used elsewhere [3] *process structure trees* to decompose the workflow graph into fragments. Such a decomposition allows us to factor out cycles and therefore to apply the replacements in processes containing cycles.

We have shown that the synchronization provided by the IOR-join cannot, in general, be implemented by free-choice constructs. Translations of a workflow graph containing an IOR-join into a (non-free-choice) Petri net exist, however, known translations have an exponential blowup and, usually, do not preserve the structure of the original process. These results show a difficulty to translate the non-local semantics of the IOR-join into a modeling language that only contains local gateways and therefore a difficulty to fully leverage Petri net based techniques for process models containing IOR-joins.

References

1. Alves, A., et al.: Web services business process execution language version 2.0. OASIS Standard 11 (2007)
2. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press (1995)
3. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl. Eng. 70(5), 448–466 (2011)
4. Favre, C., Völzer, H.: Symbolic Execution of Acyclic Workflow Graphs. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 260–275. Springer, Heidelberg (2010)
5. Favre, C., Völzer, H.: The Difficulty of Replacing an Inclusive OR-Join. Technical report, IBM Research, RZ3824 (2012)
6. Kiepuszewski, B., ter Hofstede, A., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (2003)
7. Mendling, J., van Dongen, B., van der Aalst, W.M.P.: Getting rid of or-joins and multiple start events in business process models. Enterprise Information Systems 2(4), 403–419 (2008)
8. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.: 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)
9. van der Aalst, W.M.P., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
10. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990)
11. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic Workflow Graph Refactoring and Completion. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 100–115. Springer, Heidelberg (2008)
12. Völzer, H.: A New Semantics for the Inclusive Converging Gateway in Safe Processes. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 294–309. Springer, Heidelberg (2010)
13. Wynn, M., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A., Edmond, D.: Business Process Verification–Finally a Reality! Business Process Management Journal 15(1), 74–92 (2009)

Automatic Information Flow Analysis of Business Process Models

Rafael Accorsi[1] and Andreas Lehmann[2]

[1] University of Freiburg, Germany
rafael.accorsi@iig.uni-freiburg.de
[2] University of Rostock, Germany
andreas.lehmann@uni-rostock.de

Abstract. We present an automated and efficient approach for the verification of information flow control for business process models. Building on the concept of Place-based Non-Interference, the novelty is that Petri net reachability is employed to detect places in which information leaks occur. We show that the approach is sound and complete, and present its implementation, the Anica tool. Anica employs state of the art model-checking algorithms to test reachability. An extensive evaluation comprising over 550 industrial process models is carried out and shows that information flow analysis of process models can be done in milliseconds.

1 Introduction

Business processes (BPs) specify how operational activities are executed to provide a service; for instance steps in a supply-chain or profile updates in customer relationship management. In doing so, they handle sensitive data that must remain confidential. However, structural flaws in BPs design may lead to a violation of such confidentiality requirements [30], i.e., they can cause leaks.

Leaks happen when data or information flows from a secret to a public domain. The former is called *data leak*, while the latter is referred to as *information leak*. A data leak is direct, but illegal access to a data object. An information leak concern the fact that subjects in the public domain can infer secret information. Although certification standards (e.g., ISO/IEC 27001 [23] and TCSEC [36]) demand the identification of both kinds of leaks, current state of the art mainly provides mechanisms to detect data leaks (e.g., [8,9,12,21,26,33]). This paper proposes a mechanism to detect information leaks in BP models.

Information flow control provides a powerful abstraction to reason about information leaks [16]. Assuming that the BP model under analysis is split into two logical security domains (*high* for secret, *low* for public), a BP model is assumed secure with regard to information leaks if it enforces *non-interference*; that is, the actions in the high domain do not produce an observable effect (*interference*) in the low domain. Details on the security domains follow in Sect. 2. By showing non-interference for a BP model one thus ensures that subjects in the low domain cannot deduce information about the high domain, thereby guaranteeing its confidentiality and isolation.

We consider the Petri net representation of BP models as a basis for the analysis. For this, mappings from common modeling languages, such as BPEL, BPMN and EPC, exist [28]. Subsequently, the activities — denoted as Petri net transitions — are separated into high and low. The analysis of these models is carried out with *Place-based Non-Interference* (PBNI) [14]. PBNI is an approach to encode and reason about *structural non-interference* (and hence information flow control) in Petri nets. The rationale is that specific places in the net encode different interferences; they denote information leaks. In showing the absence of such places in a net, one rules out structural interferences.

The current approach [3] and tool support [5,19] for information flow analysis of BP models exhibits the following drawbacks: *firstly*, the decision procedures to detect these places are based on the generation and exploitation of the *whole* state space, which renders a very inefficient approach for complex BP models. *Secondly*, formal proofs of the decision procedure are missing. In particular, soundness and completeness have not been demonstrated. The lack of these guarantees weakens the security guarantees provided by tool-support, because it could provide false-negative results, i.e., that a BP model is considered secure even though information leaks exist.

Contributions. This paper provides the following contributions:

– It introduces an approach for the information flow analysis of BP models based on Petri net reachability [31]. By reducing the analysis to reachability, we obtain a decision procedure based on standard Petri net reasoning.
– It shows that the approach based on reachability is sound and complete.
– It presents the design and implementation of Anica, a tool for efficient information flow analysis of BP models. Anica employs Petri net analysis and state space reduction methods to generate and analyze only relevant parts of the state space. Its realization employs the model-checking tool LoLA [37].
– It evaluates Anica on a BP repository comprising over 550 industrial models. The information flow analysis of each BP takes only fractions of a second. Anica analyzes the entire repository, whereas tools using full state space exploration fail for complex BPs.

This paper shows that the well-founded information flow analysis of industrial BPs can be done on-the-fly. This opens the possibility of security analysis *during* BP editing, or at run-time upon the event of ad-hoc process changes. In doing so, the approach contributes to the reliably secure modeling of flexible BP models.

Organization. For completeness, Sect. 2 revisits the concept of PBNI. Section 3 presents the approach to detect non-interference based on reachability, and shows its main properties. Section 4 introduces Anica and Sect. 5 presents its evaluation. Section 6 compares our contribution with related work and Sect. 7 summarizes the lessons learnt and indicates further work.

Running example. We employ a simple example to illustrate the main insights on the information flow analysis. The Petri net in Fig. 1, which serves as input to the information flow analysis approach, represents two instances of a process. The instance classified high is encoded in the upper part; the instance classified

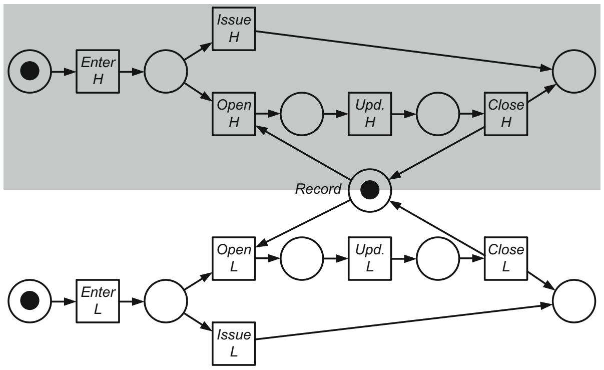


Fig. 1. Petri net model of the Update Patient Record process

low is encoded in the lower part. Each instance realizes the update of a patient record in a hospital. Given an existing patient ID, the corresponding record is opened, updated and closed. The record is represented with the token "Record", which is shared between the two instances. The central security requirement for the process is that information about the patients and the patient record remain secret. This is usually achieved with access control techniques. Below we show that leaks can occur.

2 Information Flow Control in Petri Nets

This section provides the formal basis necessary to reduce the detection of interferences in BPs to a reachability problem. We first define the necessary Petri net background, then revisit Place-based Non-interference (PBNI) [14] to capture leaks as interferences.

Petri Nets. We assume *sound* [1] and *safe* Petri nets; that is, all places contain at most one token in all markings. Safeness is no restriction, because sound nets are bounded and can be expressed as safe Petri nets as well.

Definition 1 (Petri net). *A* Petri net $N = [P, T, F, m_0]$ *consists of two finite and disjoint sets P of* places *and T of* transitions*, a* flow relation $F \subseteq (P \times T) \cup (T \times P)$*, and an* initial marking m_0*. A* marking $m \subseteq P$ *represents a state of the Petri net and is visualized as a distribution of tokens on the places. Let $x \in (P \cup T)$. The* preset *of x is the set ${}^\bullet x = \{y \mid [y, x] \in F\}$, the* postset *of x is $x^\bullet = \{y \mid [x, y] \in F\}$. Transition t is* enabled *in marking m iff, ${}^\bullet t \subseteq m$. An enabled transition t can* fire*, transforming m into the new marking m' with $m' = (m \setminus {}^\bullet t) \cup t^\bullet$. The firing of one transition is denoted as $m \xrightarrow{t} m'$ and a sequence $\sigma \in T^*$ of transitions transforming m to m' is denoted as $m \xrightarrow{\sigma} m'$. A marking m' is* reachable *from m, if $m \xrightarrow{*} m'$ (with $*$ for an arbitrary sequence).*

PBNI and Security Model. PBNI is an approach to reason about structural non-interference in Petri nets. Specifically, PBNI tackles the characterization and

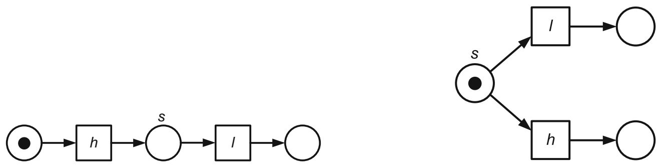


Fig. 2. Patterns for potential causal and conflict places s

detection of places that correspond to a violation of non-interference. Previous work focuses on the characterization of *Bisimulation Non-Deducibility on Composition* (BNDC) [18]. BNDC is a strong structural non-interference property which guarantees that the high and the low parts of the net do not interfere.

The consideration of a security model consisting of two levels does not restrict the generality of the results. According to Anderson [7, Chap. 7], as well as previous work [13,14,15,34], a violation in models using a more complex, linear lattice of security levels (e.g., top secret, secret, moderate, public) can be mapped to information transfer between two levels.

We employ *labeled Petri nets* to encode the two security levels in the Petri nets modeling BPs. To this end, the set of transitions T is divided into two disjoint subsets L and H, capturing, respectively, the low and high levels.

Definition 2 (Labeled Petri Net). *A* labeled Petri net $N = [P, L, H, F, m_0]$ *is a Petri net* $[P, L \cup H, F, m_0]$ *with* $L \cap H = \emptyset$*.*

Given a labeled Petri net, the main insight in PBNI is that some places in a net characterize interferences between a high and a low domain, thereby denoting information flows and violations of confidentiality requirements. Specifically, these places are the *causal places* and *conflict places* (named s in Fig. 2).

Definition 3 (Causal and conflict places [14]). *Let* $N = [P, L, H, F, m_0]$ *be a labeled Petri net. Let* $s \in P$ *be a place of* N *such that* $s^\bullet \cap L \neq \emptyset$*.*

A place s is a potential causal place *if* ${}^\bullet s \cap H \neq \emptyset$*. A potential causal place s is an* active causal place *if the following condition holds: there exists (1)* $l \in s^\bullet \cap L$*, (2)* $h \in {}^\bullet s \cap H$*, and (3) a transition sequence σ and a reachable marking m such that* $m \xrightarrow{h\sigma l} m'$ *and* $s \notin t^\bullet$ *for all* $t \in \sigma$*.*

A place s is a potential conflict place *if* $s^\bullet \cap H \neq \emptyset$*. A potential conflict place is an* active conflict place *if the following condition holds: there exists (1)* $l \in s^\bullet \cap L$*, (2)* $h \in s^\bullet \cap H$*, and (3) a transition sequence σ and a reachable marking m such that* $m \xrightarrow{h} m'$*,* $m \xrightarrow{\sigma l} m''$ *and* $s \notin t^\bullet$ *for all* $t \in \sigma$*.*

A place which is either a potential (active) causal or conflict place is also called a potential (active) place*.*

Causal and conflict places capture the following interferences: in case of a *causal* place s (cf. Fig. 2(a)) transition l always depends on transition h. Then, the fact that transition h has fired leaks to a low user. In case of a *conflict* place s

(cf. Fig. 2(b)), a high and a low transition compete for a token; if transition l cannot obtain the token, a low user may deduce that transition h has been performed (i.e., the occurrence of h leaks). Together, causal and conflict places capture the BNDC property. Its equivalent formalization for Petri nets is called Positive Place-Based Non-Interference (PBNI+) [14]. If causal and conflict places do *not* occur in a net, PBNI+ holds. In other words, causal and conflict places encode all possible leaks. Thus, from now on this paper focuses on PBNI+.

Frau et al. [19] propose a two-step algorithm for the verification of PBNI+. The *static* step analyzes the structure of the net to determine whether *potential* causal or conflict places exist; if so, the *dynamic* step generates and investigates the state space of the net to determine whether they are *active*; that is, on an execution path in the net. The algorithm proposed by Frau et al. is PSPACE-complete. In particular, the *whole* state space must be considered, which makes tool-support based upon these algorithms inefficient. In fact, nets representing complex BPs cannot be analyzed using state space exploration (cf. Sect. 5 for details.)

Running example (cont.). Being employed in a hospital, the BP model in Fig. 1 can be triggered by different users, including nurses (typically low) and physicians (high). The net is designed to analyze the concurrent interaction of subjects in these two levels with the BP to identify possible interferences. The net in Fig. 1 violates PBNI+ and, thereby, the confidentiality requirements. The analysis identifies *Record* as potential causal and potential conflict place. The dynamic analysis further indicates that *Record* is active in both cases.

Technically, *Record* induces a storage covert channel [34] by which users in the low domain may deduce information about high. This happens because users share the same storage resource. Specifically, the hospital information system encompassing this BP allows, for instance, the deduction which patients were hospitalized (e.g., the low part observes which records are being processed) and the kind of treatment for a patient (the low part observes the timespan necessary to update a record, associating the elapsed time to a particular treatment).

3 Verification of PBNI+ as a Reachability Problem

In this section we show how PBNI+ verification is decided using reachability [31]. To this end, we introduce *objectives* which we use to encode potential interferences. To decide whether such a potential interference, encoded as objective, is an active one, we create an *extended net* based on the objective. Subsequently, we perform a reachability check on each extended net. In case a dedicated place in the extended net can be marked, the potential interference is an active one. The provided proofs guarantee that our translation to reachability is correct (soundness) and that we capture all interferences (completeness).

3.1 Creating Reachability Problems

The central concepts in our new definitions are the objective and the *undesired transitions*. An objective is a triple $[s, h, l]$ consisting of a place s, a transition h

labeled high and a transition l labeled low. With these three nodes we can describe each potential and active place (cf. Def. 3). The difference between a potential and an active place, expressed as objective, lies in the dynamic behavior of their labeled Petri net. This difference is based on a transition sequence σ in which some transitions are prohibited for each particular objective $[s, h, l]$, these transitions are therefore undesired.

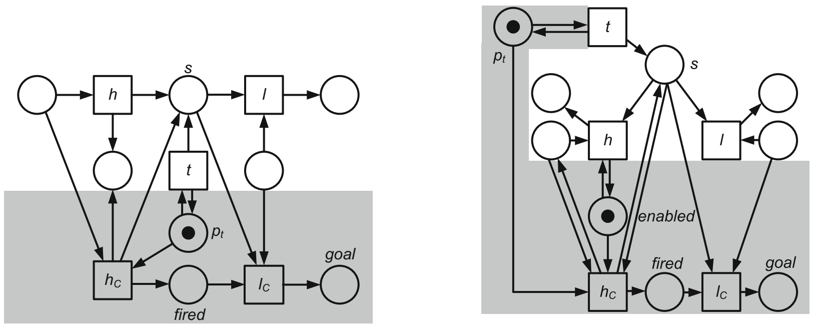
Definition 4 (Objective, undesired transitions). *An* objective $[s, h, l]$ *is a triple for both a potential causal or a potential conflict place with* $s \in P$, $l \in s^\bullet \cap L$ *and* $h \in {}^\bullet s \cap H$ *(in causal case) or* $h \in s^\bullet \cap H$ *(in conflict case). We call* $[s, h, l]$ *an* active objective *(or* potential objective*) if* s *is an active (or potential) causal or conflict place for the transitions* h *and* l*. Let* $U_{[s,h,l]} = \{t \in T \mid s \in t^\bullet\}$ *be the* undesired transitions *in the transition sequence* σ *(cf. Def. 3), necessary to decide whether* s *is active.*

For each potential place s, there exists at least one objective $[s, h, l]$, which we use to construct a Petri net $N_{[s,h,l]}$. The next definition shows how we construct this extended net based on the given net N and the objective $[s, h, l]$. The first four changes apply to both causal and conflict places.

Definition 5 (Extended net). *Let* $N = [P, L, H, F, m_0]$ *be a labeled Petri net and* $[s, h, l]$ *be an objective of* N*. The* extended net $N_{[s,h,l]} = [P', T, F', m'_0]$ *is a Petri net, which is constructed based on* N *as follows:*

1. $P_T := \left\{p_t \mid t \in U_{[s,h,l]}\right\}$,
2. $P' := P \cup \{\mathit{fired}, \mathit{goal}\} \cup P_T$,
3. $T := L \cup H \cup \{h_C, l_C\}$,
4. $F' := F \cup \{[p, h_C] \mid p \in {}^\bullet h\} \cup \{[p, l_C] \mid p \in {}^\bullet l\} \cup \{[h_C, \mathit{fired}], [\mathit{fired}, l_C], [l_C, \mathit{goal}]\} \cup \left\{[p_t, t], [t, p_t] \mid t \in U_{[s,h,l]}, p \in P_T\right\} \cup \{[p_t, h_C] \mid p_t \in P_T\}$,
5. $m'_0 := m_0 \cup P_T$,
6. $F' := F' \cup \{[h_C, p] \mid p \in h^\bullet\}$ *(only causal case),*
7. $P' := P' \cup \{\mathit{enabled}\}$ *(only conflict case),*
8. $F' := F' \cup \{[h_C, p] \mid p \in {}^\bullet h\} \cup \{[\mathit{enabled}, h_C], [h, \mathit{enabled}], [\mathit{enabled}, h]\}$ *(only conflict case),*
9. $m'_0 := m'_0 \cup \{\mathit{enabled}\}$ *(only conflict case).*

According to Def. 5, extended nets are Petri nets (cf. Def. 1), because the labeling of the original transitions is not considered, so we disregard the labels in $N_{[s,h,l]}$. Extended nets are safe by construction. Regarding Def. 3, we need to disable all undesired transitions, therefore we create, for each undesired transition $t \in U_{[s,h,l]}$, an additional place p_t (1). Self loops between the undesired transitions $t \in U_{[s,h,l]}$ and their new places $p_t \in P_T$ (4), together with their initial marking (5) provide the desired behavior. The places of the extended net (2) consist of those from (1) and the places *fired* (indicating whether h_C has fired) and *goal* (marked, if the objective $[s, h, l]$ is active). The new transitions (3) of the extended net are h_C and l_C. Think of them as copies of h and l. To make use of the additional nodes in the extended Petri net, we add arcs (4) from the preset of h to h_C, from the preset of l to l_C and the flow between *fired*, *goal*, h_C and l_C.



(a) Causal case. (b) Conflict case.

Fig. 3. Extension patterns for causal and conflict case

In the *causal* case, we need the effect of firing h_C (instead of h), so we add the postset of h to the postset of h_C (6). In the *conflict* case, we need an extra place *enabled* (7); this place ensures that after firing h_C (instead of h) h cannot fire anymore, even though all preconditions for h remain. This is necessary because, in the conflict case, h and l compete for the token on s. In contrast to the causal case, we do not need the effect of firing h or h_C. Instead, we must notice that h was activated. We thus add the preset of h to the postset of h_C and use *enabled* to ensure that this happens only once (8). The initial marking must hence contain an additional token on *enabled* (9).

In Fig. 3 the extension patterns for both cases are shown exemplarily. The extended parts of $N_{[s,h,l]}$ (over N) are highlighted. For simplicity, only the neighborhoods of the interesting nodes s, h, and l are shown with just one element.

Considering declassification. In practice, information flow control is often too strict because every interference is considered as bad. Think of a login check where one bit distinguishes whether the login is successful. Here, an authorized flow from high to low happens, even though it violates non-interference. *Declassification* is used to controllably downgrade flows from high to low. Busi and Gorrieri extended PBNI+ to intransitive non-interference with downgrading transitions, so-called PBNID [20]. For this, Def. 3 is modified: it adds a set D of downgrading transitions and discards any $d \in D$ from transition sequence σ. Hence, in addition to Def. 3, the firing sequence σ is restricted to $\sigma \in (H \cup L)^*$. Reducing PBNID to reachability is easily achieved by adding all downgrading transitions $d \in D$ in $N_{[s,h,l]}$ to the undesired transitions $U_{[s,h,l]}$.

3.2 Completeness and Soundness Proofs

The main property to be proven is stated in Theorem 1: a leak encoded in an objective is active if and only if it is comprised in a reachable marking.

Theorem 1. *The objective* $[s, h, l]$ *is active in* N *if and only if a marking* m *with* $goal \in m$ *is reachable in* $N_{[s,h,l]}$.

To show Theorem 1, it suffices to show completeness and soundness. We state each as lemma, proving them separately. Due to the lack of space, the proofs concentrate on the causal case. The conflict case is similar.

Lemma 1 (Completeness). *The objective $[s, h, l]$ is active in N if a marking m with goal $\in m$ is reachable in $N_{[s,h,l]}$*

For completeness, we need an additional proposition.

Proposition 1 (Same firing sequences). *Let N be a labeled Petri net, $[s, h, l]$ a potential objective of N and $N_{[s,h,l]}$ be the extended net for N and $[s, h, l]$. Then $N_{[s,h,l]}$ contains all firing sequences of N.*

Proof (Proposition 1). Observations: No transition and no arc is removed but additional places are added in some presets. Thus, we only consider these new places in this proof.

In both cases, for each transition $t \in U_{[s,h,l]}$, an additional place p_t is added to its preset. In the initial marking m'_0, these places are marked, hence their enabling is not restricted. Every time a transition $t \in U_{[s,h,l]}$ fires, it produces the token on p_t again. The only transition which consumes all tokens from P_T is h_C, which is a new transition.

Consequently, all firing sequences of N, which do not contain any new transitions, are also possible firing sequences in $N_{[s,h,l]}$. □

With Proposition 1, we can prove Lemma 1 for completeness.

Proof (Lemma 1). Assume $[s, h, l]$ is active in N. Then a marking m^* with *goal* $\in m^*$ must be reachable in $N_{[s,h,l]}$. N can fire $m_0 \xrightarrow{\sigma_0} m \xrightarrow{h\sigma} (s \cup m') \xrightarrow{l} m''$, so we have to show that $N_{[s,h,l]}$ can fire $m'_0 = (m_0 \cup P_T) \xrightarrow{*} m^*$, where m'_0 is the initial marking of $N_{[s,h,l]}$.

Based on Proposition 1, the extended net $N_{[s,h,l]}$ can fire the same firing sequences as N, especially σ_0 from m_0 to m. After this firing sequence, the marking of $N_{[s,h,l]}$ is $m \cup P_T$. In this marking, $N_{[s,h,l]}$ can fire h_C instead of h, which has the same effect to the original part of the net as firing h. Because $[s, h, l]$ is active, σ contains no transitions of $U_{[s,h,l]}$. Therefore disabling them by h_C does not disable any transition of σ. So instead of $(m \cup P_T) \xrightarrow{h\sigma} (s \cup m' \cup P_T)$ (by firing h), we obtain $(m \cup P_T) \xrightarrow{h_C\sigma} (s \cup m' \cup \mathit{fired})$ (by firing h_C). In both cases, the net $N_{[s,h,l]}$ can either fire l or l_C, in case of l_C the place *goal* is marked. □

Until here we proved that every active objective $[s, h, l]$ leads to a reachable marking m in which *goal* is marked in the extended net $N_{[s,h,l]}$. Next we show that in all cases *goal* is reachable in $N_{[s,h,l]}$ the objective $[s, h, l]$ is active in N.

Lemma 2 (Soundness). *The objective $[s, h, l]$ is active in N only if a marking m with goal $\in m$ is reachable in $N_{[s,h,l]}$.*

The proof of soundness is analogous to the proof of completeness, except for the fact that we use an indirect approach.

Proof (Lemma 2). Assume that a marking m^* with $goal \in m^*$ is reachable in $N_{[s,h,l]}$ and $[s,h,l]$ is not active in N.

Place *goal* can only be marked by firing l_C. If l_C was able to fire, l was also able to fire $({}^\bullet l \subsetneq {}^\bullet l_C)$. Before l_C can fire, firing h_C is necessary. If h_C was able to fire, h was also able to fire $({}^\bullet h \subsetneq {}^\bullet h_C)$. A possible firing sequence between h_C and l_C cannot contain transitions from $U_{[s,h,l]}$, because h_C consumes all tokens on all places $p \in P_T$. This means σ contains no undesired transitions. Consequently $h\sigma l$ is a valid firing sequence in N starting in some marking reachable from m_0 in contradiction to the assumption. □

The proof of the conflict case is similar. The only difference is place *enabled* which is comparable to the places in P_T. As in the conflict case, transitions h and l compete for the token in place s, place *enabled* is used to indicate that h was enabled but not fired, and by choosing h_C instead, h is deactivated by consuming the token on *enabled*.

By proving completeness and soundness we ensure that our approach detects all possible violations (completeness) and that all violations detected by our approach are indeed violations (soundness).

4 Anica – Tool Support for Detecting Information Flows

Anica (Automated Non-Interference Check Assistant) implements Theorem 1. It checks PBNI+ (and PBNID) much faster and with less space than existing tools (cf. Sect. 5 for details) by creating reachability problems automatically. As part of the open source service-technology.org-family [29], Anica uses the model checker LoLA [37] and the Petri Net API [27].

Figure 4 depicts the current architecture. Anica first reads a labeled Petri net as input, then creates pairs of extended nets and task files. A task file contains the reachability statement for place *goal*. These pairs are passed to LoLA. If the checked objective is not active, more pairs are created, otherwise the result is presented to the user.

The implementation of Anica follows the constructive Def. 3, 4 and 5. In this section, we focus on how to combine the definitions to a tool. The input of Anica

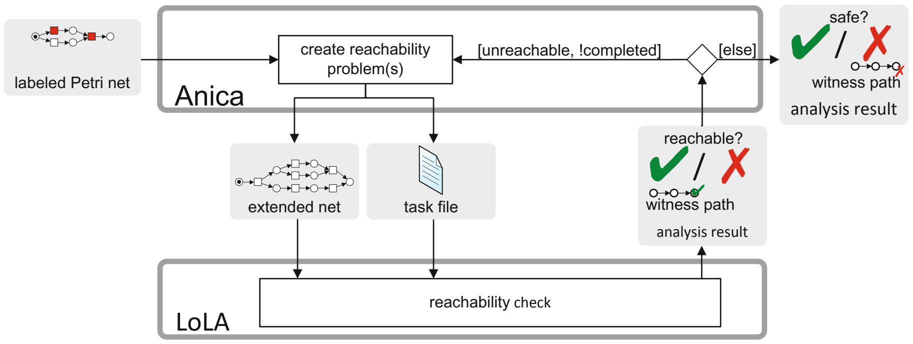


Fig. 4. Architecture of Anica

is a labeled Petri net. The transitions of such a net are labeled with high ($h \in H$) or low ($l \in L$) and for PBNID possibly with downgrade ($d \in D$). A net violates the non-interference properties PBNI+ or PBNID if it contains at least one active place. So we check for each place whether it is active (cf. Alg. 1). In case we are only interested in whether the whole net is secure we can abort after the first active place (Alg. 1, line 3). For a complete check of the whole net we enumerate all places, which can be done simultaneously.

A necessary condition for an active place is that it must be a potential (causal or conflict) place (cf. Def. 3). Checking whether a place is potential can be decided by the structure of the input net, which has linear complexity in terms of nodes. If the input net contains no potential place, it is secure. If a place s is a potential place, we generate all its objectives (cf. Alg. 2). For causal and conflict places we need the transitions labeled low ($l \in L$) in the postset of s. Which transitions labeled high ($h \in H$) are necessary depends on the case: we use the preset of s for potential causal places and the postset of s for potential conflict places. Whereas one active objective $[s, h, l]$ (tested by Alg. 3) is enough to make a potential place s active, we break on the first active objctive (Alg. 2, lines 4 and 7). All checks of the objectives are independent (Alg. 2, lines 2-4 and 5-7).

Algorithm 1. isNoninterfering(N): *Boolean*

```
1: for all $s \in S$ do
2:    if isActive($N, s$) then
3:       return false                              // PBNI+ or PBNID is violated
4: return true
```

Algorithm 2. isActive(N, s): *Boolean*

```
1: for all $l \in (L \cap s^\bullet)$ do
2:    for all $h \in (H \cap {}^\bullet s)$ do
3:       if isActiveObjective($N, [s, h, l]$) then      // $s$ is a potential conflict place
4:          return true                                  // $s$ is an active causal place
5:    for all $h \in (H \cap s^\bullet)$ do
6:       if isActiveObjective($N, [s, h, l]$) then      // $s$ is a potential conflict place
7:          return true                                  // $s$ is an active conflict place
8: return false                                          // $s$ is not active
```

Algorithm 3. isActiveObjective(N, $[s,h,l]$): *Boolean*

```
1: construct extended net $N_{[s,h,l]}$                 // according to Def. 5
2: if $m_0 \xrightarrow{*} m$ in $N_{[s,h,l]}$ with $goal \in m$ then      // call LoLA
3:    return true
4: return false
```

5 Evaluation

The evaluation will demonstrate that information flow analysis becomes feasible with our approach in contrast to the existing ones. We use a library of 735 industrial BPs from different business branches, including financial services, ERP, supply-chain, and online sales. These BPs were modeled in the IBM WebSphere Business Modeler format. Only 559 models out of the 735 are sound [1], thus building the core set of BPs for the evaluation. Fahland et al. [17] translated this set of BPs to role annotated Petri nets.

The remaining 559 sound BPs contain no semantic information with respect to the security domains. That is, they are not labeled for security analysis. We use the roles of the BPs (called swim lanes) for labeling. For each role r (swim lane) in a BP B, we create one labeled Petri net N_{B_r}. In N_{B_r}, all transitions labeled with r are set secret (labeled high). This means we check PBNI+. Figure 5 gives an overview on this approach. Intuitively, with this strategy we can check the interference between the different roles within a process, (e.g., interference between the back- and front office of a financial service). We cannot judge the resulting interferences, because they are constructed by us, instead of derived from practice. But we believe this approach is comprehensible, acceptable, and more realistic than creating randomized BPs and security domains. This yields 1,206 labeled Petri nets which build the base for the evaluation. Table 1 summarizes their structural and behavioral properties.

Intentionally and for the sake of exhaustiveness, all the 1,206 processes violate the non-interference property: in each process, there exist at least one, at most 25 and in average 3 potential causal places. All of them are also active causal places and none of the 52,751 examined places is a potential conflict place. The absence of conflict places can be explained by the construction of the security

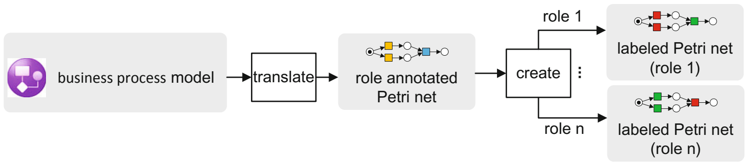


Fig. 5. Testsuite generation

Table 1. Structural and behavioral properties of the 1,206 labeled Petri nets

	Min	Average	Max
Places	5	44	234
Transitions (labeled high)	4 (2)	28 (5)	100 (60)
Arcs	8	93	476
Full state space (< 10,000,000 states)	5	1,777	588,508
Full state space (> 10,000,000 states)	28,451,334	> 1,074,797,069	> 2,214,007,203

Table 2. Comparing existing techniques (the state space) with Anica (reachability)

	Existing techniques	**Anica**
Labeled Petri nets verified	1,178	1,206
Minimum states computed	5	5
Average states computed	1,777	51
Maximum states computed	588,508	735
Overall states computed	2,088,135	62,049

domains: Labeling along the swimlanes results in a clear responsibility between two roles. After a role has finished a specific task, it provides its results to exactly one successor role. The successor role is elected by the current role, i.e., the choice is in the initial swimlane. Overall, 8 percent of all places violate the non-interference property.

State space and runtime. Using existing techniques to analyze PBNI+ or PBNID, the whole state space must be created. As 28 of the 1,206 labeled Petri nets (see Tab. 2) contain more than 10 million states (see Tab. 1), those are infeasible using common desktop hardware (8 GB RAM). The existing state space approaches require about 2 million states to verify the remaining 1,178 nets and fail for the 28 big nets. For those, we used a server with 128 GB RAM, computed more than 30 billion states in several days and were able to verify 15 of 28 nets.

Our reachability approach instead verifies *all* 1,206 labeled Petri nets with about 0.06 million states in about 30 seconds on common desktop hardware. The proofs provided in Sect. 3.2 ensure the correctness of our approach. We performed all tests in a sequential manner (i.e., used no parallelism) and the 30 seconds include the time consumed by LoLA for deciding the reachability problems. This leads to an average time consumption of about 24 ms for each net (212 ms for the biggest net), without exploiting the inherent parallelism.

Running example (cont.). Anica verifies the net in Fig. 1 by checking 12 states, instead of 32 states for the full state space. *Record* is marked as an active causal and active conflict place. The objective for the causal case is [*Record*, *Close H*, *Open L*] and the conflict case is [*Record*, *Open H*, *Open L*].

The testsuite shows that, by reducing PBNI+ to reachability, (1) we can handle more complex inputs from industry and (2) we can check them independently. Anica does not depend on LoLA. All results can be achieved with any other model checker capable of state-of-the-art reduction techniques for reachability.

6 Related Work

Business process security focuses on the enforcement of confidentiality, integrity and availability requirements in BP models [10]. The focus of information flow analysis is confidentiality and, dually, integrity [34]. While approaches employing formalisms other than Petri nets exist (e.g., [9,11,21,26]), the vast majority of proposed technologies build upon Petri nets. This is due to the extensive use of Petri net models to reason about BPs and the availability of mature tool support. The state of the art addresses: (1) the detection of *explicit* information flows (i.e., data flows), both in terms of discretionary access control (DAC) and mandatory access control (MAC) based upon multi-level security; and (2) the detection of *implicit* flows over covert channels (i.e., interferences). Today's focus is on (1), whereas the tool Anica and the formal results presented here focus on (2).

Tackling DAC, Shafiq et al. [35] present a colored Petri net (CPN) framework for verifying the consistency of role based access control (RBAC) policies. Similarly, Armando and Ranise [8] provide a SAT-based tool support for analyzing BPs with RBAC. The Chinese Wall policy, together with the Strict Integrity Policy, is considered by Zhang et al. [38]. Here, policies are modeled with CPN, whereas the verification is based upon a coverability graph technique. Huang and Kirchner [22] use CPN to model policies in general, and conflict of interest in particular, thereby focusing on the modularity aspects of Petri net-based policies. Katt et al. [25] and ourselves [2] employ CPN to model usage control policies. However, focusing on monitoring systems, the objective is not the analysis. Focusing on DAC, these analysis technologies merely cover "point-to-point" security guarantees, which make it impossible to detect leaks originating from interferences.

Turning to MAC-based data-flow analysis, existing approaches are based upon the multi-level security (MLS) model, in particular the model of Denning [15] and Bell and LaPadula [13]. Juszczyszyn [24] uses CPN for the MLS specification of policies and CPN-Tool to realize the verification as a reachability problem. Röhrig and Knorr [33] define task based access control as a dynamic BP and then specify the BPs with Petri nets; the approach is unclear as to the realization of the analysis, though. Barkaoui et al. [12] employ the security rules of Bell and LaPadula for the verification of workflows expressed as Workflow nets. The reasoning reduces to a soundness check with Maude. While these technologies employ MLS to capture information flows, they do not detect interferences.

The information flow analysis of BPs models focusing on interferences is a young research strand in the business process management community. The approach is inspired by the formalization of non-interference properties as Petri net patterns given by Busi and Gorrieri [14]. On these grounds, we [3,4] present with InDico information flow nets to capture BP transformations to automatically label the model with security classes, and Petri net patterns capturing non-interference. We also consider other properties, including data flow-based properties and industrial constraints, such as separation of duties and declassification. This approach has been successfully employed to detect leaks in BP models; for example in e-auction [3], e-health [4,32] and publishing [6]. However soundness and completeness guarantees are not provided. The approach presented in this paper is thus the first to use reachability to address the verification problem and to provide soundness and completeness properties.

Turning to tools for automatic information flow analysis of BP models, Frau et al. [19] implemented the general-purpose Petri Net Security Checker (PNSC) and we implemented InDico in the Security Workflow Analysis Toolkit (SWAT) [5]. Both realizations follow the state space exploration strategy proposed in [14]. They thus require the construction and exploration of the whole state space, so that the tools do not scale well for big Petri net models. The tool Anica presented in this paper provides a faster alternative.

7 Summary and Future Work

In this paper we introduced a novel approach for the automated verification of information flow control for BPs. The main insight in the paper is the treatment of PBNI+ verification as a set of reachability problems, which opens up the possibility of employing efficient, automated tool-support for the analysis. The approach is proven to be sound and complete, and is implemented in the Anica tool. A thorough evaluation demonstrates the speed to verify complex BPs.

Lessons learnt. Although the reachability problem itself is also PSPACE-complete for safe nets and although the approach introduced in this paper comprises more of these checks for one BP than the state space approach, Anica is a much faster mechanism in the average case, as shown in Sect. 5. Hence, information flow security checks for industrial, complex BPs become efficient. This opens up the possibility of information flow analysis in background during the process modeling, thereby contributing to reliably secure BP design. Such a guarantee was not possible with previous techniques.

Future work. Future work aims at three directions: *firstly*, augment the set of properties to be analyzed. In particular, we plan to integrate the properties considered in InDico [4]; for instance parameter confidentiality and resource conflicts. We firmly believe that further properties considerably extend the expressive power of the certification, thereby contributing to reliably secure BP modeling from the outset. *Secondly*, spelling out the consequences for more complex security models beyond the high and low classes. This allows one to capture fine-grained properties which are especially useful for data-flow reasoning. *Thirdly*, provide some further information on found violations. So far the modeler can identify the cause during the modeling (when using background checks) or use provided witness paths. Nevertheless the question is raised whether it is possible to rate found violations automatically. Based on these ideas, it seems possible to help the modeler to fix the BP, therefore he could use for instance downgrading transitions (PBNID) which are already supported by Anica.

Acknowledgments. The authors cordially thank Dirk Fahland, Niels Lohmann, and Harro Wimmel for their valuable feedback and the PC members for their constructive reviews. The work is partially funded by the German Research Foundation in the project WS4Dsec of the priority program Reliably Secure Software Systems (SPP 1496).

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
2. Accorsi, R., Lowis, L., Sato, Y.: Automated certification for compliant cloud-based business processes. Bus. & Information Systems Eng. 3(3), 145–154 (2011)
3. Accorsi, R., Wonnemann, C.: Strong non-leak guarantees for workflow models. In: ACM Symposium on Applied Computing, pp. 308–314. ACM (2011)
4. Accorsi, R., Wonnemann, C.: InDico: Information Flow Analysis of Business Processes for Confidentiality Requirements. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) STM 2010. LNCS, vol. 6710, pp. 194–209. Springer, Heidelberg (2011)
5. Accorsi, R., Wonnemann, C., Dochow, S.: SWAT: A security workflow toolkit for reliably secure process-aware information systems. In: Conference on Availability, Reliability and Security, pp. 692–697. IEEE (2011)
6. Accorsi, R., Wonnemann, C., Stocker, T.: Towards forensic data flow analysis of business process logs. In: Incident Management and Forensics, pp. 94–110. IEEE (2011)
7. Anderson, R.: Security engineering. Wiley (2008)
8. Armando, A., Ranise, S.: Automated Analysis of Infinite State Workflows with Access Control Policies. In: Meadows, C., Fernandez-Gago, C. (eds.) STM 2011. LNCS, vol. 7170, pp. 157–174. Springer, Heidelberg (2012)
9. Atluri, V., Chun, S.A., Mazzoleni, P.: A Chinese Wall security model for decentralized workflow systems. In: ACM Computer & Communication Security, pp. 48–57. ACM (2001)
10. Atluri, V., Warner, J.: Security for workflow systems. In: Handbook of Database Security, pp. 213–230. Springer (2008)
11. Attali, I., Caromel, D., Henrio, L., Aguila, F.: Secured information flow for asynchronous sequential processes. Electr. Notes Theor. Comput. Sci. 180(1), 17–34 (2007)
12. Barkaoui, K., Ayed, R.B., Boucheneb, H., Hicheur, A.: Verification of workflow processes under multilevel security considerations. In: Risks and Security of Internet and Systems, pp. 77–84. IEEE (2008)
13. Bell, D., LaPadula, L.: Secure Computer Systems: Mathematical Foundations. MITRE Corporation (1973)
14. Busi, N., Gorrieri, R.: Structural non-interference in elementary and trace nets. Mathematical Structures in Computer Science 19(6), 1065–1090 (2009)
15. Denning, D.E.: A lattice model of secure information flow. Communications of the ACM 19(5), 236–243 (1976)
16. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Communications of the ACM 20(7), 504–513 (1977)
17. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl. Eng. 70(5), 448–466 (2011)
18. Focardi, R., Gorrieri, R.: Classification of Security Properties. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 331–396. Springer, Heidelberg (2001)
19. Frau, S., Gorrieri, R., Ferigato, C.: Petri Net Security Checker: Structural Noninterference at Work. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 210–225. Springer, Heidelberg (2009)

20. Gorrieri, R., Vernali, M.: On Intransitive Non-interference in Some Models of Concurrency. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2011. LNCS, vol. 6858, pp. 125–151. Springer, Heidelberg (2011)
21. Harris, W., Kidd, N., Chaki, S., Jha, S., Reps, T.W.: Verifying Information Flow Control over Unbounded Processes. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 773–789. Springer, Heidelberg (2009)
22. Huang, H., Kirchner, H.: Formal specification and verification of modular security policy based on colored Petri nets. IEEE Trans. Dependable Sec. Comput. 8(6), 852–865 (2011)
23. ISO/IEC Information Security Management System 27001 (2005), `http://www.27000.org/iso-27001.html` (last accessed in June 2012)
24. Juszczyszyn, K.: Verifying enterprise's mandatory access control policies with coloured Petri nets. In: Enabling Technologies, pp. 184–189. IEEE (2003)
25. Katt, B., Zhang, X., Hafner, M.: Towards a Usage Control Policy Specification with Petri Nets. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009, Part II. LNCS, vol. 5871, pp. 905–912. Springer, Heidelberg (2009)
26. Kovács, M., Seidl, H.: Runtime Enforcement of Information Flow Security in Tree Manipulating Processes. In: Barthe, G., Livshits, B., Scandariato, R. (eds.) ESSoS 2012. LNCS, vol. 7159, pp. 46–59. Springer, Heidelberg (2012)
27. Lohmann, N., Mennicke, S., Sura, C.: The Petri Net API: A collection of Petri net-related functions. In: Algorithms and Tools for Petri Nets. CEUR Workshop Proc., vol. 643, pp. 148–155. CEUR-WS.org (2010)
28. Lohmann, N., Verbeek, E., Dijkman, R.: Petri Net Transformations for Business Processes – A Survey. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 46–63. Springer, Heidelberg (2009)
29. Lohmann, N., Wolf, K.: How to Implement a Theory of Correctness in the Area of Business Processes and Services. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 61–77. Springer, Heidelberg (2010)
30. Lowis, L., Accorsi, R.: Vulnerability analysis in SOA-based business processes. IEEE T. Services Computing 4(3), 230–242 (2011)
31. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE 77(4), 541–580 (1989)
32. Pfeiffer, S., Unger, S., Timmermann, D., Lehmann, A.: Secure Information Flow Awareness for Smart Wireless eHealth Systems. In: Multi-Conference on Systems, Signals and Devices. IEEE (2012)
33. Röhrig, S., Knorr, K.: Security analysis of electronic business processes. Electronic Commerce Research 4(1-2), 59–81 (2004)
34. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
35. Shafiq, B., Masood, A., Joshi, J., Ghafoor, A.: A role-based access control policy verification framework for real-time systems. In: Object-Oriented Real-Time Dependable Systems, pp. 13–20. IEEE (2005)
36. Trusted Computer Security Evaluation Criteria, DoD (1983), `http://csrc.nist.gov/publications/history/dod85.pdf` (last accessed in June 2012)
37. Wolf, K.: Generating Petri Net State Spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)
38. Zhang, Z.-L., Hong, F., Xiao, H.-J.: Verification of strict integrity policy via Petri nets. In: Conference on Systems and Networks Communications, p. 23 (2006)

Managing and Tracing the Traversal of Process Clouds with Templates, Agendas and Artifacts

Marian Benner, Matthias Book, Tobias Brückmann,
Volker Gruhn, Thomas Richter, and Sema Seyhan

Paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen, Gerlingstr. 16, 45127 Essen, Germany
`{marian.benner,matthias.book,tobias.brueckmann,
volker.gruhn,thomas.richter,sema.seyhan}@paluno.uni-due.de`

Abstract. In many domains, we find tasks for which no strict process can be prescribed, but which require the expertise of case managers who work with information from a broad set of sources. To support case managers' highly individual work in such so-called "process clouds", we present an approach that enables them to structure their work along any suitable mix of keywords, activities and artifacts.

Keywords: case management, agenda items, templates, artifacts.

1 Introduction

Business process management and workflow management ensure that process instances follow the definitions and guidelines described in the form of process models. This works nicely for all kinds of processes that are well-understood and highly structured. Such business processes tend towards automation (as a consequence of ongoing industrialization) [1]. Usually, however, certain parts of business processes cannot be automated; very often, they cannot even be described in detail. They are less structured and depend much more on the expertise of a responsible person (called a case manager in the following). Very often, such a case manager does not follow a predefined process model, but a more or less coarse agenda that he arranges flexibly as the case demands. Here we leave the traditional field of business process management and arrive at so called case handling [2]. When working on an "open process cloud", as we call it, we observe that case managers employ other cognitive approaches for planning and structuring their work than they would need to "blindly" follow predefined processes: To build their agenda for proceeding with a knowledge/research-intensive task, some will follow an ontological approach, planning their work along the conceptual entities of the business domain that the business process touches on (in the health care domain, for example, diagnosis, treatment, complications etc.). Others break a task down by the activities that need to be performed in the course of its completion (e.g. requesting second opinions, making appointments, checking progress etc.). Still others' agendas may focus on the artifacts they need to review and produce (e.g. doctors' reports, treatment schedules, evaluations etc.). Often, a case manager's agenda will contain items of all three classes – how they are weighed, and how many items there are, largely remains at the discretion of each individual case manager.

Besides flexible agenda management and support for artifact-oriented working, another main goal of our research is to enable further automation of unstructured processes. To learn what actually happens in a process cloud, we therefore propose to log all its relevant events in so-called traces. Trace evaluation should then give us a basis for better understanding processes and reducing their "cloudiness".

In summary, this paper introduces the notion of (i) *process clouds* that indicate parts of business processes which are not well-understood; (ii) flexible *agendas* that let case managers organize their process work in different styles; and (iii) a *workspace* that supports artifact-oriented knowledge work. In the following section, we first introduce to the core concepts of process clouds. Section 3 then suggests approaches for process cloud optimization through tracing, and describes how an OPC toolset can support case managers. We conclude with a discussion of related work in Section 4.

2 Process Cloud Concepts

Even though process clouds do not have a clearly defined internal structure (and therefore cannot be supported by traditional workflow management systems), we believe that cloud traversals can be tool-supported. Our mechanisms focus on supporting different working styles of case managers in their cloud traversals, where they shall be free to plan and structure their work according to the cognitive approach that seems most useful to them. To accomplish this, we will introduce the concepts of agendas, templates and artifacts using an example from the rehabilitation management domain in the context of disability insurance. Here, so-called rehabilitation managers are responsible for driving the medication and therapy process in order to avoid unnecessary costs and to ensure quick recovery of the injured person.

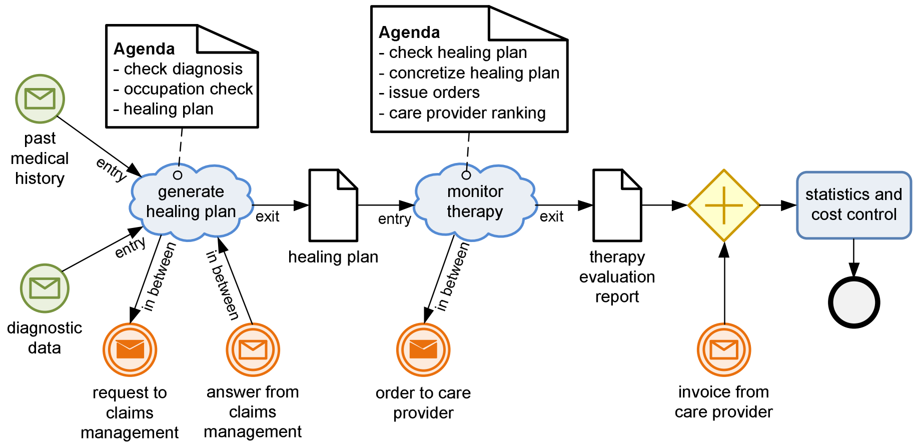


Fig. 1. Structured process parts and process clouds in a rehabilitation control process

While some of the activities performed by a rehabilitation manager can be explicity defined and supported by structured dialogs and algorithms, the details of certain other parts (such as selecting the most appropriate care provider) are rather undefined and symbolized by clouds in Fig. 1: For these process clouds, it is not specified in detail what has to be done at which step, what kinds of information sources have to be considered, and what kinds of data have to be produced – we just know that the process cloud requires and produces certain input and output artifacts, and may have a coarse agenda as an informal guideline for what needs to be done to complete the task. In the following subsections, we show how the OPC approach can support case managers in working on such unstructured, knowledge-intensive tasks:

Agendas. In any open process cloud, the notion of an *agenda* is of central importance. An agenda contains links to and names of all entities that a case manager puts on the list of things he plans to consider. If starting from an activity-driven perspective, first-class agenda items are activities that are candidates for execution. If the perspective is more artifact-oriented, then first-class agenda items are any form of artifacts (files, documents, web pages etc.) and other sources of information. If the perspective is more structure-driven, first-class agenda items are key entities of the cloud's domain. In the given scenario, for example, "healing plan", "care providers" and "therapy" are such key domain entities. This structure-driven perspective allows allocating activities, information sources and documents around key entities of the case domain.

Our notion of an agenda supports these arbitrary mixes of perspectives by just using the general notion of agenda items, and by supporting hierarchies of agenda items in order to support different levels of abstraction. Agenda item hierarchies can be arbitrarily linked with each other. If, for example, an activity such as "generate healing plan" is linked to the domain entity "healing plan", then this may be interpreted to be an activity which works on a healing plan.

Templates. For certain process cloud traversals, the case manager might start on a clean slate, having to figure out for himself how best to proceed and what agenda to follow. In other process clouds, there may be precedents that provide guidelines for subsequent cloud traversals. In our example, it might be useful to identify what is usually done in the "monitor therapy" cloud, and to check what other rehabilitation managers recently did in the context of similar injuries. They might take note of what kind of expertise has to be considered in the further cloud traversal and which information sources should be consulted.

Agenda items which are considered useful for all process clouds can be collected in what we call a *template*. Such an agenda template includes an initial set of agenda items and a set of cloud-, but not yet case-specific, artifacts that any new process cloud is initially populated with. The template itself might be defined according to an organization's documented best practice, the case manager's own experience, or accumulated histories of prior cloud traversals by colleagues. A case manager can modify the initially provided agenda by deleting, adding and revising items. When the traversal is completed, the case manager and template owner can decide whether the template should be updated. As an alternative to manual revision of the agenda template, cloud tracing mechanisms may enable automated optimization (cf. Sect. 3).

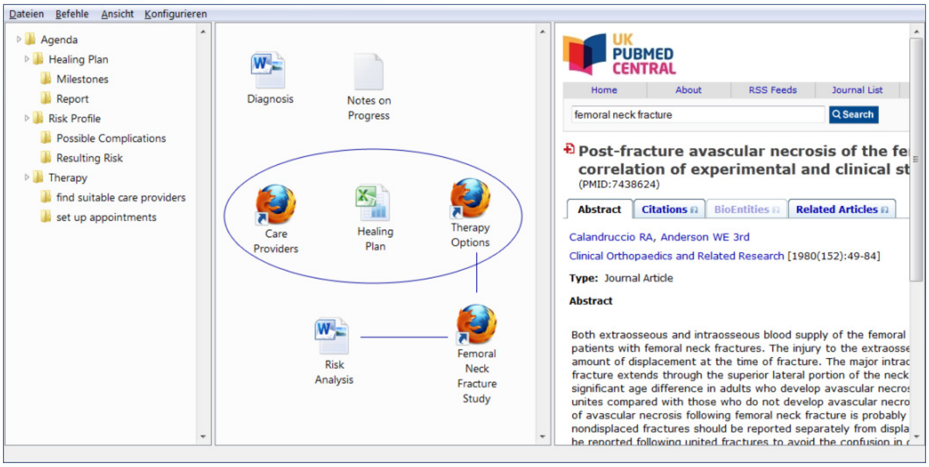
Artifacts. While the activity-driven approach to traversing clouds is based on the agenda and agenda items, the artifact-driven approach is based on artifacts and workspaces. The motivation for this perspective on process clouds is that one of the most frequently performed activities of a case manager is to search and explore information related to the case. Based on the information gathered from various heterogeneous sources, he builds a mental model of the way he wants to solve the case. We collectively call all information that the case manager gathers *artifacts*, and are developing a digital workspace (briefly introduced below) where the case manager can combine, evaluate, rate, relate and annotate all relevant information.

3 Ongoing Research

In parallel to refining the above elements of process clouds, we are developing mechanisms for tracing and analyzing case managers' actions, for several purposes:

- Individual cloud traces can document what has actually happened in individual cloud traversals. This may be required by law for certain processes (e.g. documenting rationales for therapy decisions), or desirable in order to check compliance with company policies (e.g. observing spending caps). An archive of individual cloud traces may also be helpful for reference, if a case manager would like to review a particular solution from a previous case.
- Accumulated cloud traces can serve as a basis for a variety of statistical evaluations, e.g. the time required for researching particular facts or making particular decisions (broken down e.g. by case manager or by patient), the number of iterations on particular artifacts (e.g. healing plans), the frequency of accessing paid data sources (e.g. archives of medical journals), etc.
- Insights from accumulated cloud traces can also help to improve the template that those cases were derived from, by focusing the template on the most frequently used and most relevant agenda items and artifacts.
- Alternatively, analysis of accumulated cloud traces may reveal different classes of process clouds that suggest the creation of separate templates for each class.
- Finally, analysis of accumulated cloud traces may reveal recurring structures that indicate that the process cloud actually contains implicit structured process fragments, whose automation might improve efficiency.

To help case managers to keep an overview of their progress on each case, find information relevant to it and work with it, we are also developing suitable tool support. The user interface prototype of our OPC toolset provides three window panes, as shown in Fig. 2: The *agenda pane* on the left displays the agenda items that need to be worked on during the traversal of this case. Clicking on any agenda item will bring up all associated artifacts in the *artifacts pane* – the main workspace – in the middle. The *source pane* on the right provides detailed views of the various artifacts that are relevant to a case, as well as the data sources from which those artifacts are imported.



Post-fracture avascular necrosis of the fe
correlation of experimental and clinical st
(PMID:7438624)

Calandruccio RA, Anderson WE 3rd
Clinical Orthopaedics and Related Research [1980(152):49-84]

Type: Journal Article

Abstract

Both extraosseous and intraosseous blood supply of the femoral
patients with femoral neck fractures. The injury to the extraosse
amount of displacement at the time of fracture. The major intrac
fracture extends through the superior lateral portion of the neck
significant age difference in adults who develop avascular necro
unites compared with those who do not develop avascular necro
of avascular necrosis following femoral neck fracture is probably
nondisplaced fractures should be reported separately from displa
be reported following united fractures to avoid the confusion in

Fig. 2. User interface prototype of OPC workspace with agenda, artifact and source panes

4 Related Work

A prominent approach to describe processes with a certain control flow variability is the "case handling paradigm" as developed by van der Aalst et al. [2]. There, a case consists of activities, data objects, precedence relations between activities, and relations between them, as well as a state space used to describe the current state of activities and data objects. Our approach also focuses on required data objects and defined goals of unstructured activities. However, in contrast to the "cases" in the case handling paradigm, our "process clouds" do not prescribe any precedence relation or control flow definition for their internal steps. Instead, we provide an agenda as a guideline to case managers, but no explicit order of agenda items or control flow elements that would introduce some notion of a structured process.

A further approach called "activity schemes" is presented by Schmidt et al. in [3]. They describe knowledge activities as a graph of knowledge action nodes that includes relations between activities, applications, and resources. In contrast to this knowledge-focused approach, we focus on steps that have to be taken and goals that have to be achieved during a process cloud's traversal.

Holz et al. use a "task list" as a central concept for supporting case management [4]. This task list is comparable to our agenda due to the possibility of relating resources to task list items. The idea of using templates in the context of supporting unstructured activities is motivated and described by Riss et al. in [5]. Following them, we developed a specific template lifecycle for our agenda.

Besides providing tools such as agendas and templates, a further approach for supporting case management is monitoring and analyzing the actions of case managers. Process mining [6], for example, aims to identify recurring process patterns or work patterns [7]. In the context of open process clouds, we are interested in supporting case managers to decide about the next step in a process cloud as well. In [8], for example, Schonenberg et al. describe a concept for logging the actions of case managers, and provide weight functions to recommend what to do in the next step. However, in contrast to our approach, they consider neither the context of a concrete activity, nor required or used resources.

5 Conclusion

In this paper, we introduced the OPC approach for handling so-called process clouds, i.e. segments within a structured business process that cannot be executed according to a pre-defined series of steps, but whose completion relies extensively on the expertise and judgment of a case manager who works with a broad spectrum of data sources in order to arrive at results. To support work in such process clouds, we introduced the concepts of agendas and templates to guide the case manager, and the artifact workspace in which he can collect and evaluate relevant data from highly heterogeneous resources. By having all relevant information consolidated in one tool, being able to relate them and go back to their sources if necessary, we expect to free case managers from a significant cognitive load that the operational aspects of their research work would otherwise impose on them.

References

1. Singularity: Case Management – Combining Knowledge With Process. BPTrends (July 2009), `http://www.bptrends.com`
2. van der Aalst, W.M.P., Weske, M., Gruenbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Eng. 53(2), 129–162 (2005)
3. Schmidt, B., Stoitsev, T., Muehlhaeuser, M.: Activity-centric support for weakly-structured business processes. In: Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2010), pp. 251–260. ACM (2010)
4. Holz, H., Maus, H., Bernardi, A., Rostani, O.: From Lightweight, Proactive Information Delivery to Business Process-Oriented Knowledge Management. Journal of Universal Knowledge Management. Special Issue on Knowledge Infrastructures for the Support of Knowledge Intensive Business Processes (2), 101–127 (2005)
5. Riss, U., Rickayzen, A., Maus, H., van der Aalst, W.M.P.: Challenges for Business Process and Task Management. Journal of Universal Knowledge Management (2), 77–100 (2005)
6. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
7. Makolm, J., Weiß, S., Reisinger, D.: DYONIPOS: Proactive Knowledge Management. In: Proc. 21st Blede Conference eCollaboration: Overcoming Boundaries through Multi-Channel Interaction, pp. 476–482 (2008)
8. Schonenberg, H., Weber, B., van Dongen, B., van der Aalst, W.M.P.: Supporting Flexible Processes through Recommendations Based on History. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)

A Generic Framework for Service-Based Business Process Elasticity in the Cloud

Mourad Amziani[1,2], Tarek Melliti[2], and Samir Tata[1]

[1] Institut Mines-Telecom, TELECOM SudParis, UMR CNRS Samovar, Evry, France
[2] University of Evry Val d'Essonne, IBISC, Evry, France

Abstract. Cloud computing is a new model for the provisioning of dynamically elastic and often virtualized resources at the levels of infrastructures, platforms and software. Cloud platforms are being increasingly used for the deployment and execution of service-based business processes (SBPs). Nevertheless, the provisioning of elastic infrastructures and/or platforms is not sufficient to provide users with elasticity at the level of SBPs. Therefore, there is a need to provide SBPs with mechanisms to scale their resource requirements up and down whenever possible. This can be achieved using mechanisms for duplicating and consolidating business services that compose the SBPs. In this paper, we propose a formal model and a generic framework for elasticity of SBPs.

1 Introduction

Cloud computing is a new delivery model for IT services based on Internet protocols. It typically involves provisioning of dynamically scalable and often virtualized resources at the infrastructure, platform and software levels. Cloud environments are being increasingly used for deploying and executing business processes and particularly service-based business processes (SBPs) that are made up of components that provide business services. One of the expected facilities of cloud environments is elasticity at different levels.

At the platform as a service (PaaS) level, the deployed processes should be provided with platform mechanisms that can scale up and down whenever needed. In this context, we have conducted studies of existing application servers and SBP engines. These classical platforms are not elastic [6]. For that reason, we have developed a new model for service deployment called micro-container [8]. Our approach was based on a simple idea that consists in dedicating a micro-container with minimal and personalized functionalities to manage the life cycle of each deployed services. With this idea we have shown the elasticity of services deployed at the PaaS level can be ensured [8]. In addition, we have shown that elastic micro-containers can be used to host service-based application.

Nonetheless, provisioning of elastic platforms, *e.g.* based on micro-containers, is not sufficient to provide users with elasticity of the deployed business process (at the Software as a Service (SaaS) level). Therefore, SBPs should be provided with elasticity so that they would be able to adapt to the workload changes while ensuring the desired functional and non-functional properties. In this paper we address elasticity at the level of SBPs that mainly raises the following questions.

- What mechanisms should be developed to perform elasticity of SBPs?
- How to evaluate elasticity strategies of SBPs?

Performing elasticity consists in providing cloud environments with mechanisms that allow deployed SBPs to scale up or down. To scale up a SBP, elasticity mechanisms have to create, as many copies as necessary, of some business services (part of the considered SBP). To scale down a SBP, elasticity mechanisms have to remove unnecessary copies of some services.

Many strategies that decide on when SBP elasticity is performed can be proposed. It would be useful for a Cloud provider to have an evaluation framework in order to make a better decision on the elasticity strategy to adopt.

In this paper, we propose a formal model for SBP elasticity and a framework to evaluate elasticity strategies.

2 Model for SBPs Elasticity

We are interested in this paper in modelling elasticity of SBPs. A SBP is a business process that consists in assembling a set of elementary IT-enabled services. These services realise the business activities of the considered SBP. Assembling services into a SBP can be ensured using any appropriate service composition specifications (*e.g.* BPEL). Elasticity of a SBP is the ability to duplicate or consolidate as many instances of the process or some of its services as needed to handle the dynamic of received requests. Indeed, we believe that handling elasticity does not only operate at the process level but it should operate at the level of services too. It is not necessary to duplicate all the services of a considered SBP while the bottleneck comes from some services of the SBP.

2.1 SBP Modeling

We model the SBP using Petri nets. Many approaches model SBPs using petri nets, but instead of focusing on the execution model of processes and their services, we focus on the dynamic (evolution) of loads on each basic service participating in the SBP's composition. In the proposed model, each service is represented by a place. The transitions represent calls transfers between services.

Definition 1. *A SBP load model is a petri net* $N =< P, T, Pre, Post >$:

- *P: a set of places (represents the set of services involving in a SBP).*
- *T: a set of transitions (represents the call transfers between services according to the SBP behavioural specification).*
- $Pre : P \times T \rightarrow \{0, 1\}$
- $Post : T \times P \rightarrow \{0, 1\}$

For a place p and a transition t we give the following notations:

- $p^\bullet = \{t \in T | Pre(p,t) = 1\}$. ${}^\bullet p = \{t \in T | Post(t,p) = 1\}$
- $t^\bullet = \{p \in P | Post(t,p) = 1\}$. ${}^\bullet t = \{p \in P | Pre(p,t) = 1\}$

Definition 2. *Let N be a Petri net, we define a net system $S = \langle N, M \rangle$ with $M : P \to \mathbb{N}$ a marking that associates to each place an amount of tokens.*

The marking of a Petri net represents a distribution of calls over the set of services that compose the SBP. A Petri net system models a particular distribution of calls over the services of a deployed SBP.

Definition 3. *Given a net system $S = \langle N, M \rangle$ we say that a transition t is fireable in the marking M, noted by $M[t\rangle$ iff $\forall p \in^\bullet t : M(p) \geq 1$.*

Definition 4. *The firing of a transition t in marking M changes the marking of the net system to M' s.t. $\forall p : M'(p) = M(p) + (Post(t,p) - Pre(p,t))$, we note the transition by $M[t\rangle M'$.*

The transition firing represents the evolution of the load distribution after calls transfer. The way that calls are transferred between services depends on the behaviour specification (workflow operators) of the SBP.

2.2 Elasticity Operations

Place Duplication

As noted before places represent services deployed on containers. The marking of a place denote the number of the current instances (or requests) of the service. Each service has a maximal capacity over what the QoS of the service decrease and can leads to the stuck of the container and by the same way the crash of the service. Giving to the container more memory and/or more CPU time will not change the issue of the problem [8]. A solution to this problem is to duplicate the service without changing underlying SBP.

Definition 5. *Let $S = \langle N, M \rangle$ be a net system and let $p \in P$, the duplication of p in S by a new place p^c ($\notin P$), noted as $D(S, p, p^c)$, is a new net system $S' = \langle N', M' \rangle$ s.t*

- $P' = P \cup \{p^c\}$
- $T' = T \cup T''$ *with* $T'' = \{t^c | t \in ({}^\bullet p \cup p^\bullet)\}$
- $Pre' : P' \times T' \to \{0, 1\}$
- $Post' : T' \times P' \to \{0, 1\}$
- $M' : P' \to \mathbb{N}$ *with* $M'(p') = M(p')$ *if* $p' \neq p^c$ *and* 0 *otherwise.*

The Pre' (respectively $Post'$) functions are defined as follow:

$$Pre'(p',t') = \begin{cases} Pre(p',t') & p' \in P \wedge t' \in T \\ Pre(p',t) & t \in T \wedge t' \in (T' \setminus T) \wedge p' \in (P \setminus \{p\}) \\ Pre(p,t) & t \in T \wedge t' \in (T' \setminus T) \wedge p' = p^c \\ 0 & otherwise. \end{cases}$$

$$Post'(t',p') = \begin{cases} Post(t',p') & p' \in P \wedge t' \in T \\ Post(t,p') & t \in T \wedge t' \in (T' \setminus T) \wedge p' \in (P \setminus \{p\}) \\ Post(t,p) & t \in T \wedge t' \in (T' \setminus T) \wedge p' = p^c \\ 0 & otherwise. \end{cases}$$

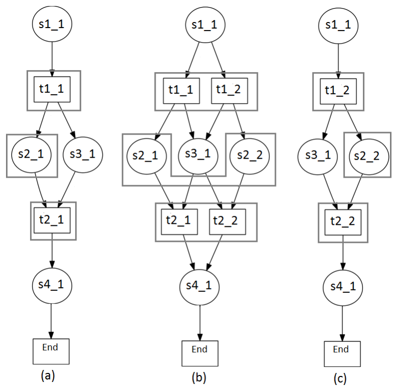


Fig. 1. Example of the elasticity of a SBP

Place Consolidation
When a service has few calls, the containers that host its instances use more resources than required for the same QoS. As a dual operator to duplication we define the consolidation or merging operator that removes a copy of a service.

Definition 6. *Let $S = \langle N, M\rangle$ be a net system and let p, p^c be two places in N with $p \neq p^c$, the consolidation of p^c in p, noted as $C(S, p, p^c)$, is a new net system $S' = \langle N', M'\rangle$ s.t*

– *N': is the net N after removing the place p^c and the transitions $(p^c)^\bullet \cup {}^\bullet p^c$*
– *$M' : P' \to \mathbb{N}$ with $M'(p) = M(p) + M(p^c)$ and $M'(p') = M(p')$ if $p' \neq p$.*

Example 1. Figure 1-(a) shows an example of nets system that represents an SBP. Figure 1-(b) is the resulted system from the duplication of $s2\_1$ in (a). Figure 1(c) is the consolidation of the place $s2\_1$ in its copy $s2\_2$.

3 A Generic Framework for SBPs Elasticity

Usually in the Cloud, a set of policies is implemented to guarantee some SLA properties to the deployed applications. These policies are implemented in what is usually called controller. In our case, we are interested in elasticity policies of services that compose a SBP. In order to achieve this, we want to develop a controller to provide an optimal ratio QoS and allocated resources of a SBP. The proposed controller (Figure 2)-(a) is able to perform three actions:

– Routing: Routing decision is about the way a load of services is routed over the set of their copies. It determines under which condition a given transition (call transfer) is fired. One can think of routing as a way to define a strategy to control the flow of load according to some rules *e.g.* a call is transferred iff the resulted marking does not violate the capacity of the services.
– Duplication: Duplication decision is about the creation of a new copy of a service in order to meet its increased workload.

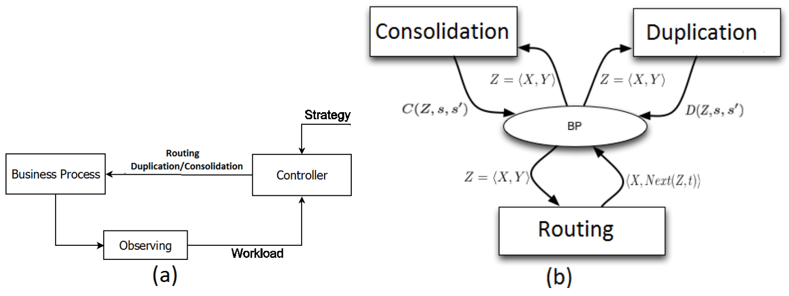

Strategy
Business Process
Routing
Duplication/Consolidation
Controller
Observing
Workload
(a)
Consolidation
Duplication
$Z = \langle X, Y \rangle$
$Z = \langle X, Y \rangle$
$C(Z, s, s')$
$D(Z, s, s')$
BP
$Z = \langle X, Y \rangle$
$\langle X, Next(Z, t) \rangle$
Routing
(b)

Fig. 2. General architecture of the controller

– Consolidation: Consolidation decision is about the removing of an unnecessary copy of a service in order to meet its workload decrease.

If we consider the three actions that can be performed by an elasticity controller, any combination of conditions associated with a decision of routing, duplication and consolidation is an elasticity strategy. The strategy is responsible of making decisions on the execution of elasticity mechanisms *i.e.* deciding when and how to use these mechanisms. Several strategies can be used to manage the SBP elasticity [4]. The abundance of possible strategies requires evaluating these strategies before implementing them. Our goal here is not to propose an additional elasticity strategy, but a framework, called generic controller that allows the implementation and evaluation of SBPs elasticity strategies.

We model the controller as a high level Petri net (HLPN). The structure of the controller is shown in Figure 2-(b). The controller HLPN contains one place (BP) of type net systems (SBPs). The marking of this place is modified by three transitions that represent the three elasticity mechanisms (Routing, Duplication and Consolidation). Each of these transitions is guarded by a generic condition. Implementing a strategy consists then in instanciating the three generic conditions. The reachability graph resulted from the instanciated controller represents the different evolutions of the SBP according to the strategy. Using HLPN analysis tools, the evaluation of the strategy can be processed by model-checking its reachability graph.

4 Related Work

The elasticity in the Cloud has been studied in the past. Proposed approaches use generally sets of rules to make decisions about the elasticity of the infrastructure. In this kind of approaches, several techniques have been used. In [3] the authors propose to add or remove VMs according to demands. In [5,1] the authors propose to calculate the optimal number of VMs to be deployed according to variations of demands.

The use of duplication/consolidation mechanisms to provide elasticity have been considered in the area of dynamic service deployment [2,7]. The proposed mechanisms allow the duplication/consolidation of the entire SBP (and so, of all its services) while the bottleneck may come from some services of the SBP.

At the best of our knowledge the approaches for elasticity are interested in the infrastructure level of cloud environments (IaaS). As stated before, ensuring elasticity at the IaaS level is not sufficient to provide users with elasticity of deployed SBPs. Similarly, ensuring elasticity at the PaaS level is not enough to ensure elasticity of deployed SBPs. We believe that elasticity should be handled and tuned at different levels of cloud environments. The work we present in this paper is novel in the sense that it (1) tackles the problem of elasticity at the SaaS level and (2) proposes a generic framework for evaluating SBPs elasticity.

5 Conclusion

This paper addresses the problem of elasticity of service-based business processes (SBPs) deployed in cloud environments. Unlike existing work, the proposed approach tackles the elasticity at the level of SBPs. To perform elasticity we proposed using Petri nets tow operations: duplication and consolidation. In addition, we have proposed a framework to evaluate SBPs elasticity. As perspectives of this work, we are working on the implementation of the elasticity operations into CloudServ (a PaaS under development within the French FUI CompatibleOne project).

References

1. Bi, J., Zhu, Z., Tian, R., Wang, Q.: Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: IEEE International Conference on Cloud Computing, pp. 370–377 (2010)
2. Chrysoulas, C., Kostopoulos, G., Haleplidis, E., Haas, R., Denazis, S., Koufopavlou, O.: A decision making framework for dynamic service deployment
3. Duong, T.N.B., Li, X., Goh, R.S.M.: A framework for dynamic resource provisioning and adaptation in iaas clouds. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 312–319 (2011)
4. Ghanbari, H., Simmons, B., Litoiu, M., Iszlai, G.: Exploring alternative approaches to implement an elasticity policy. In: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011, pp. 716–723 (2011)
5. Han, R., Guo, L., Guo, Y., He, S.: A deployment platform for dynamically scaling applications in the cloud. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 506–510 (2011)
6. Mohamed, M., Yangui, S., Moalla, S., Tata, S.: Web service micro-container for service-based applications in cloud environments. In: WETICE, pp. 61–66 (2011)
7. Weissman, J.B., Kim, S., England, D.: A framework for dynamic service adaptation in the grid: Next generation software program progress report. In: International Parallel and Distributed Processing Symposium (2005)
8. Yangui, S., Mohamed, M., Tata, S., Moalla, S.: Scalable service containers. In: CloudCom, pp. 348–356 (2011)

A Business Process-Driven Approach for Requirements Dependency Analysis

Juan Li[1], Ross Jeffery[2,3], Kam Hay Fung[4], Liming Zhu[2,3], Qing Wang[1], He Zhang[2,3], and Xiwei Xu[2]

[1] Institute of Software, Chinese Academy of Sciences, China
[2] National ICT Australia, Australia
[3] School of Computer Science and Engineering, The University of New South Wales, Australia
[4] School of Information Systems, Technology and Management, The University of New South Wales, Australia
{lijuan,wq}@itechs.iscas.ac.cn,
{Ross.Jeffery,Liming.Zhu,He.Zhang,Xiwei.Xu}@nicta.com.au,
kamhayfung@ieee.org

Abstract. Dependencies among software artifacts are very useful for various software development and maintenance activities such as change impact analysis and effort estimation. In the past, the focus on artifact dependencies has been at the design and code level rather than at the requirements level. This is due to the difficulties in identifying dependencies in a text-based requirements specification. We observed that difficulties reside in the disconnection among itemized requirements and the lack of a more systematic approach to write text-based requirements. Business process models are an increasingly important part of a requirements specification. In this paper, we present a mapping between workflow patterns and dependency types to aid dependency identification and change impact analysis. Our real-world case study results show that some participants, with the help of the mapping, discovered more dependencies than other participants using text-based requirements only. Though many of these additional dependencies are highly difficult to spot from the text-based requirements, they are however very useful for change impact analysis.

Keywords: Business process modeling, workflow pattern, software development and maintenance, requirements dependency.

1 Introduction

In a volatile environment, software systems must evolve to adapt to the rapid changes of stakeholders' needs, technologies and the business [1]. A change can impact not only source code, but also other software artifacts, such as requirements, design and test cases [2]. To analyze the impact of a proposed software change, one should determine which parts of the software system may be affected by the change and ascertain their possible risks [3]. Bohner [3] proposed an impact analysis process that examines change requests to identify the Starting Impact Set (SIS) of software artifacts that could be affected by these requests. The SIS is then analyzed to identify other artifacts to be affected, which are then incorporated into SIS to form the Candidate Impact Set (CIS). His process's goal is to estimate a CIS that is as close as possible to the set of artifacts that is actually modified after all the changes are made.

A CIS can be determined starting from the requirements specifications affected by the changes to the source code level [4] through information about traceability [5]. A traceability link is "any relationship that exists between artifacts involved in the software-engineering life cycle" [6]. A link can be between artifacts in different models (e.g. requirements and code) or between artifacts within a model. Requirements dependency, an example link, characterizes the relationship between requirements within a requirement model and acts as a basis from which a CIS is analyzed. The CIS include not only requirements to be affected directly by the change requests but also requirements to change potentially. To improve the accuracy of a CIS, it is useful to associate semantics with traceability links [3], such as the use of requirement dependency types since they convey important information for change impact analysis. Requirements dependency discovery tends to require a significant effort especially when the set of requirements is large. Studies have explored ways to automate the discovery (e.g. [7]). However, the solutions to date are immature and human experts are still relied upon for a large set of requirements [8].

Traditionally text is the primary (or even only) means of documenting requirements specification. Our previous empirical evaluation [9] found that many dependencies, which are especially useful for change impact analysis, were not spotted during dependency discovery in text-based requirements alone. An explanation for this is that even when they were closely related to business processes and rules, they could not be represented explicitly in text-based requirements.

It is no doubt that a diagram is worth ten thousand words [10] and hence Business Process Modeling Notation (BPMN) based models [11] combined with text-based requirements should increase the likelihood of finding more dependencies than the latter alone in requirements dependency analysis. In this research, however, we are interested in *how* BPMN models can help requirements dependency analysis and change impact analysis in a more concrete fashion. In this spirit, we propose a mapping between workflow patterns in BPMN and dependency types to supplement text-based requirement dependency analysis by systematically and manually deriving dependencies from a typical business process model. We conducted a case study on a real-world industrial project to evaluate our approach and confirmed that practitioners using our approach did discover additional dependencies useful to change impact analysis.

2 Related Work

2.1 Dependency and Change Impact Analysis

Research in traceability is gaining attention in requirements engineering [12]. Correct traceability is the basis for change propagation analysis [13], important for all aspects of a software development project. In requirements engineering, requirements relationships are classified into dependency types based on the structural and semantic properties of requirements, to help practitioners identify these relationships (e.g. [14]). Requirements dependencies also play an important role in change impact analysis. Hassine et al. applied dependency analysis at the use case map level (rather than between requirements in natural languages) to identify the potential impact of requirement changes on the overall software system [12]. Yan et al. discussed the ripple-effect of requirements evolution based on requirements dependencies [15].

2.2 Business Process Modeling in Requirements Engineering

Business process models (BPMs) are widely used in requirements engineering. To the best of our knowledge, however, no studies have considered applying BPMs in requirements dependency discovery to facilitate change impact analysis. For instance, to bridge business process modeling with requirements elicitation and analysis, de la Vara González and Díaz [16] described a business process-driven requirements engineering approach to derive requirements from organizational models that express business strategies and from business processes in BPMN. Cardoso et al. [17] used business process models to derive alternative sets of requirements for a process-oriented software system. These sets capture different decisions regarding the intended "level of automation" for various activities in a business process. Mathisen et al. [18] presented an approach for early detection of structural changes that have implications for the software architecture. Their approach hinges on using business process modeling to increase the level of understanding of the problem domain in early stages of a project.

3 Requirements Dependency Analysis Based on Business Process Models

Before delving into the details of our approach, we use a stimulating example to highlight its use. In Fig. 1, a simple BPM for processing home loan applications consists of four sequential processes as shown. Let us focus on the middle two processes, viz. "Check Credit" and "Approve Loan". The former performs credit checking on loan applicants and fulfills three requirements (UC1, UC2 and UC3). The latter lets a loan assessor approves loan applications and involves two requirements (UC4 and UC5). These two processes form a *sequence* workflow pattern in that "Check Credit" precedes "Approve Loan". The pattern logically *connects* requirements UC1, UC2 and UC3 to UC4 and UC5. In the context of dependency, UC1, UC2 and UC3 are regarded as *preconditions* for UC4 and UC5. More generally, the power of BPMs connects individual text-based requirements into a manageable and well-scoped visual structure. The connections are often useful guidance for implementation-related dependency identification. Sometimes it is easy to spot preconditions based on the textual descriptions of the requirements without the help of BPMs. To calculate how much an applicant can borrow (UC5), one first needs to know how much the person owes (UC2) and his/her income level (UC3). At other times, dependencies are non-trivial from textual descriptions whence BPMs may provide some hints (e.g. connecting UC1 and UC4).

The example just showcased that BPMs can be useful for dependency discovery. Now we present the formal definitions of the dependency type model plus the mapping between workflow patterns and dependency types.

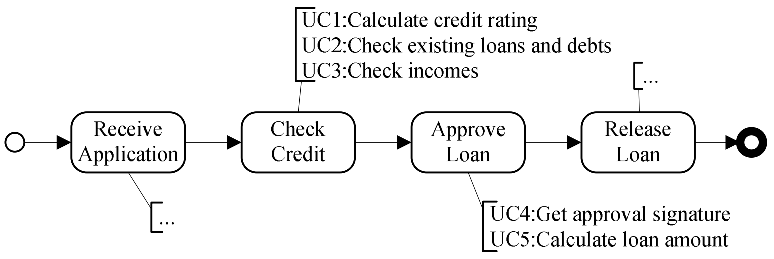

UC1:Calculate credit rating
UC2:Check existing loans and debts
UC3:Check incomes
...
Receive Application
Check Credit
Approve Loan
Release Loan
...
UC4:Get approval signature
UC5:Calculate loan amount

Fig. 1. BPM example

3.1 Requirements Dependency Model

Many dependency types have been proposed and they have different levels of abstraction and different criteria for categorization. Their complexity and diversity gives rise to a steep learning curve, which contributes to the difficulty in comprehending, using and evaluating them. Thus, in earlier work [9], we surveyed dependency types from the literature, consolidated them into a requirements dependency model and empirically evaluated its applicability in dependency identification and change impact analysis in a real-world industry project. Here we divide its twenty-three dependency types into three categories:

- *Document-related*: This category of dependency types is embedded in the structure, content and version relationships in the requirements representation. For instance, a requirement can have a "formalizes" dependency on another requirement, which means the former is defined more formally (using computational logic, business rules, constraints, etc.) than the latter.
- *Value-related*: This category is concerned with the relation between the realization of one requirement to the value that a customer/user perceives the realization of another requirement will provide [9]. For example, the adoption of a complex user-interface style can increase the usability of the user interface for a web-based application (i.e. "increases_cost_of" dependency type). This category can be useful for selecting the set of requirements to be fulfilled in release planning.
- *Implementation-related*: In this category, the realization of a requirement relates to one or more requirements. For instance, the requirement "withdraw cash" calls for the requirement "calculate account balance" to be realized to determine the withdrawal limit for a bank account. Dependency types in this category often indicate change propagations among low level models such as the detailed design and the code, which are important for change impact analysis.

The dependency types in each category are listed in Table 1. In this study, we concentrate on the implementation-related dependency types because they are the most useful and relevant to software design and development. Due to space limitations, we do not elaborate further on other dependency types. Interested readers may refer to [9] for the in-depth discussions. The implementation-related dependency types are:

- *Constraints*: A requirement can relate to another by being a constraint to the latter. For instance, the requirement "cash withdrawal is limited to $2000 daily" is a constraint for the requirement "withdraw cash".
- *Refines*: One requirement can be a refinement of another requirement, providing more detailed descriptions for the latter.
- *Precondition*: Only after one function prescribed by a requirement is finished, or one condition described by the requirement is satisfied, that another function prescribed by a requirement can be performed. Usually precondition reflects the business rules or the sequence relationship between (sub-)processes. For instance, the requirement "a customer successfully logged in to the application" is a precondition for "a customer withdraws cash from bank account".
- *Satisfies*: This type expresses that if one requirement is implemented in an application, the implementation will also satisfy another requirement. For example, the requirement "when a user logs out, all opened documents will be automatically saved" satisfies the requirement "no unsaved work will be allowed before the editor terminates".
- *Similar_to*: The prescription of one requirement (e.g. "display account balance") is similar to or overlapping with one or more requirements (e.g. "display amount available for withdrawal").

Table 1. Dependency Type Model [9]

Category	Dependency types
Document-related	Compares, Contradicts, Conflicts, Example_for, Test_case_for, Purpose, Comments, Background, Replaces, Based_on, Elaborates, Generation, Changes_to, Formalizes
Value-related	Increase_cost_of, Decreases_cost_of, Increase_value_of, Decreases_value_of
Implementation-related	Constraints, Refines, Precondition, Satisfies, Similar_to

3.2 Business Process Driven Dependency Identification

In recent years, BPMs are increasingly used in requirements analysis to define system goals and requirements. We believe BPMs can also be used to discover missing and ambiguous requirements in requirements specifications. In our approach, we adopted BPMN as the business process modeling language because it is a widely used business process modeling language and covers most of the workflow patterns compared to other modeling languages.

A well-known collection of Workflow Patterns was proposed in [19, 20]. This work provides a comprehensive examination of the various perspectives (control flow, data, resource, and exception handling) of workflows. Each workflow pattern precisely defines recurring semantics between some process elements. Using the evaluation of BPMN version 1.0 against the workflow control-flow patterns [21], we map all relevant workflow patterns supported by BPMN models to all the implementation-related dependency types. The mapping is an informal one with no mathematic rigor as it is targeted for business analysts and requirements engineers to identify informal dependencies. We derived the mappings largely from the expert opinions and our observation in the previous case study [9]:

- Certain dependency types are often associated with certain workflow patterns. For example, we observed empirically that Precondition dependency types are often associated with workflow patterns such as the sequence workflow pattern.
- Some workflow patterns are similar to each other in terms of their associations with the dependency types. For example, sequence, merge/split-related workflow patterns are too often pointing to the Precondition dependency type. Thus, we created more generic definitions for these similar patterns to further reduce the complexity of the mapping.

The mapping only acts as an informal guidance and is highly indicative. There are many-to-many relationships between the workflow patterns and the dependency types. It is up to practitioners to confirm each instance of dependency identification in a particular project. We acknowledge the limitation and imprecision of the mapping but consider it helpful in the task as demonstrated by the later evaluation. We also excluded the workflow patterns on data, resource and exceptional handling in this study due to the scope. However, we believe they will also be helpful in identifying requirements dependencies and we plan to investigate them in future work. The mapping results are summarized in Table 2.

Table 2. Mapping between workflow patterns and dependency types

Workflow Patterns [19, 20]	Relevant Generic Pattern	Dependencies
sequence, split-related, merge-related, triggers	Generic Sequence	Precondition
merge-related	Generic Merge	Similar_to
split-related	Generic Split	Similar_to
cancelling/termination-related, interleaved routing, thread merge	Generic Crosscut	Constraints
Individual workflow patterns that provide more details to the decision points logic, sub-process	Generic Detailing	Refines, Satisfies

Generic Sequence

A (sub-)process is enabled after the completion of a preceding (sub-)process. For example, a bank allocates a valuation task to a valuer after a client has submitted the valuation request. Many workflow patterns, such as split-related, merge-related and triggers [19, 20], indicate a precedence-successor relationship and are therefore grouped under it. Due to the large number of workflow patterns involved, we do not list all of them here. Although this is one of the most obvious patterns, the textual descriptions of this pattern's instances are often scattered around or only implied in text-based requirements, resulting in the dependencies not being found. This is a basic pattern which may be included in the other workflow patterns such as split-related and merge-related patterns. The corresponding dependency type for Generic Sequence is "Precondition" (e.g. A is a precondition of B in Fig. 2(a)).

Generic Split
This represents the divergence of a (sub-)process into two or more (sub-)processes (i.e. branches). For example, after a client has proposed property valuation requests, one or more of the "desktop valuation", "curbside valuation" or "full valuation" can be performed. This generic pattern includes all the workflow patterns for choices and splits, such as parallel split, synchronization, exclusive choice, multi-choice, thread split [19, 20]. In some of the splitting instances, the branches serve a similar purpose (Similar_to each other) and only one or more of them are selected. However, the textual descriptions may look very different and are again scattered within requirements. Our case study described later on is one such example. Practitioners missed some of the similar relationship among the text-based requirements but they were easily identified in the BPMN models. The corresponding dependency type for Generic Split is "Similar_to" (e.g. B and C are similar to each other in Fig. 2(b)).

Generic Merge
This represents the convergence of two or more (sub-)processes (i.e. branches) into a single subsequent (sub-)process. For example, "prepare invoice" and "send for approval" happen simultaneously and join before "send report". This generic pattern covers all the workflow patterns for synchronization, merge and join and discriminators, such as structured synchronizing merge, multi-merge, thread merge [20]. The rationale behind this pattern is similar to the Generic Split pattern. For example, task "prepare valuation reports" can be started if and only if all the valuation results are returned. An example of change request is checking the quality of the reports before allowing an administrator to print them. This may result in checking all the valuation results before they are returned. The corresponding dependency type for Generic Merge is "Similar_to" (e.g. A and B are similar to each other in Fig. 2(c))

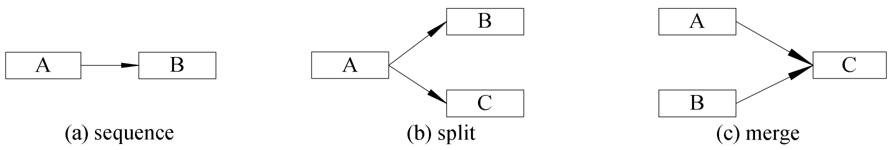


Fig. 2. Workflow patterns used in the mapping

Generic Crosscut
The workflow patterns grouped under this category are cancelling, thread merge and interleaved routing [20]. Cancelling-related workflow patterns (cancelling discriminator, cancelling partial join, etc.) abort the execution of other (sub-)processes and so

crosscut many of these (sub-)processes. BPMs help identifying cross-cutting impacts of exceptional events such as Canceling. In BPMN models, for example, all the preceding (sub-)processes of a cancellation point can act as a source of dependency candidates narrowing down the scope of possibly affected requirements. These dependencies are difficult to identify across text-based requirements. Thread merge and interleaved routing workflow patterns concern the control of execution threads (i.e. multiple (sub-)processes) at runtime. The information they specify crosscuts multiple requirements and is often not captured in text-based requirements. The corresponding dependency type for Generic Crosscut is "Constraints".

Generic Detailing

All decision points in BPMN models have precise meanings. For example, a split can mean running in parallel, an exclusive choice or a multiple choice. In text-based requirements, such precise meanings are often elaborated separately and it is often difficult to spot the dependency between the elaboration and its original requirements. All such decision points related patterns also help tremendously in terms of identifying missing and/or ambiguous requirements. The corresponding dependency types are "Refines" and "Satisfies". The sub-process notations in BPMN also have similar meanings with respect to and are obvious helpers for identifying these two dependency types.

The same type of dependencies identified in BPMs often has different nature from those identified in text-based requirements. For example, Similar_to mostly refers to task similarity in BPMs but data similarity is often identified in text-based requirements. In BPMs, Similar_to can exist in the generic merge pattern where several (sub-)processes join to proceed to the next (sub-)process. These (sub-)processes may have some similarity in talking to the next activity. But in text-based requirements, Similar_to can exist between many functional requirements dealing with the same data information. Another example is Constraints. It indicates constraints that non-functional requirements, such as security-related non-functional requirements, have on functional requirements. These new dependencies are usually not identified in text-based requirements but very useful to impact analysis.

3.3 Usage Guidelines

While the mapping is useful for identifying dependencies in requirements, it is expected to be used in a larger framework. For instance, one can define a mini-process for requirements elicitation for business-process oriented applications as below:

1. Develop text-based requirements and supplement them with BPMs. They cover business-level activities, business goals and business rules.
2. Identify dependencies among requirements. This makes use of our mapping between workflow patterns and dependency types to help analyze requirements dependencies.
3. Perform coverage checking between BPMs and text-based requirements so that missing and ambiguous text-based requirements can be discovered and resolved along with dependency analysis.

Our work presented in this paper contributes to Step 2 above, relying on availability of BPMs. Steps 1 and 3 have their own complexities and we do not elaborate them further because of space limitations. Note that the mapping involves overheads, which includes learning the dependency types and the mapping, eliciting detailed BPMs and applying the mapping. However, these can be offset with the productivity gained from identifying dependencies and a more accurate CIS (since more impacted requirements are found), both of which exemplified by the case study in the next section.

Note that while the mapping can be useful for identifying the dependencies of and hence linking text-based requirements, mapping text-based requirements onto BPM structures and vice versa is an additional problem in itself. For instance, a security requirement can cross-cut several parts of an application and may be linked to many parts of a BPM. Due to space limitations, this issue is deemed out of scope.

4 Case Study

4.1 Questions to Evaluate and Case Selection

To evaluate our approach, we undertook a case study to answer these questions:

Q1: Can more dependencies be found by using both BPM and text-based requirements than using text-based requirements alone?
Q2: Are additional dependencies found in Question 1 actually useful in change impact analysis and how?

The kinds of data linked to research questions and collected through a questionnaire are shown in Table 3.

Table 3. Research question and data to collect

Question	Data to collect
Q1	Number of dependencies found through BPMs
	Number of dependencies found through text-based requirements
	Time spent on dependency discovery
Q2	Impacted requirements found through dependencies in BPMs
	Impacted requirements found through dependencies in text-based requirements
	Dependency types used in change impact analysis

The case we selected is a property valuation system (PVS) developed by NICTA (National ICT Australia) for a company employing the LIXI (Lending Industry XML Initiative) standards for the format and exchange of lending-related data using XML. LIXI is an independent non-profit organization established to remove data exchange barriers within the Australian lending industry. Through the work of LIXI, member organizations - including major banks, mortgage originators and brokers, mortgage insurers, property valuers, settlement agents, trustees and information technology providers - offer services to customers more efficiently and at lower costs. PVS had gone through changes with V2 being the latest. It included a “Pocket Valuer” sub-system on Personal Digital Assistant (PDA) for capturing property valuation data onsite, a desktop sub-system for managing information and a web-based business process system for managing the valuation workflows. The company planned to migrate all the desktop sub-system functions to a web-based system and implement new functions for PVS. In this paper, this whole new system is called PVS V3. This study was conducted as part of the PVS V3 requirements development and system design project. Our study used requirements in PVS V2 for dependency identification and change propagation analysis triggered by new requirements in PVS V3.

4.2 Case Study Procedures

Four practitioners, all experienced in requirements engineering, participated in this case study. They were evenly split into two groups: the Text group and the BPMN+Text group. The former used only text-based requirements specifications and the latter used both a BPMN process model and a text-based requirements specification. The BPMN+Text group were also familiar with the BPMN language. The case study was conducted as follows. Participants in the Text group were firstly given:

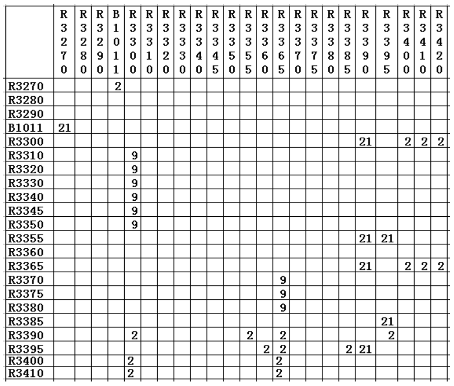
- thirty three change requests from PVS V3 (e.g. new functions);
- the dependency types and their definitions; and
- the PVS V2 requirements document written in natural language, consisting of 144 requirements organized into nineteen modules. Snippets of example modules “Valuation Requests” and “Valuation Bookings” are shown in Table 4:

Table 4. An example of text-based requirements for desktop application specification

Module	ID	Specification
1 - Valuation Requests	R3000	The ability to create valuation requests
	…..	…..
2 - Valuation Bookings	R3200	Requests should be able to be assigned to a valuer and booked for a specific date and time
	…..	…..

Participants in the Text group individually learned the dependency types, identified dependencies from the text-based requirements of PVS V2 and recorded them in a dependency matrix (example snippet in Fig. 3). Next, they analyzed the requirements estimated to be changed because of the change requests from PVS V3. Fig. 5(a) summarizes this procedure.

For the BPMN+Text group, the participants were handed the documentation to the Text Group plus the BPMN models to accompany PVS V2 and V3 requirements, a set of workflow patterns and the mapping between these patterns. After learning dependency types, the workflow patterns and the mapping, this group identified dependencies in the PVS V2 text-based requirements and the BPMN model, during which they identified the workflow patterns in the BPMN model and noted the dependencies in patterns in the BPMN model. Dependencies were recorded in a dependency matrix (e.g. Fig. 3) and annotated on the BPMN model (e.g. Fig. 4). Finally, they were asked to identify change propagation paths and dependency types used in change impact analysis and to analyze the requirements estimated to be changed as triggered by change requests in PVS V3. Fig. 5(b) depicts this procedure.



	R3270	R3280	R3290	B1011	R3300	R3310	R3320	R3330	R3340	R3345	R3350	R3355	R3360	R3365	R3370	R3375	R3380	R3385	R3390	R3395	R3400	R3410	R3420
R3270				2																			
R3280																							
R3290																							
B1011	21																						
R3300																			21		2	2	2
R3310					9																		
R3320					9																		
R3330					9																		
R3340					9																		
R3345					9																		
R3350					9																		
R3355																			21	21			
R3360																							
R3365																			21		2	2	2
R3370														9									
R3375														9									
R3380														9									
R3385																				21			
R3390					2							2		2						2			
R3395													2	2				2	21				
R3400					2									2									
R3410					2									2									

Fig. 3. An example of the dependency matrix

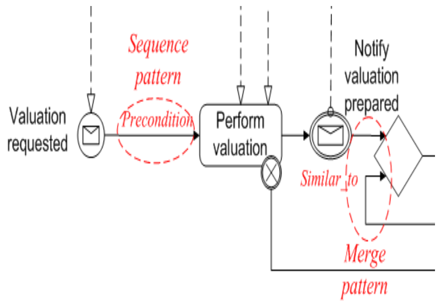


Fig. 4. An example of patterns and dependencies in the BPMN model

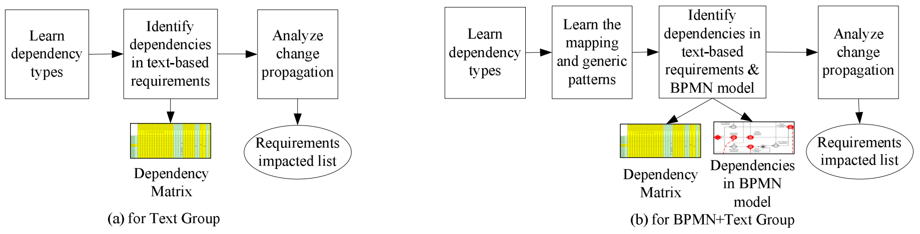

Learn dependency types
Identify dependencies in text-based requirements
Analyze change propagation
Dependency Matrix
Requirements impacted list
(a) for Text Group
Learn dependency types
Learn the mapping and generic patterns
Identify dependencies in text-based requirements & BPMN model
Analyze change propagation
Dependency Matrix
Dependencies in BPMN model
Requirements impacted list
(b) for BPMN+Text Group

Fig. 5. Case study procedures

4.3 Analysis Results

Q1. Can more dependencies be found by using both BPM and text-based requirements than using text-based requirements alone?

The BPMN+Text group found more dependencies with higher efficiency (i.e. higher number of dependencies found per hour) than the Text group (cf. Table 5). None found more than ten wrong dependencies because of their familiarity with PVS.

Table 5. Comparison of dependencies found

Category	Text Group	BPMN+Text Group
Number of implementation-related dependencies	125	231
Number of document-related and value-related dependencies	95	103
Total number of dependencies	220	334
Efficiency (number of dependencies identified per hour)	55	74.2

The numbers of document-related dependency found by two groups are similar, but there is 85% difference between the numbers of implementation-related dependencies found by both groups. To explain this difference, we analyzed the data, interviewed the participants and confirmed that the BPMN model and the mapping helped the BPMN+Text group to understand the text-based requirements and find more dependencies. This is because some dependencies are missing or not explicitly expressed in text-based requirements. This group also found twenty three requirements from the BPMN model that should be but were left out in the text-based requirements. This could be explained by the ambiguity of text-based requirements and a clearer relational view of business related requirements as provided by the BPMN model.

The implementation-related dependency distribution is shown in Fig. 6. The BPMN+Text group identified more instances of Precondition and Constraints dependencies than the Text group. There were many sequence patterns in the BPMN model corresponding to the Precondition dependency which were not explicitly represented in text-based requirements. Therefore, it was easier for the BPMN+Text group to discover Precondition dependencies. Additionally, there were many cancelling-related workflow patterns in the BPMN model which indicated the Constraints dependency. However, most of the canceling-related information was missing in text-based requirements as only normal event flows were described. The BPMN model, on the other hand, captured these exceptional events comprehensively and helped the BPMN+Text group to find more Constraints dependencies.

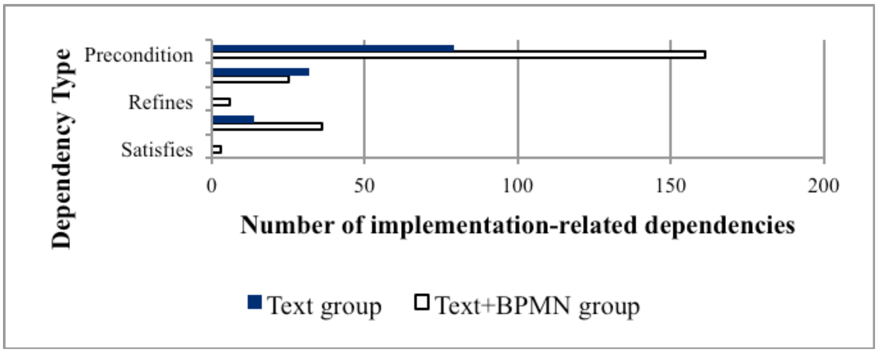


Fig. 6. Implementation-related dependency distribution

Q2. Are additional dependencies found in Question Q1 actually useful in change impact analysis and how?

We grouped thirty three change requests from PVS V3 into nine categories according to their intent and descriptions and compared the number of requirements impacted by the nine categories as reported by each of the participant groups. The number of impacted requirements for two request categories was the same for both groups and they were discarded from further analysis. The numbers of impacted requirements found for the remaining seven categories are shown in Fig. 7, The BPMN+Text group scored higher in each of the seven categories than the Text group since they found more dependencies and potentially more change impacts.

Although both groups identified all five implementation-related dependency types, the BPMN+Text group discovered 106 more implementation-related dependencies. Subsequently, this group used these extra dependencies to find forty seven impacted requirements, a significant portion of all the impacted requirements found (47%), suggesting that these dependencies were useful for change impact analysis.

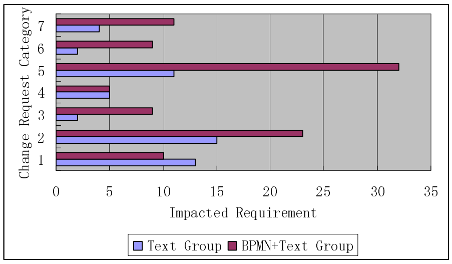


Fig. 7. Number of requirements impacted by change requests

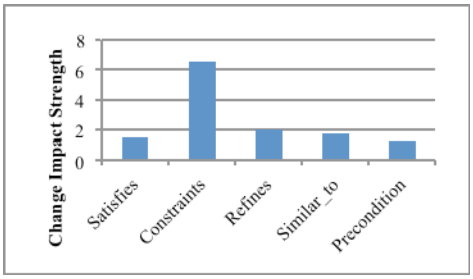


Fig. 8. Change impact strengths of dependency types

To show how the five dependency types are useful to change impact analysis, we used the concept of change impact strength – which we define as the average number of requirements impacted by one change request - to measure the capability of a dependency type to propagate changes between requirements. Fig. 8 shows the change impact strengths of the dependency types. “Constraints” has the highest change impact strength because this dependency type indicates that one requirement may interact with many other requirements to an extent like a crosscutting relation in aspect-oriented software development. Please note, from Fig. 8, that the change impact strengths of dependency types are different, and therefore it is important to pay attention to dependency types, which tend to have higher change impact strengths. Our findings provide a sweet spot for practitioners to analyze change impacts.

5 Discussions

Our case study emphasized realism and integrity of commercial confidentiality rather than the sample size. All our chosen participants were and/or had been involved in the development of the application at some stage of its lifecycle. However, this recruitment opened potential threats to the credibility of the case study. Thus, we investigated whether or not any relevant factors could lead to the bias of its results. In terms of the extent of domain knowledge about the application, one particular participant had been involved in the development of the project since inception. However, the other participant in the same group who had less experience with the application found more dependencies. The lack of domain knowledge did not seem to influence the results. We also observed that having specific knowledge and experience in requirements engineering or development determined greatly on what dependency types participants could find. One participant in the Text group, who has more requirements analysis experience, found more document-related dependencies while the other, who has more development experience, found more implementation-related dependencies. Both found similar numbers of dependencies. That means experience could influence the dependency types the participants focus on. On the other hand, we do believe that experience did influence a participant in learning the dependency types and later on the number of dependencies he/she found but the case study lacks statistical evidence to support it (i.e. only four participants).

There was no evidence of learning effect in this case study because both groups discovered dependency in one round. The Text group found dependencies in text-based requirements and the BPMN+Text group identified dependencies using both text-based requirements and the BPMN model together at the same time.

For external validity, we have only evaluated our approach on one project and acknowledge it as a limitation. Thus our evaluation results were constrained by the project. However, PVS appeared to be fairly complex and representative of a real application. The project offered us an opportunity to conduct an in-depth case analysis as an initial evaluation for our approach.

With regard to reliability, the participants reported that certain definitions of dependency types were vague when they applied the types. This might result in different understanding of the dependency types and affect the dependencies found. To assure the correctness and consistency of dependencies found, we conducted a follow-up interview with participants, discussed about disputed dependency types and achieved an agreement on the understanding of those disputed dependency types.

In related work, System Modeling Language (SysML) defines a number of requirement relationships for systems-of-systems and enterprise modeling [22]: derive, copy, verify, refine, satisfy and trace. The "trace" relationship is an abstract class for all the other relationships. "Refine" and "satisfy" are equivalent to our "Refines" and "Satisfies" relationships. The "verify" relationship links a test case to a requirement and it is not applicable to our dependency model which it is limited to the requirement stage and BPMN. Being software-centric, our current dependency model does not consider or explore SysML's "copy" and "derive" relationships. The former refers to one requirement's text property being a read-only copy of another requirement's for requirement reuse. The "derive" relationship relates one requirement to its derived requirements, which are at the next level of the system hierarchy. A future extension to our model and mapping is to incorporate all of SysML's requirement relationships to support systems-of-systems and enterprise architecture.

6 Conclusion and Future Work

From the requirements perspective, change impact analysis can be efficiently supported by dependency information. To facilitate dependency discovery in business process oriented applications, we proposed a mapping between a set of frequently occurring workflow patterns typical in BPMN, and a set of dependency types: Similar_to, Constraints, Precondition, Satisfies and Refines. Through this mapping, instances of these dependency types can be systematically derived from a typical BPM by identifying those workflow patterns from the BPMN-based model.

We conducted a case study in a real-world project to compare BPMN-and-text with text-only dependency discovery. In the former, case study participants discovered new dependencies in the requirements that were highly difficult to spot from the text-based requirements, suggesting an increased scope of change impacts. Through this case study, we also found the dependency types discovered through the mapping and BPMs very useful for change impact analysis and the change impact strengths of dependency types were different. These findings suggest that it is important to consider the nature of dependency during change impact analysis and inspire us in future work to explore the change impact abilities of different dependency types which may help improve the accuracy of change impact analysis.

Our study provides insights into applicability of business process modeling in requirement dependency discovery and change impact analysis for both research and practice. In the future, we will apply our approach to more industrial projects to validate its applicability.

Acknowledgement. This work is supported by National ICT Australia and funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. This work is also supported by the Commonwealth of Australia under the Australia-China Science and Research Fund, the National Natural Science Foundation of China under grant No. 60803023, No. 61073044, National Science and Technology Major Project 2010ZX01036-001-002, 2010ZX01037-001-002.

References

1. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution-The nineties view. In: Proc. 4th Intl. Software Metrics Symp., pp. 20–32. IEEE Computer Society Press (1997)
2. Arnold, R.S., Bohner, S.A.: Software Change Impact Analysis. Wiley-IEEE Computer Society (1996)
3. Bohner, S.A.: Software change impacts-an evolving perspective. In: Int. Conf. Softw. Maintenance (ICSM 2002), pp. 263–272. IEEE Computer Society Press (2002)
4. Fasolino, A.R., Visaggio, G.: Improving software comprehension through an automated dependency tracer. In: Proc. 7th Int. Wkshp. Program Comprehension, pp. 58–65. IEEE Computer Society Press (1999)
5. De Lucia, A., Fasano, F., Oliveto, R.: Traceability management for impact analysis. In: Frontiers of Software Maintenance (FoSM 2008), pp. 21–30. IEEE Press (2008)
6. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. 45(3), 515–526 (2006)
7. Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: A feasibility study of automated natural language requirements analysis in market-driven development. Requirements Engineering 7, 20–33 (2002)
8. Pohl, K.: PRO-ART: Enabling requirements pretraceability. In: 2nd Int. Conf. Requirements Eng., pp. 76–84. IEEE Computer Society Press (1996)

9. Li, J., Zhu, L., Jeffery, R., Liu, Y., Zhang, H., Wang, Q., Li, M.: An initial evaluation of requirements dependency types in change propagation analysis. In: 16th Int. Conf. Evaluation & Assessment in Softw. Eng., EASE 2012 (2012)
10. Larkin, J.H., Simon, H.A.: Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science 11, 65–100 (1987)
11. Object Management Group: Business Process Modeling Notation Specification v1.2 (2009)
12. Hassine, J., Rilling, J., Hewitt, J., Dssouli, R.: Change impact analysis for requirement evolution using use case maps. In: 8th Int. Wkshp. Principles of Software Evolution, pp. 81–90. IEEE Computer Society Press (2005)
13. von Knethen, A., Grund, M.: QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In: Int. Conf. Softw. Maintenance (ICSM 2003), pp. 246–255. IEEE Computer Society Press (2003)
14. Dahlstedt, Å., Persson, A.: Requirements Interdependencies: State of the Art and Future Challenges. In: Engineering and Managing Software Requirements, pp. 95–116. Springer, Berlin (2005)
15. Yan, Y.-Q., Li, S.-X., Liu, X.-M.: Quantitative Analysis for Requirements Evolution's Ripple-Effect. In: Int. Asia Conference on Informatics in Control, Automation and Robotics (CAR 2009), pp. 423–427. IEEE Computer Society Press (2009)
16. de la Vara González, J.L., Díaz, J.S.: Business process-driven requirements engineering: a goal-based approach. In: Pernici, B., Gulla, J.A. (eds.) Proc. CAiSE 2006 Workshops (vol. 1) - 8th Wkshp. Business Process Modeling, Development, and Support (BPMDS 2007), Trondheim, Norway (2007)
17. Cardoso, E.C.S., Almeida, J.P.A., Guizzardi, G.: Requirements engineering based on business process models: A case study. In: 13th Enterprise Distributed Object Computing Conference Workshops (EDOCW 2009), pp. 320–327. IEEE Computer Society Press (2009)
18. Mathisen, E., Ellingsen, K., Fallmyr, T.: Using business process modelling to reduce the effects of requirements changes in software projects. In: 2nd Int. Conf. Adaptive Science & Technology (ICAST 2009), pp. 14–19. IEEE Computer Society Press (2009)
19. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14, 5–51 (2003)
20. Wohed, P., Russell, N., ter Hofstede, A.H.M., Andersson, B., van der Aalst, W.M.P.: Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark. Information and Software Technology 51, 1187–1216 (2009)
21. Kous, K.: Comparative analysis versions of BPMN and its support with Control-flow patterns. In: MIPRO, 2010 Proc. 33rd Int. Convention, pp. 315–319. IEEE Computer Society Press (2010)
22. Object Management Group: OMG Systems Modeling Language (OMG SysML^{TM}) v1.2 (2011)

A Recommendation Algorithm to Capture End-Users' Tacit Knowledge

David Martinho[1,2] and António Rito Silva[1,2]

[1] ESW Software Engineering Group – INESC-ID
[2] IST – Technical University of Lisbon
{davidmartinho,rito.silva}@ist.utl.pt

Abstract. To capture knowledge workers' tacit knowledge, while they are performing their work, we consider the use of an ad-hoc workflow system that does not leverage on any predefined model. To avoid the noisy divergence of ad-hoc executions of business processes, we propose a recommendation algorithm that promotes convergent behavior through a goal-driven strategy based on data instead of activity control flow.

Keywords: Recommendations, Tacit Knowledge, Data-driven, Ad-hoc.

1 Introduction

To identify and automate their business processes, organizations need to rely on the empirical knowledge owned by their workers. This practical knowledge, usually known as tacit knowledge, which is difficult to transfer to others, allows workers to deal with particular business situations according to their experience and the organization's implicit set of business rules. Such tacit knowledge empowers workers to interact with one another using generic tools such as documents, spreadsheets, email, telephone and fax, resulting in interactions that are mainly ad-hoc and based on their experience, i.e. they are not driven by formal business process models automated in workflow tools.

Given the necessity of capturing tacit knowledge to build efficient business process models, knowledge elicitation techniques such as interviews or collaborative business process modeling environments are gaining popularity. However, these approaches are considered intrusive to knowledge workers, whose main function is, and should continue to be, the execution of business processes.

One way to support a non-intrusive elicitation of knowledge is through the means of a *capture-while-doing* strategy. The idea behind the *capture-while-doing* strategy is to support a flexible execution of business processes within an environment that simultaneously captures the flow of work in a structured way. Flexibility is enabled by an ad-hoc execution of business processes where there are no models prescribing the activities and control flow orchestrating them. Nevertheless, providing too much flexibility inevitably results in a combinatorial explosion of cases that, if left unmanaged, introduce noisy divergence that hinders standardization. In this paper we propose a recommendation algorithm that promotes convergent behavior through a goal-driven strategy based on data produced in previous business process instances instead of activity control flow.

2 Algorithm

In our previous work [1], developed within the context of the Processpedia project [2], we proposed a people-driven ad-hoc workflow tool to non-intrusively capture organizational behavior. To ensure this non-intrusiveness quality, the tool enables end-users to interact and exchange data objects using an interface similar to an email client system. Despite the interface similarities, interactions are supported by a more structured and semantically rich approach settled on the foundations of Language-Action Perspective (LAP). Therefore, without business process models to support their interactions, knowledge workers interact through the means of a request-response interaction pattern that allows them to request work from one another by providing the necessary input data objects, while expecting the data objects they need as a response (see Figure 1).

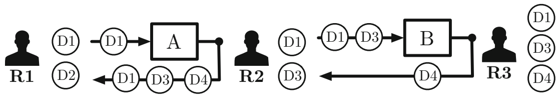


Fig. 1. The end-user playing role *R1*, owns two data objects named *D1* and *D2*, sends a request named *A* to another end-user playing role *R2*, only providing *D1* as input. The end-user playing *R1* is the initiator of request *A*. The end-user playing *R2*, after receiving the data object *D4* from the end-user playing *R3*, replies with an updated version of *D1*, *D3* and *D4* to the initial requestor. The end-user playing *R2* is simultaneously the executor of request *A*, and the initiator of request *B*.

In our approach, the sets of produced data objects represent the business process goals, which can be achieved in different ways: either by the same set of requests that execute in different orders, or by different sets of requests that produce the same set of data objects, i.e. that achieve the same process goal. Moreover, a business process can have different goals as their instances can produce different sets of data objects.

Given the absence of an underlying business process model, end-users must name requests and data objects to semantically differentiate them, inevitably resulting in a combinatorial explosion of names if the naming process is left unmanaged. To identify organizational behavior, we must promote convergence by avoiding the creation of terminologically different entities that are semantically equivalent. There are two main strategies to deal with this divergence problem: we either foster reutilization by providing the end-users with recommendations based on the past execution of their co-workers, or we can infer semantic equivalence between entities that may have different nomenclature and structure. We follow the former strategy because we are focusing on capturing and standardizing organizational behavior. Using the knowledge captured within previous ad-hoc executions, we are enabled to foster convergence by recommending request configurations (see Table 1), based on previous contextually similar executions.

The recommendation algorithm fosters convergence of behavior towards the achievement of previously attained process goals by suggesting request configurations used in the previous achievements of those goals. To do so, the algorithm

Table 1. An example of two request configurations, 1 and 2, under the same request name. The second column represents the number of process instances where the request configuration was applied, and the respective achieved process goals where it was used. The third and fourth column represents, respectively, the request input and output data objects. The fifth column represents the data objects available to the request initiator. The sixth column represents the data objects produced already in the process instance. Finally, the last column holds information concerning the organizational roles played by the request initiator.

Config	Occurrences	Input Context	Output Context	Request Context	Process Context	Initiator Context
1	$G_1(20)$,$G_2(10)$,$G_3(4)$	D_1	D_5,D_6	D_1,D_2	D_1,D_2	R_1,R_3
2	$G_1(2)$,$G_2(1)$,$G_3(8)$	D_1	D_5	D_1,D_2	D_1,D_2,D_3,D_4	R_2,R_3

deals with the immediate problem of ordering request configurations according to their contextual relevance to the end-user to whom the set of recommendations is addressed. However, in order to establish order within those set of requests, we established some quantitative metrics for comparison:

- **Goal Match:** reflects how the data objects already produced in the process instance fit within a particular process goal.
- **Request Configuration Fitness:** defines how fit is a request configuration, in the context of a process instance, an initiator, and its available data objects. It considers five fitness dimensions: process data, request data, input data, output data and initiator roles fitness (uses columns 3 to 6 in Table 1).
  - *Input Data Fitness:* how much the input data objects of the request configuration fit the set of data objects available to the initiator.
  - *Output Data Fitness:* how much the output data objects of the request configuration contribute to achieve the process goal.
  - *Request Data Fitness:* how much the request context data objects of the request configuration fit the data objects available to the initiator.
  - *Process Data Fitness:* how much the process context data objects of the request configuration fit the data objects already produced in the process instance.
  - *Initiator Fitness:* checks if the initiator plays at least one of the organizational roles in the initiator context of the request configuration.
- **Support:** measures the contribution of the request configuration to the achievement frequency of each process goal (uses column 2 in Table 1).

Algorithm 1 computes a set of request configuration recommendations, associating to each request configuration the metrics described above. A recommendation is therefore a tuple *(rc, goal-match, fitness, support)*, where *rc* is a request configuration to be recommended, and *goal-match*, *fitness* and *support* are the metrics that will be used to order the recommendations before they are presented to the end-user initiating a new request, in the context of the current request r. Given a process goal g, a process instance p, and the current request r, by applying the *data()* function to each respectively, we obtain the process goal data objects, the data objects already produced within the process instance, and the data objects available to the executor of the current request r.

Algorithm 1. Compute the set of request configuration recommendations

```
Require: Fitness Weights $\sigma_i, \sigma_o, \sigma_r, \sigma_p, \sigma_f$
Require: Process $P$
Require: Process instance $p \in P$
Require: Current request $r \in request(p)$
Ensure: $\sigma_i + \sigma_o + \sigma_r + \sigma_p + \sigma_f = 1$
  $recommendations \leftarrow \{\}$
  for all process goal $g \in goal(P)$ do
    $missing\text{-}data \leftarrow data(g) \setminus data(p)$
    $goal\text{-}match \leftarrow goal\text{-}match(g,\ p)$
    for all data object $d \in missing\text{-}data$ do
      for all distinct request configuration $rc$ where $d \in output\text{-}context(rc)$ do
        $input\text{-}fit \leftarrow \sigma_i \cdot data\text{-}fit(input\text{-}context(rc),\ data(r))$
        $output\text{-}fit \leftarrow \sigma_o \cdot data\text{-}fit(output\text{-}context(rc),\ missing\text{-}data)$
        $request\text{-}fit \leftarrow \sigma_r \cdot data\text{-}fit(request\text{-}context(rc),\ data(r))$
        $process\text{-}fit \leftarrow \sigma_p \cdot data\text{-}fit(process\text{-}context(rc),\ data(p))$
        $initiator\text{-}fit \leftarrow \sigma_f \cdot role\text{-}fit(initiator\text{-}context(rc),\ executor\text{-}roles(r))$
        $fitness \leftarrow input\text{-}fit + output\text{-}fit + request\text{-}fit + process\text{-}fit + initiator\text{-}fit$
        $support \leftarrow occurrences(rc, g)$
        $recommendations \leftarrow recommendations \cup (rc,\ goal\text{-}match,\ fitness,\ support)$
      end for
    end for
  end for
```

Given a process goal g and a process instance p, we compute the goal match by using the Tversky asymmetric similarity measure parametrized with $\alpha = 0$ and $\beta = 1$:

$$goal\text{-}match(g,p) = \frac{\mid data(g) \cap data(p) \mid}{\mid data(g) \cap data(p) \mid + \mid data(p) \setminus data(g) \mid}$$

The data fit function, used in Algorithm 1, is defined in Equation 1, illustrating how well a set of data objects A fits inside a set of data objects B.

$$data\text{-}fit(A,B) = \frac{1}{1 + \mid \mathrm{A} \setminus \mathrm{B} \mid} \tag{1}$$

Finally, we define role fit as the Kronecker Delta function established on the existence of an intersection between two sets of roles, as given by Equation 2.

$$role\text{-}fit(A,B) = \begin{cases} 1 & \mathrm{A} \cap \mathrm{B} \neq \emptyset \\ 0 & \mathrm{A} \cap \mathrm{B} = \emptyset \end{cases} \tag{2}$$

Although the *goal-match* is independent of the request configuration, it provides an important metric to identify how the current process data objects comply with process goals.

The computation of the overall fitness value is weighted by a vector of fitness weights $\langle \sigma_i, \sigma_o, \sigma_r, \sigma_p, \sigma_f \rangle$, respectively, for input, output, request, process and initiator roles fitness. To compute such request configuration fitness, the *input-fit, request-fit, process-fit* and *initiator-fit* are metrics that represent the capability the initiator has to initiate that request configuration, while the *output-fit* measures how closer the output of that request configuration will bring the process instance towards the considered process goal. Finally, the support of a given request configuration represents the number of times that a request configuration has contributed to achieve the considered process goal.

A last comment on how the algorithm orders recommendations: first recommendations are ordered by goal match since this metric reflects how much the process instance data is convergent with the process goal; then, for all the recommendations that have the same goal match, order is given by the request configuration overall fitness because it defines how able is the end-user to perform the request given its current context; finally, for the recommendations that also have the same fitness, they are ordered by the request recommendation support, which reflects the frequency a particular request configuration contributed to the achievement of the considered process goal.

3 Related Work

There are other approaches [3] that attempt to non-intrusively extract business process models from email logs. However, mining email logs is likely to produce incomplete business process models given its unstructured and noisy nature.

In [4], Vanderfeesten et al. use a Product Data Model (PDM) approach to drive the execution of business processes based on data dependencies. Further work [5] has focused on making decision process knowledge explicit by logging the interaction of knowledge workers with a simulated environment containing all data needed for the decision, with the intention of mining a PDM. In both cases, it is necessary to explicitly pre-define some of the data objects.

Other recent work [6] proposes recommendations based on document structure, using process structure and document's semantics to compute recommendations. Nevertheless, they focus on recommending different control flow sequences, based on pre-defined activities, to achieve the same goals.

In [7], Motahari and Nezad recommend next steps based on the automated discovery of annotated models, based on both structured and unstructured information on conversational and social interactions. Nevertheless, the similarity functions focus on tags and historical flow matching.

The work present in [8] proposes a recommendation service to guide end-users during process execution, providing recommendation on possible next steps. Although recommendations are also founded on similar past process executions, they focus on specific optimization goals, and like other works [9,10], are only concerned with activity control flow and disregard data objects.

Object-aware processes' work [11] also centers on the data produced, and avoids the pre-definition of activities. Our work relates by centering recommendations on data and capturing different ways to reproduce it, following a multi-dimensional context aware recommendation system approach [12].

4 Conclusions

In this paper we propose a goal-driven algorithm to compute request recommendations in people-driven ad-hoc processes. The algorithm calculates which requests are most suitable by leveraging on the state of the process instance and execution context. Such process instance state and execution context are defined using data objects and organizational roles. We defined the suitability of a recommendation using notions of goal match, fitness and support, respectively relating the recommended request to its contribution to the accomplishment of a goal, its contextual adequacy, and its execution frequency. Implicit feedback from the acceptance of the recommendation increases its support, whereas its goal match and fitness metrics are based on the availability and necessity of data objects, and on the organizational roles adequacy of their participants.

In the future we plan to conduct empirical evaluations of the algorithm in real organizations. Additionally, it would be interesting to extend the algorithm to also recommend the addressees of a request based on their organizational roles and accountabilities, as also establishing a credibility metric to promote the re-use of request configurations executed by experienced end-users.

References

1. Martinho, D., Rito Silva, A.: Non-intrusive Capture of Business Processes Using Social Software. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 207–218. Springer, Heidelberg (2012)
2. Rito Silva, A., Rosemann, M.: Processpedia: an ecological environment for BPM stakeholders' collaboration. Business Process Management Journal 18(1), 20–42 (2012)
3. van der Aalst, W.M.P., Nikolov, A.: Mining e-mail messages: Uncovering interaction patterns and processes using e-mail logs. International Journal of Intelligent Information Technologies 4(3), 27–45 (2008)
4. Vanderfeesten, I., Reijers, H., van der Aalst, W.M.P.: Product-based workflow support. Information Systems 36(2), 517–535 (2011)
5. Petrusel, R., Vanderfeesten, I., Dolean, C.C., Mican, D.: Making Decision Process Knowledge Explicit Using the Decision Data Model. In: Abramowicz, W. (ed.) BIS 2011. LNBIP, vol. 87, pp. 172–184. Springer, Heidelberg (2011)
6. Dorn, C., Marín, C.A., Mehandjiev, N., Dustdar, S.: Self-learning Predictor Aggregation for the Evolution of People-Driven Ad-Hoc Processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 215–230. Springer, Heidelberg (2011)
7. Motahari-Nezhad, H.R., Bartolini, C.: Next Best Step and Expert Recommendation for Collaborative Processes in IT Service Management. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 50–61. Springer, Heidelberg (2011)
8. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting Flexible Processes through Recommendations Based on History. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)

9. Haisjackl, C., Weber, B.: User Assistance during Process Execution - An Experimental Evaluation of Recommendation Strategies. In: zur Muehlen, M., Su, J. (eds.) BPM 2010 Workshops. LNBIP, vol. 66, pp. 134–145. Springer, Heidelberg (2011)
10. Barba, I., Weber, B., Del Valle, C.: Supporting the Optimized Execution of Business Processes through Recommendations. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 135–140. Springer, Heidelberg (2012)
11. Künzle, V., Weber, B., Reichert, M.: Object-aware business processes: Fundamental requirements and their support in existing approaches. International Journal of Information System Modeling and Design (IJISMD) 2(2), 19–46 (2011)
12. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Recommender Systems Handbook, pp. 217–253 (2011)

Defining Process Performance Indicators by Using Templates and Patterns*

Adela del–Río–Ortega, Manuel Resinas Arias de Reyna, Amador Durán Toro, and Antonio Ruiz–Cortés

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

Abstract. *Process Performance Indicators* (PPIs) are a key asset for the measurement of the achievement of strategic and operational goals in process–oriented organisations. Ideally, the definition of PPIs should not only be unambiguous, complete, and understandable to non–technical stakeholders, but also traceable to business processes and verifiable by means of automated analysis. In practice, PPIs are defined either informally in natural language, with its well–known problems, or at a very low level, or too formally, becoming thus hardly understandable to managers and users. In order to solve this problem, in this paper, a novel approach to improve the definition of PPIs using templates and ontology–based linguistic patterns is proposed. Its main benefits are that it is easy to learn, promotes reuse, reduces ambiguities and missing information, is understandable to all stakeholders and maintains traceability with the process model. Furthermore, since it relies on a formal ontology based on Description Logics, it is possible to perform automated analysis and infer knowledge regarding the relationships between PPI definitions and other process elements.

Keywords: Business Process Management, Process Performance Management, Key Performance Indicator, Process Performance Indicator, Templates, Patterns.

1 Introduction

Many companies are adopting a process–oriented approach in their business. In order to measure progress towards their business goals, it is important to evaluate the performance of their *business processes* (BPs) by means of the so–called *Process Performance Indicators* (PPIs), a particular case of *Key Performance Indicators* (KPIs) dedicated to BPs. For example, for the process depicted in Fig. 1, some PPIs could be defined based on metrics such as the average time of the Analyse RFC activity, the registered/approved RFC ratio, or the average delay of elevating a RFC to committee.

PPIs are recommended to satisfy the SMART criteria [1], i.e to be *Specific*, *Measurable*, *Achievable*, *Relevant* and *Time–bounded*, but also to be understandable, traced to the related BPs and automatically analysable [2,3,4]. A notation for PPI definition satisfying these requirements is still a challenge, mainly because of the conflict between understandability and automatic analysis. In practice, PPIs are defined either in

* This work has been partially supported by the European Commission (FEDER), Spanish Government under the CICYT project SETI (TIN2009–07366); and projects THEOS (TIC–5906) and ISABEL (P07–TIC–2533) funded by the Andalusian local Government.

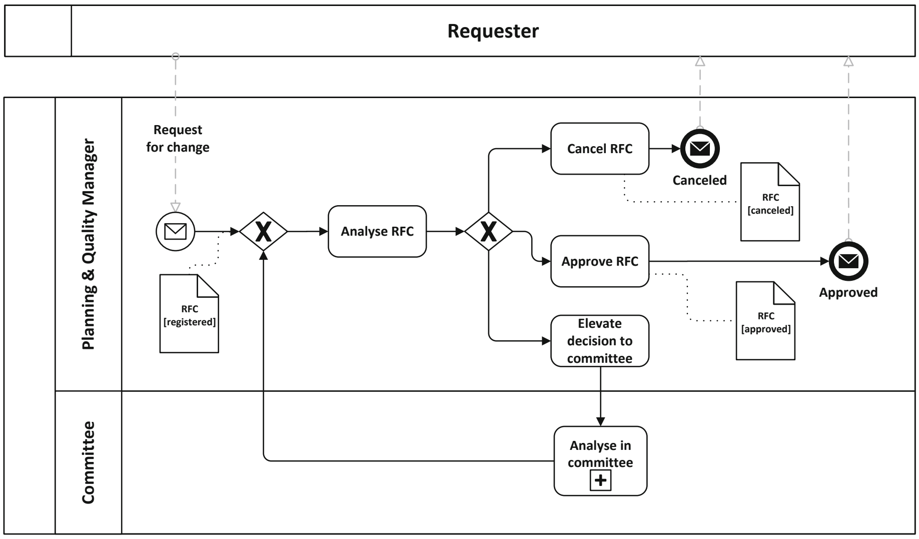


Fig. 1. Sample business process: Request for Change (RFC) management

(1) natural language, with its well–known problems of ambiguity and incompleteness; (2) at implementation level; or (3) too formally, becoming thus hardly understandable for managers and users.

In this paper we address this challenge and propose a novel approach to improve the definition of PPIs using templates and linguistic patterns (L–patterns, i.e. very used sentences in natural language that can be reused by parametrisation), which have been successfully applied in the areas of Requirements Engineering [5,6] and Service Level Agreements [7]. The proposed notation is formally supported by the PPINOT ontology [3], allowing their automated analysis using Description Logics.

2 PPI Template

Our proposal for PPI template, inspired by the requirements templates originally proposed in [5], is shown in Table 1 and an example is shown in Table 2. It has been designed in order to fulfil the SMART criteria [1] and is heavily based on the PPINOT ontology [3]. As commented in [5], using templates helps to organise the information in a structured form, reduces ambiguity, promotes reuse, and also serves as a guide to avoid missing relevant information. The notation used in the template is the following: words between "<" and ">" are placeholders for either literals (lower case) or L–patterns (upper case); words between "{" and "}" and separated by "|" are one–only options; words between "[" and "]" are optionals. The meaning of the template fields is the following:

– **Identifier** and **descriptive name**: unique PPI identifier, needed for traceability, and a self–descriptive name for the PPI.

Table 1. Template for PPI specification

PPI–<*ID*>	***<PPI descriptive name>***
Process	<*process ID the PPI is related to*>
Goals	<*strategic or operational goals the PPI is related to*>
Definition	The PPI is defined as { <*DurationMeasure*> | <*CountMeasure*> | <*ConditionMeasure*> | <*DataMeasure*> | <*DerivedMeasure*> | <*AggregatedMeasure*> } [expressed in <*unit of measure*>].
Target	The PPI value must { be {greater | lower} than [or equal to] <*bound*> | be between <*lower bound*> and <*upper bound*> [inclusive] | fulfil the following constraint: <*target constraint*> }
Scope	The process instances considered for this PPI are { the last <*n*> ones | those in the analysis period <*AP–x*> }
Source	<*source from which the PPI measure can be obtained*>
Responsible	{ <*role*> | <*department*> | <*organization*> | <*person*>}
Informed	{ <*role*> | <*department*> | <*organization*> | <*person*>}
Comments	<*additional comments about the PPI*>

- **Process** and **goals**: traces to the process for which the PPI is defined and to the strategic or operational goals the PPI is related to (*Relevant* SMART criteria).
- **Definition**: kind of measure and units, if needed, the PPI is based on (*Specific* and *Measurable* SMART criteria). Corresponding measure L–patterns are described in next section.
- **Target**: target value of the PPI for achieving previously referenced goals (*Achievable* SMART criteria).
- **Scope**: number of process instances or analysis period considered for computing the PPI value (*Time–Bounded* SMART criteria). Due to space limitations, analysis period descriptions are not included in this paper (see [3,4] for more information).
- **Source of information**: source from where the required information to compute the PPI is gathered.
- **Responsible** and **Informed**: resources in charge of or interested in the PPI. They can be persons, roles, departments or organisations.
- **Comments**: any other relevant information that cannot be fitted in previous fields.

3 L–Patterns for PPI Specification

Following [5,6], L–patterns are integrated in the proposed PPI template because filling blanks in prewritten sentences is easier, faster and less error–prone than doing it from scratch. The six proposed L–patterns are described in this section.

Table 2. PPI specification example

PPI–001	**Average time of RFC analysis**
Process	Request for change (RFC)
Goals	• BG–002: Improve customer satisfaction • BG–014: Reduce RFC time–to–response
Definition	The PPI is defined as *the average of* Duration of Analyse RFC activity.
Target	The PPI value must be lower than or equal to 1 *working day*.
Scope	The process instances considered for this PPI are the last *100* ones.
Source	Event logs of BPMS.
Responsible	*Planning and quality manager*
Informed	*CIO*
Comments	*Most RFCs are created after 12:00.*

3.1 Duration Measure L–Pattern

In the PPI context, a duration can be defined as the difference between two events, considering as events not only BP event triggerings but also BP element transitions. Following the BPMN 2.0 specification [8], we consider *activities*, *pools* and *data objects* as elements; and *ready, active, withdrawn, completing, completed, failing, failed, terminating, terminated, compensating* and *compensated* as states (data object states are user–defined). Having said that, the *DurationMeasure* L–pattern can be defined as:

the duration between the time instants when <event$_1$> and when <event$_2$>

where *<event>* is defined as:

{ *<BP element> changes to state <BP state>* | *<BP event> is triggered* }

For example, in order to measure the duration of the Analyse RFC activity, the L–pattern can be instantiated as:

the duration between the time instants when RFC analysis activity changes to state active *and when* RFC analysis activity changes to state completed

3.2 Count Measure L–Pattern

A count measure for PPIs counts the number of times a specific *event*—as considered in previous section—happens. Therefore, its corresponding L–pattern is as simple as *the number of times <event>*, for example:

the number of times Analyse RFC activity changes to state completed

3.3 Condition Measure L–Pattern

A condition measure takes boolean values depending on either the state of a BP element or a condition specified on a data object. The two corresponding L–patterns are:

<BP element> { *is currently* | *has finished* } *in state <BP state>*

Data object <object> satisfies: *<condition on object properties>*

For example:

Activity Analyse in committee *is currently in state* active

Data object RFC *satisfies*: priority = high

3.4 Data Measure L–Pattern

A data measure takes the value of a specific property of a data object. The L–pattern is as simple as: *the value of <property> of <object>*. For example, assuming the RFC data object has a property indicating the affected departments:

the value of affected departments *of* RFC.

3.5 Derived Measure L–Pattern

A derived measure is a function defined over other measures expressed using some of the previous L-patterns. For the sake of simplicity, they are referred to by means of a symbolic name. In this case, the L–pattern includes the expression of the function and a mapping from function variables to the measures of other measures:

the function <expression over $x_1 \ldots x_n$*>*, *where* { *<*x_i*> is <Measure*$_i$*>* }$_{i=1..n}$

For example, assuming two Measures such as Number of approved RFCs and Number of registered RFCs, a derived measure for the ratio of RFCs approved from registered could be defined as:

the function $\frac{a}{r} * 100$, *where* a *is* Number of approved RFCs and r *is* Number of registered RFCs

3.6 Aggregated Measure L–Pattern

In a similar way to derived measures, aggregated measures are defined over one of the previous measures by applying one aggregation function, i.e. sum, maximun, minimum, average, etc. The corresponding L–pattern is the following:

the { *sum* | *maximum* | *minimum* | *average* | … } of *<Measure>*

An example of the use of an aggregated measure L–pattern can be seen in the sample PPI definition in Table 2.

4 Conclusions and Future Work

As a major conclusion we can claim that it is possible to define PPIs with a notation that is easy to learn, promotes reuse, reduces ambiguities and avoids missing information, is understandable to all stakeholders, maintains traceability with the process model, and can be automatically analysed. The only price to pay is to restrict the employed sentences to the ones allowed by the underlying PPINOT ontology [3].

Some possible lines for future work can include adapting templates when more feedback from real scenarios is available, discovering more patterns, specially for the definition of resource–aware PPIs, and developing a tool to integrate it into the PPINOT tool, allowing thus the definition of PPIs through either the approach presented here or using our graphical notation, and their subsequent analysis, enabling also the automatic generation of documentation.

References

1. Shahin, A., Mahbod, M.A.: Prioritization of key performance indicators: An integration of analytical hierarchy process and goal setting. International Journal of Productivity and Performance Management 56, 226–240 (2007)
2. Franceschini, F., Galetto, M., Maisano, D.: Management by Measurement: Designing Key Indicators and Performance Measurement Systems. Springer (2007)
3. del-Río-Ortega, A., Resinas, M., Ruiz-Cortés, A.: Defining Process Performance Indicators: An Ontological Approach. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 555–572. Springer, Heidelberg (2010)
4. del Río-Ortega, A., Resinas, M., Ruiz-Cortés, A.: PPI definition and automated design-time analysis. Technical report, Applied Software Engineering Research Group (2012)
5. Durán, A., Bernárdez, B., Ruiz-Cortés, A., Toro, M.: A Requirements Elicitation Approach based in Templates and Patterns. In: Proc. Workshop de Engenharia de Requisitos (WER 1999), Buenos Aires, Argentina (1999)
6. Durán, A., Ruiz-Cortés, A., Corchuelo, R., Toro, M.: Supporting Requirements Verification using XSLT. In: Proc. IEEE Joint International Conference on Requirements Engineering, pp. 165–172 (2002)
7. Ruiz-Cortés, A., Martín-Díaz, O., Durán, A., Toro, M.: Improving the automatic procurement of web services using constraint programming. Int. J. Cooperative Inf. Syst. 14(4), 439–468 (2005)
8. OMG: Business Process Model and Notation (BPMN) Version 2.0 (January 2011)

Repairing Process Models to Reflect Reality

Dirk Fahland and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{d.fahland,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining techniques relate observed behavior (i.e., event logs) to modeled behavior (e.g., a BPMN model or a Petri net). Processes models can be discovered from event logs and conformance checking techniques can be used to detect and diagnose differences between observed and modeled behavior. Existing process mining techniques can only uncover these differences, but the actual repair of the model is left to the user and is not supported. In this paper we investigate the problem of *repairing a process model w.r.t. a log* such that the resulting model can replay the log (i.e., conforms to it) and is as similar as possible to the original model. To solve the problem, we use an existing conformance checker that aligns the runs of the given process model to the traces in the log. Based on this information, we decompose the log into several sublogs of non-fitting subtraces. For each sublog, a subprocess is derived that is then added to the original model at the appropriate location. The approach is implemented in the process mining toolkit ProM and has been validated on logs and models from Dutch municipalities.

Keywords: process mining, model repair, Petri nets, conformance checking.

1 Introduction

Process mining techniques aim to extract non-trivial and useful information from event logs [1,14]. Typically three basic types of process mining are considered: (a) *process discovery*, (b) *conformance checking*, and (c) *model enhancement* [1]. The first type of process mining is *process discovery*, i.e., automatically constructing a process model (e.g., a Petri net or a BPMN model) describing the causal dependencies between activities. The basic idea of control-flow discovery is very simple: given an event log containing a set of traces, automatically construct a suitable process model "describing the behavior" seen in the log. However, given the characteristics of real-life event logs, it is notoriously difficult to learn useful process models from such logs. The second type of process mining is *conformance checking* [2,4,5,8,9,13,18,19,21,23]. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. The conformance check could yield that the model *does not describe the process executions observed in reality*: activities in the model are skipped in the log, the log contains events not described by the model, or activities are executed in a different order than described by the model.

Here, the third type of process mining comes into play: to *enhance* an existing model to reflect reality [3]. In principle one could use process discovery to obtain a model that

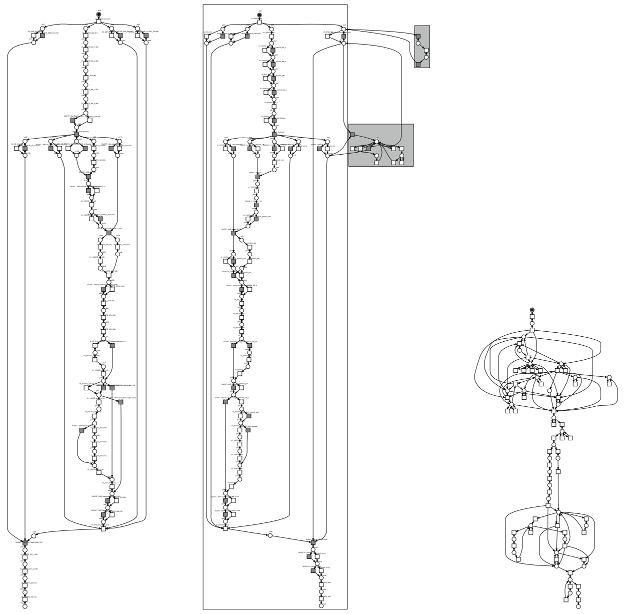


Fig. 1. Original model (left), model (middle) obtained by repairing the original model w.r.t. a given log, and model (right) obtained by rediscovering the process in a new model

describes reality. However, the discovered model is likely to bear no similarity with the original model, discarding any value the original model had, in particular if the original was created manually. A typical real-life example is the references process model of a Dutch municipality shown in Fig. 1(left); when rediscovering the actual process using logs from the municipality one would obtain the model in Fig. 1(right). A more promising approach is *repair*, that is change, the original model *so that the repaired model can replay the log and is as similar as possible to the original model*. This is the first paper to focus on model repair with respect to a given log.

The concrete problem addressed in this paper reads as follows. We assume a Petri net N (a model of a process) and a log L (being a multiset of observed cases of that process) to be given. N conforms to L if N can execute each case in L, i.e., N can replay L. If N cannot replay L, then we have to *change* N to a Petri net N' s.t. N' can replay L and N' is as similar to N as possible.

This problem is effectively a *continuum problem* between *confirming that N can replay L* and *discovering a new model N′ from L* in case N has nothing to do with L. The goal is to avoid the latter case as much as possible. We solve this problem in a compositional way: we identify subprocesses that have to be added in order repair N. In more detail, we first compute for each case $l \in L$ an *alignment* that describes at which parts, N and l deviate. Based on this alignment, we identify transitions of N that have to be skipped to replay l and which particular events of l could not be replayed on N. Moreover, we identify the *location* at which N should have had a transition to replay each of these events. We group sequences of non-replayable events at the same location to a *sublog* L' of L. For each sublog L', we construct a small subprocess N'

that can replay L' by using a process mining algorithm. We then insert N' in N at the location where each trace of L' should have occurred. By doing this for every sublog of non-replayable events, we obtain a repaired model that can replay L. Moreover, by the way we repair N, we preserve the structure of N giving process stakeholders useful insights into the way the process changed. We observed in experiments that even in case of significant deviations we could identify relatively few and reasonably structured subprocesses: adding these to the original model always required fewer changes to the original model than a complete rediscovery. Repairing the model of Fig. 1(left) in this way yields the model shown in Fig. 1(middle).

The remainder of this paper is structured as follows. Section 2 recalls basic notions on logs, Petri nets and alignments. Section 3 investigates the model repair problem in more detail. Section 4 presents a solution to model repair based on subprocesses. We report on experimental results in Sect. 5 and discuss related work in Sect. 6. Section 7 concludes the paper.

2 Preliminaries

This section recalls the basic notions on Petri nets and introduces notions such as event logs and alignments.

2.1 Event Logs

Event logs serve as the starting point for process mining. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed.

Definition 1 (Trace, Event Log). *Let Σ be a set of actions. A trace $l \in \Sigma^*$ is a sequence of actions. $L \in \mathbb{B}(\Sigma^*)$ is an event log, i.e., a multiset of traces.*

An event log is a *multiset* of traces because there can be multiple cases having the same trace. If the frequency of traces is irrelevant, we refer to a log as a set of traces $L = \{l_1, \ldots, l_n\}$. In this simple definition of an event log, an event is fully described by an action label. We abstract from extra information such as the resource (i.e., person or device) executing or initiating the activity and the timestamp of the event.

2.2 Petri Nets

We use *labeled* Petri nets to describe processes. We first introduce unlabeled nets and then lift these notions to their labeled variant.

Definition 2 (Petri net). *A Petri net (P, T, F) consists of a set P of* places*, a set T of* transitions *disjoint from P, and a set of arcs $F \subseteq (P \times T) \cup (T \times P)$. A* marking *$m$ of N assigns each place $p \in P$ a natural number $m(p)$ of* tokens*. A* net system *$N = (P, T, F, m_0, m_f)$ is a Petri net (P, T, F) with an* initial *marking m_0 and a final marking m_f.*

We write ${}^\bullet y := \{x \mid (x, y) \in F\}$ and $y^\bullet := \{x \mid (y, x) \in F\}$ for the *pre-* and the *post-set* of y, respectively. Fig. 2 shows a simple net system N with the initial marking $[p_1]$ and final marking $[p_6]$. N will serve as our running example.

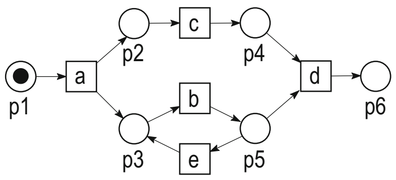


Fig. 2. A net system N

The semantics of a net system N are typically given by a set of *sequential runs*. A transition t of N is *enabled* at a marking m of N iff $m(p) \geq 1$, for all $p \in {}^\bullet t$. If t is enabled at m, then t may *occur* in the step $m \xrightarrow{t} m_t$ of N that reaches the *successor marking* m_t with $m_t(p) = m(p) - 1$ if $p \in {}^\bullet t \setminus t^\bullet$, $m_t(p) = m(p) + 1$ if $p \in t^\bullet \setminus {}^\bullet t$, and $m_t(p) = m(p)$ otherwise, for each place p of N. A sequential run of N is a sequence $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_k} m_f$ of steps $m_i \xrightarrow{t_{i+1}} m_{i+1}, i = 0, 1, 2, \ldots$ of N beginning in the initial marking m_0 and ending in the final marking m_f of N. The sequence $t_1 t_2 \ldots t_k$ is an *occurrence sequence* of N. For example, in the net N of Fig. 2 transitions a is enabled at the initial marking; abcd is a possible occurrence sequence of N.

The places and transitions of a Petri net can be *labeled* with names from an alphabet Σ. In particular, we assume label $\tau \in \Sigma$ denoting an *invisible* action. A *labeled Petri net* (P, T, F, ℓ) is a net (P, T, F) with a *labeling* function $\ell : P \cup T \rightarrow \Sigma$. A *labeled net system* $N = (P, T, F, \ell, m_0, m_f)$ is a labeled net (P, T, F, ℓ) with initial marking m_0 and final marking m_f. The semantics of a labeled net is the same as for an unlabeled net. Additionally, we can consider *labeled* occurrence sequences of N. Each occurrence sequence $\sigma = t_1 t_2 t_3 \ldots$ of N induces the *labeled* occurrence sequence $\ell(\sigma) = \ell(t_1)\ell(t_2)\ell(t_3)\ldots\ell(t_k)|_{\Sigma\setminus\{\tau\}}$ obtained by replacing each transition t_i by its label $\ell(t_i)$ and omitting all τ's from the result by projection onto $\Sigma \setminus \{\tau\}$. We say that N *can replay* a log L iff each $l \in L$ is a labeled occurrence sequence of N.

2.3 Aligning an Event Log to a Process Model

Conformance checking techniques investigate how well an event log $L \in \mathbb{B}(\Sigma^*)$ and a labeled net system $N = (P, T, F, \ell, m_0, m_f)$ fit together. The process model N may have been discovered through process mining or may have been made by hand. In any case, it is interesting to compare the observed example behavior in L and the potential behavior of N. In case the behavior in L is not possible according to N (L cannot replay N), we want to repair N.

In the following we recall a technique for identifying where L and N deviate, and hence where N has to be repaired. It will allow us to determine a *minimal set of changes* that are needed to replay L on N [2,5,4]. It essentially boils down to relate $l \in L$ to an occurrence sequence σ of N s.t. l and σ are as similar as possible. When putting l and σ next to each other, i.e., *aligning* σ and l, we will find (1) transitions in σ that are not part of l and (2) activities of l that are not part of σ [2].

For instance, a trace $l = \mathsf{accd}$ is similar to the occurrence sequence $\sigma = \mathsf{abcd}$ of the net of Figure 2 where trace l deviates from σ by skipping over b and having an additional c.

In order to repair N to fit trace l, N has to allow to skip over transitions of the first kind and has to be extended to execute activities of the second kind. In [5,4] an approach was presented that allows to automatically align a trace l to an occurrence sequence of N with a minimal number of deviations in an efficient way. All of this is based on the notion of an alignment and a cost function.

Definition 3 (Alignment). *Let* $N = (P, T, F, \ell, m_0)$ *be a labeled net system. Let* $l = a_1 a_2 \dots a_m$ *be a trace over* Σ. *A* move *is a pair* $(b, s) \in (\Sigma \cup \{\gg\}) \times (T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$. *An* alignment *of* l *to* N *is a sequence* $\alpha = (b_1, s_1)(b_2, s_2) \dots (b_k, s_k)$ *of moves, s.t.*

1. *the restriction of the first component to actions* Σ *is the trace* l, *i.e.,* $(b_1 b_2 \dots b_k)|_\Sigma = l$,
2. *the restriction of the second component to transitions* T, $(s_1 s_2 \dots s_k)|_T$, *is an occurrence sequence of* N, *and*
3. *transition labels and actions coincide (whenever both are defined), i.e., for all* $i = 1, \dots, k$, *if* $s_i \neq \gg, \ell(s_i) \neq \tau$, *and* $b_i \neq \gg$, *then* $\ell(s_i) = b_i$.

Move (b_i, s_i) is called (1) a *move on model* iff $b_i = \gg \wedge s_i \neq \gg$, (2) a *move on log* iff $b_i \neq \gg \wedge s_i = \gg$, and (3) a *synchronous move* iff $b_i \neq \gg \wedge s_i \neq \gg$.

For instance, for trace $l = \mathsf{accd}$ and the net of Figure 2, a possible alignment would be $(\mathsf{a}, \mathsf{a})(\mathsf{c}, \mathsf{c})(\gg, \mathsf{b})(\mathsf{c}, \gg)(\mathsf{d}, \mathsf{d})$.

Each trace usually has several (possibly infinitely many) alignments to N. We are typically interested in a best alignment, i.e., one that has as many synchronous moves as possible. One way to find a best alignment is to use a cost function on moves and to find an alignment with the least costs.

Definition 4 (Cost function, cost of an alignment). *Let* $\kappa : \Sigma \cup T \to \mathbb{N}$ *define for each transition and each action a positive cost* $\kappa(x) > 1$ *for all* $x \in \Sigma \cup T$. *The* cost *of a move* (b, s) *is* $\kappa(b, s) = 1$ *iff* $b \neq \gg \neq s$, $\kappa(b, s) = \kappa(s)$ *iff* $b = \gg$, *and* $\kappa(b, s) = \kappa(b)$ *iff* $s = \gg$. *The cost of an alignment* $\alpha = (b_1, s_1) \dots (b_k, s_k)$ *is* $\kappa(\alpha) = \sum_{i=1}^{k} \kappa(b_i, s_i)$.

In this paper, we abstract from concrete cost functions. However, we assume that *desirable* moves, i.e., synchronous moves (b, s) with $\ell(s) = b$ and invisible moves on model $(\gg, s)$ with $\ell(s) = \tau$, have low costs compared to *undesirable* moves such as moves on log $(b, \gg)$ and visible moves on model $(\gg, s)$ with $\ell(s) \neq \tau$.

Definition 5 (Best alignment). *Let* $N = (P, T, F, \ell, m_0)$ *be a labeled net system. Let* κ *be a cost function over moves of* N *and* Σ. *Let* l *be a trace over* Σ. *An alignment* α *(of* l *to* N*) is a* best *alignment (wrt.* κ*) iff for all alignments* α' *(of* l *to* N*) holds* $\kappa(\alpha') \geq \kappa(\alpha)$.

Note that a trace l can have several best alignments with the same cost. A best alignment α of a trace l can be found efficiently using an $\mathrm{A}^\star$-based search over the space of all prefixes of all alignments of l. The cost function κ thereby serves as a very efficient heuristics to prune the search space and guide the search to a best alignment. See [5,4] for details.

Using the notion of best alignment we can relate any trace $l \in L$ to an occurrence sequence of N.

3 Model Repair: The Problem

The model repair problem is to transform a model N that does not conform to a log L into a model N' that conforms to L. We review the state-of-the-art in conformance checking and investigate the model repair problem in more detail.

3.1 Conformance of a Process Model to a Log

Conformance checking can be done for various reasons. First of all, it may be used to audit processes to see whether reality conforms to some normative or descriptive model. Deviations may point to fraud, inefficiencies, and poorly designed or outdated procedures. Second, conformance checking can be used to evaluate the results of process discovery techniques. In fact, genetic process mining algorithms use conformance checking to select the candidate models used to create the next generation of models [17].

Numerous conformance measures have been developed in the past [2,4,5,8,21,9,13,18,19,23]. These can be categorized into four quality dimensions for comparing model and log: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that can explain the behavior seen in the log is the best model. This principle is known as Occam's Razor. A model is *precise* if it is not "underfitting", i.e., the model does not allow for "too much" behavior. A model is *general* if it is not "overfitting", i.e., the model is likely to be able to explain unseen cases [1,2].

The fitness of a model N to a log L can be computed using the alignments of Sect. 2.3 as the fraction of moves on log or move on model relative to all moves [2]. The aligned event log can also be used as a starting point to compute other conformance metrics such as precision and generalization.

3.2 Repairing a Process Model to Conform to a Log

Although there are many approaches to compute conformance and to diagnose deviations given a log L and model N, we are not aware of techniques to repair model N to conform to an event log L.

There are two "forces" guiding such repair. First of all, there is the need to improve conformance. Second, there is the desire to clearly relate the repaired model to the original model, i.e., repaired model and original model should be similar. Given metrics for conformance and closeness of models, we can measure the weighted sum or harmonic mean of both metrics to judge the quality of a repaired model. If the first force is weak (i.e., minimizing the distance is more important than improving the conformance), then the repaired model may remain unchanged. If the second force is weak (i.e., improving the conformance is more important than minimizing the distance), then repair can be seen as process discovery. In the latter case, the initial model is irrelevant and it is better to use conventional discovery techniques. Put differently, the model repair problem is positioned in a *spectrum* of two extremes:

keep. Keep the original model because it is of high value and non-conformance is within acceptable limits, e.g., 99.9% of all cases can be replayed.

discover. Model and log are effectively unrelated to each other, e.g., no case can be replayed and alignments find few or no synchronous moves.

Typically model repair is applied in settings in-between these two extremes. This creates a major challenge: How to identify which parts of a model shall be kept, and which parts of a model shall be considered as nonconformant to the log and hence changed, preferably automatically? The latter is a *local* process discovery problem which requires to balance the four quality dimensions of conformance as well.

3.3 Addressing Different Quality Dimensions

In this paper, we primarily focus on fitness which is often seen as the most important quality dimension for process models. A model that does not fit a given log (i.e., the observed behavior cannot be explained by the model) is repaired using the information available in the alignments.

Only for a fitting model precision can be addressed [18]; our particular technique for model repair will cater for precision as well. The two other criteria of generalization and simplicity may contradict these aims [1,14]. Generalization and precision can be balanced, for instance using a post-processing technique such as the one presented in [11].

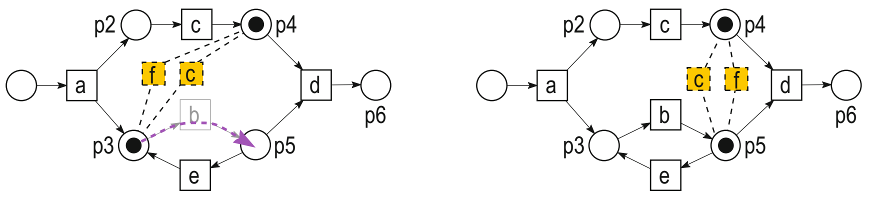
Similarity of the repaired model to the original model, as well as simplicity of the repaired model in general, is harder to achieve. It may require tradeoffs with respect to the other quality dimensions. For model repair basically the same experiences apply as for classical process discovery: while repairing, one should not be forced to extend the model to allow for all observed noisy behavior — it could result in overly complicated, spaghetti-like models. Therefore, we propose the following approach.

1. Given a log L and model N, determine the multiset L_f of fitting traces and the multiset L_n of non-fitting traces.
2. Split the multiset of non-fitting traces L_n into L_d and L_u. According to the domain expert the traces in L_d should fit the model, but do not. Traces in L_u could be considered as outliers/noise (according to the domain expert) and do not trigger repair actions.
3. Repair should be based on the multiset $L' = L_f \cup L_d$ of traces. L' should perfectly fit the repaired model N', but there may be many candidate models N'.
4. Return a repaired model N' that can be easily related back to the original model N, and in which changed parts are structurally simple.

The critical step of separating L_n into L_d and L_u does not require manual inspection of each case by a domain export. A number of standard preprocessing techniques can help to *filter* L_n: by (1) including in L_d only cases with particular start and end events, by (2) filtering infrequent events (that occur only rarely), and by (3) identifying events that occur out of order using *trace alignment* [7]; see [6] for a comprehensive use of preprocessing techniques in a case study.

a	c	f	c	≫	e	≫	d
a	c	≫	≫	b	e	b	d
[p2,p3]	[p4,p3]			[p4,p5]	[p4,p3]	[p4,p5]	[p6]

a	b	c	c	f	e	≫	d
a	b	c	≫	≫	e	b	d
[p2,p3]	[p2,p5]	[p4,p5]			[p4,p3]	[p4,p5]	[p6]



after replaying acfc   after replaying abccf

Fig. 3. Alignments of log L = {acfced, abccfed} to the net of Fig. 2

In the remainder, we assume L' to be given, i.e., outliers L_u of L are removed. If an event log is noisy and one includes also undesired traces L_u, it makes no sense to repair the model while enforcing a perfect fit as the resulting model will be spaghetti-like and not similar to the original model.

4 Repairing Processes by Adding Subprocesses

In the following, we present a solution to model repair. We first sketch a naive approach which completely repairs a model w.r.t. the quality dimension of *fitness* but scores poorly in terms of *precision*. We then define a more advanced approach that also caters for precision. Improvements w.r.t. *simplicity* are discussed at the end.

4.1 Naive Solution to Model Repair – Fitness

Alignments give rise to a naive solution to the model repair problem that we sketch in the following. It basically comprises to extend N with a τ-transition that skips over a transition t whenever there is a move on model $(\gg, t)$, and to extend N with a self-looping transition t with label a whenever there is a move on log $(a, \gg)$. This extension has to be done for all traces and all moves on log/model. The crucial part is to identify the locations of these extensions.

Figure 3 illustrates how the non-fitting log L = {acfced, abccfed} aligns to the net N of Fig. 2. The nets below each alignment illustrate the differences between log L of Fig. 3 and net N of Fig. 2. After replaying ac, the net is in marking [p4, p3] and the log requires to replay f which is not enabled in the net. Thus a log move (f, ≫) is added. Similarly, c is not enabled at this marking and log move (c, ≫) is added. Then e should occur, which requires to move the token from p3 to p5, i.e., a model move (≫, b). Correspondingly, the rest of the alignment, and the second alignment is computed. The third line of the alignment describes the marking that is reached in N by replaying this prefix of the alignment on N.

Using this information, the extension w.r.t. a move on model $(\gg, t)$ is trivial: we just have to create a new τ-labeled transition t^* that has the same pre- and post-places as t. The extension w.r.t. a move on log $(a, \gg)$ provides various options that only require that an a-labeled transition is enabled whenever this move on log occurs. We can use the

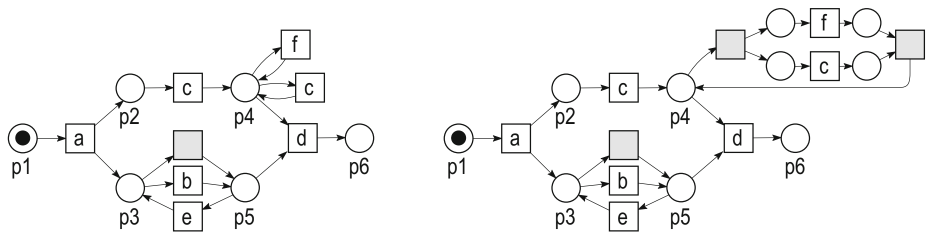


Fig. 4. Result of repairing the net of Fig. 2 w.r.t. the log of Fig. 4 by the naive approach (left) and by adding subprocess (right)

alignment to identify for each move on log $(a, \gg)$ in which marking m of N it should have occurred (the "enabling location" of this move). In principle, adding an a-labeled transition that consumes from the marked places of m and puts the tokens back immediately, repairs N w.r.t. to this move on log. However, we improve the extension by checking if two moves on log would overlap in their enabling locations. If this is the case, we only add one a-labeled transition that consumes from and produces on this overlap only.

Figure 4(left) shows how model N of Fig. 2 would be repaired w.r.t. the alignment of Fig. 3. The move on model $(\gg, \mathsf{b})$ requires to repair N by adding a τ transition that mimics b as shown in Fig. 4. The move on log $(\mathsf{c}, \gg)$ occurs at two different locations $\{\mathsf{p4}, \mathsf{p3}\}$ and $\{\mathsf{p4}, \mathsf{p5}\}$ in the different traces. They overlap on p4. Thus, we repair N w.r.t. $(\mathsf{c}, \gg)$ by adding a c-labeled transition that consumes from and produces on p4. Correspondingly for $(\mathsf{f}, \gg)$. The extended model that is shown in Fig. 4(left) can replay log L of Fig. 3 without any problems.

4.2 Identify Subprocesses – Precision

The downside of the naive solution to model repair is that the repaired model has low precision. For a log L where a best alignment contains only few synchronous moves, i.e., N does not conform to L, many τ-transitions and self-loops are added. In fact, we observed in experiments that self-looping transitions were often added at the same location creating a "flower sub-process" of events $\Sigma' \subset \Sigma_L$ that locally permitted arbitrary sequences $(\Sigma')^*$ to occur.

In the following, we turn this observation into a structured approach to model repair. Instead of just recording for individual events $a \in \Sigma$ their enabling locations w.r.t. log moves, we now record enabling locations of *sequences of log moves*. Each maximal sequence of log moves (of the same alignment) that all occur at the same location is a *non-fitting sub-trace*. We group non-fitting subtraces at the same location Q into a non-fitting sublog L_Q of that location. We then *discover* from L_Q a *subprocess* $N(L_Q)$ that can replay L_Q by using a mining algorithm that guarantees perfect fitness of $N(L_Q)$ to L_Q. We ensure that $N(L_Q)$ has a unique start transition and a unique end transition. We then add subprocess $N(L_Q)$ to N and let the start transition of $N(L_Q)$ consumes from Q and let the end transition of $N(L_Q)$ produce on Q, i.e., the subprocess models a structured loop that starts and ends at Q.

Figure 4(right) illustrates this idea. The model depicted is the result of repairing N of Fig. 2 by adding subprocesses as described by the alignments of Fig. 3. We can identify two subtraces cf and fc that occur at the same sublocation p4. Applying process discovery on the sublog {cf, fc} yields the subprocess at the top right of Fig. 4(right) that puts c and f in parallel. The two grey-shaded silent transitions indicate the start and end of this subprocess.

4.3 Formal Definitions

The formal definitions read as follows. For the remainder of this paper, let N be a Petri net system, let L be a log. For each trace $l \in L$, assume an arbitrary but fixed best fitting alignment $\alpha(l)$ to be given. Let $\alpha(L) = \{\alpha(l) \mid l \in L\}$ be the set of all alignments of the traces in L to N.

Locations. Let $\alpha = (a_1, t_1) \dots (a_n, t_n)$ be an alignment w.r.t. $N = (P, T, F, m_0, m_f, \ell)$. For any move (a_i, t_i), let m_i be the marking of N that is reached by the occurrence sequence $t_1 \dots t_{i-1}|_T$ of N. For all $1 \le i \le n$, if $(a_i, t_i) = (a_i, \gg)$ is a log move, then the *enabling location* of $(a_i, \gg)$ is the set $loc(a_i, \gg) = \{p \in P \mid m_i(p) > 0\}$ of places that are marked in m_i. For example in Fig. 3, $loc(\mathsf{c}, \gg) = \{\mathsf{p4}, \mathsf{p3}\}$ in the first alignment and $loc(\mathsf{c}, \gg) = \{\mathsf{p4}, \mathsf{p5}\}$ in the second alignment.

It is easy to check that extending N with a new a-labeled transition t with ${}^\bullet t = loc(a_i, \gg) = t^\bullet$ turns the log move $(a_i, \gg)$ into synchronous move (a_i, t), i.e., repairs N w.r.t. $(a_i, \gg)$. We now lift this local repair for one log move to a repair for all alignments of a log to N.

Subtraces. Any two consecutive log moves have the same location as the marking of N does not change. We group these moves into a subtrace. A *maximal* sequence $\beta = (a_i, \gg) \dots (a_{i+k}, \gg)$ of consecutive log moves of α is a *subtrace of α at location Q* iff $loc(a_j, \gg) = loc(a_i, \gg) = Q, i \le j \le i + k$, and no longer sequence of log moves has this property. We write $loc(\beta) = loc(a_i, \gg)$ for the location of subtrace β. Let $\beta(L)$ be the set of all subtraces of all alignments $\alpha(L)$ of L to N.

For example, in Fig. 4, fc is a subtrace of the first alignment at location {p4, p3} and cf is a subtrace of the second alignment at location {p4, p5}. We could repair the net by adding two subprocesses, one that can replay fc at $Q_1 = \{\mathsf{p4}, \mathsf{p3}\}$ and one that can replay cf at $Q_2 = \{\mathsf{p4}, \mathsf{p5}\}$. However, we could instead just add one subprocess that can replay fc *and* cf at location $Q_1 \cap Q_2 = \{\mathsf{p4}\}$.

Formally, we say that Q is a sublocation of a subtrace $\beta = (a_1, \gg) \dots (a_k, \gg)$ iff $Q \subseteq loc(\beta)$. A *sublog* (L_Q, Q) of $\alpha(L)$ at location Q is a set of subtraces $L_Q \subseteq \beta(L)$ s.t. for all $\beta \in L_Q$, $\emptyset \neq Q \subseteq loc(\beta)$, that is, each trace in L_Q can start at sublocation Q of its first event.

Sublogs. The entire set of subtraces $\beta(L)$ can be partitioned into several sublogs of disjoint sublocation, though there are multiple ways of partitioning. We call a set $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ of sublogs of $\alpha(L)$ *complete* iff $L_{Q,1} \cup \dots \cup L_{Q,k} = \beta(L)$. While completeness is enough to repair N w.r.t. L, one may want to have as few sublogs at as few locations as possible, for instance, by merging two sublogs $(L_{Q,1}, Q_1)$ and $(L_{Q,2}, Q_2)$ to $(L_{Q,1} \cup L_{Q,2}, Q_1 \cap Q_2)$ if $Q_1 \cap Q_2 \neq \emptyset$. We call $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ *minimal* iff $Q_i \cap Q_j = \emptyset$ for all $1 \le i < j \le k$. There may be multiple minimal complete sets of sublogs of L. This allows us to configure the repair w.r.t. the locations and the contents of the different sublogs, yielding different repair options.

We now have all notions to formally define how to repair model N w.r.t. log L. For a complete set of sublogs of $\alpha(L)$, we can repair N w.r.t. $\alpha(L)$ by discovering for each sublog (L_Q, Q) a process model N_Q, adding N_Q to N and connecting the start- and end transition of N_Q to Q.

Definition 6 (Subprocess of a sublog). *Let L be a log, let N be a Petri net, let $\alpha(L)$ be an alignment of L to N, and let (L_Q, Q) be a sublog of $\alpha(L)$.*

Let $L_Q^+ = \{start\ a_1 \dots a_k\ end \mid (a_1, \gg) \dots (a_k, \gg) \in L_Q\}$ be the sequences of events described in L_Q extended by a start event and an end event ($start, end \notin \Sigma_L$).

Let $\mathcal{M}$ be a mining algorithm that returns for any log a fitting model (i.e., a Petri net that can replay the log). Let $N_Q = \mathcal{M}(L_Q^+)$. Then (N_Q, Q) is the subprocess *of L_Q.*

The mining algorithm $\mathcal{M}$ will produce transitions labeled with the events in L_Q^+ and a start transition $t_{start}^{N_Q}$ with label *start* and an end transition $t_{end}^{N_Q}$ with label *end*. In the following, we assume that ${}^\bullet t_{start}^{N_Q} = \emptyset$ and $t_{end}^{N_Q\,\bullet} = \emptyset$, i.e., that start and end transitions have no pre- or post-places. In case $\mathcal{M}$ produced pre- and post-places for start and end, these places can be safely removed without changing that N_Q can replay L_Q^+. When repairing N, we connect $t_{start}^{N_Q}$ and $t_{end}^{N_Q}$ to the location Q of the subprocess.

Definition 7 (Subprocess model repair). *Let L be a log, let N be a Petri net. Let $\alpha(L)$ be the alignments of the traces of L to N.*

Let $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ be a minimal and complete set of sublogs of $\alpha(L)$. The subprocess-repaired model *of N w.r.t. $\alpha(L)$ is the net N' that is obtained from N as follows.*

- *Add to N a fresh transition $t_\tau \notin T_N$ with ${}^\bullet t_\tau = {}^\bullet t$ and $t_\tau^\bullet = t^\bullet, \ell'(t_\tau) = \tau$ iff there exists an alignment $\alpha \in \alpha(L)$ containing a visible model move $(\gg, t)$, and*
- *For each sublog $(L_{Q,i}, Q_i), i = 1, \dots, k$, let $(N_{Q,i}, Q_i)$ be the subprocess of $L_{Q,i}$ s.t. $N_{Q,i}$ and N are disjoint (share no transitions or places). Extend N with $N_{Q,i}$ (add all places, transition and arcs of $N_{Q,i}$ to N) and add arcs $(q, t_{start}^{N_{Q,i}})$ and $(t_{end}^{N_{Q,i}}, q)$ for each $q \in Q_i$, and set labels $\ell'(t_{start}^{N_{Q,i}}) = \tau$ and $\ell'(t_{end}^{N_{Q,i}}) = \tau$.*

Theorem 1. *Let L be a log, let N be a Petri net. Let $\alpha(L)$ be the alignments of the traces of L to N and $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ be a minimal and complete set of sublogs of $\alpha(L)$. Let N' be a subprocess-repaired model of N w.r.t. these subtraces. Then each trace $l \in L$ is a labeled occurrence sequence of N', that is, N' can replay L.*

Proof (Sketch). The theorem holds from the observation that each alignment $\alpha = (a_1, t_1) \dots (a_n, t_n) \in \alpha(L)$ of L to N can be transformed into an alignment of L to N' having synchronous moves or model moves on invisible transitions only as follows.

Every move on model $(\gg, t_i)$ of α w.r.t. N is replaced by a model move $(\gg, t_{i,\gg})$ w.r.t. N' on the new invisible transition $t_{i,\gg}, \ell(t_{i,\gg}) = \tau$ which allows to skip over t_i.

Every move on log $(a, \gg)$ w.r.t. N is part of a subtrace $\beta = (a_1, \gg) \dots (a_k, \gg)$ of a sublog $(L_{Q,i}, Q_i)$. By adding the subprocess $N_{Q,i}$ at location Q_i, the subtrace β is replayed by a sequence $(\gg, t_{start}^{N_{Q,i}})(a_1, t_1) \dots (a_k, t_k)(\gg, t_{end}^{N_{Q,i}})$ of synchronous moves in the subprocess $N_{Q,i}$. Moves $(t_{start}^{N_{Q,i}}, \gg)$ and $(t_{end}^{N_{Q,i}}, \gg)$ are harmless because they are made silent by relabeling $t_{start}^{N_{Q,i}}$ and $t_{end}^{N_{Q,i}}$ with τ. $\square$

This theorem concludes the techniques for process model repair presented in this paper. Observe that original model N is preserved *entirely* as we only add *new transitions* and *new subprocesses*. By taking a best alignment $\alpha(L)$ of L to N, one ensures that number of new τ-transitions and the number of new subprocesses (or of new self-looping transitions) is minimal.

4.4 Improving Repair – Simplicity

The quality of the model repair step can be improved in some cases. According to Def. 7, each sublog $(L_{Q,i}, Q_i)$ is added as a subprocess $N_{Q,i}$ that consumes from and produces on the same set Q_i of places, i.e., the subprocess is a loop. If this loop is executed in each case of L only once, then $N_{Q,i}$ is also executed exactly once. Thus, N could be repaired by inserting $N_{Q,i}$ in sequence (rather than as a loop), by refining the places $Q_i = \{q_1, \ldots, q_k\}$ to places $\{q_1^-, \ldots, q_k^-\}$ and $\{q_1^+, \ldots, q_k^+\}$ with

1. $^\bullet q_j^- = {^\bullet q_j}, q_j^{-\bullet} = \{t_{start}^{N_{Q,i}}\}, j = 1 \ldots, k$, and
2. $q_j^{+\bullet} = q_j^\bullet, {^\bullet q_j^+} = \{t_{end}^{N_{Q,i}}\}, j = 1 \ldots, k$.

Also, the repaired model N' can structurally be simplified by removing those model elements which are no longer used. Consider for instance a transition t which is never executed because the alignment only includes moves on model $(\gg, t_\tau)$, where t_τ is the new transition to skip over t. In this case t is always bypassed by transition t_τ and t can be removed from N'.

5 Experimental Evaluation

The technique for model repair presented in this paper is implemented in the Process Mining Toolkit ProM 6 in the package, available from http://www.promtools.org/prom6/.

Uma provides a plugin *Repair Model* that takes as input a Petri net N, a log L, and a best-fitting alignment $\alpha(L)$ of L to N. The alignment can be computed in ProM 6 using the Conformance Checker of [2,5,4]. The *Repair Model* plugin repairs N by extending N with subprocesses as defined in Sect. 4.3 and Sect. 4.4. For this, it first replays each alignment on N, and identifies all subtraces. Then subtraces are grouped to sublogs at the same location. The resulting sublogs are merged if they share the same location in a greedy way (by merging sublogs with the largest overlap of places first), until the resulting set of sublogs is minimal (i.e., all locations are disjoint). Each sublog is then passed to the ILP miner [24] which guarantees to return a model that can replay the sublog. The returned model is then simplified according to [11] and added to N as a subprocess as defined in Def. 7.

We validated our implementation on real-life logs from a process that is shared by five Dutch municipalities. Figure 1(left) shows the reference base model that is used in several municipalities. However, each municipality runs the process differently as demanded by the "couleur locale". As a result, the process observed in each municipality substantially deviates from the base model. To validate our technique, we repaired the

Table 1. Results on model repair for 10 logs from Dutch municipalities

	log		deviations			subprocesses			change to original			
			moves on		per	#	added $\lvert T\rvert$		total $\lvert T\rvert$		similarity-dist.	
	traces	length	model	log	case		avg.	max.	add.	rem.	repair	discover
M1	434	1-51	3327	310	1-26	7	7	21	69	3	0.144	0.476
M2	286	1-72	1885	323	1-41	5	10	23	65	3	0.147	0.486
M3	481	2-82	3079	1058	1-49	10	13	37	151	3	0.199	0.542
M4	324	1-37	2667	192	2-21	8	7	13	71	4	0.139	0.541
M5	328	2-43	3107	342	2-25	6	9	24	60	3	0.143	0.540
M1^f	249	24-40	681	229	1-12	2	6	9	25	4	0.074	0.473
M2^f	180	23-70	516	240	1-41	2	12	21	37	5	0.103	0.539
M3^f	222	22-82	465	598	1-49	7	10	26	87	5	0.164	0.543
M4^f	239	15-37	1216	180	2-17	6	7	13	60	4	0.124	0.542
M5^f	328	13-43	1574	280	2-16	4	9	23	51	3	0.111	0.541

base model for each municipality based on the municipality's log. In the following, we report our findings.

We obtained 5 raw logs (M1-M5) from the municipalities' information systems. From these we created filtered logs (M1^f-M5^f) by removing all cases that clearly should not fit the base model, for instance because they lacked the start of the process or were incomplete (see Sect. 3 for the discussion). Table 1 shows the properties of these 10 logs (over 44 different event classes) discussed in the following. The table lists the number of traces, minimum and maximum length, and the properties of a best matching alignment of the log to the model of Fig. 1(left) as the total number of model moves, number of log moves and the minimum and maximum number of deviations (log move or model move) per case. None of the traces could be replayed on the base model, in some cases deviations were severe.

Repairing the base model of Fig. 1(left) w.r.t. the filtered log M1^f yields the model of Fig. 1(middle). Repairing the same model w.r.t. the raw log M1 results in the model shown in Fig. 5(left). Repairing the base model w.r.t. the filtered log M2^f yields the model of Fig. 5(right). In each case, model repair requires only several seconds; a best-fitting alignment (needed for repair) could be obtained in about a minute per log. We checked all resulting models for their capability to replay the respective log and could confirm complete fitness for all models.

Moreover, we could re-identify the original model as a sub-structure of the repaired model, making it easy to understand the made repairs in the context of the original model. The original model had 68 transitions, 59 places, and 152 arcs. Table 1 shows for each log the number of added subprocesses, the average and maximal number of transitions per subprocess, and the total number of added and of removed transitions in the entire process. We can see that in the worst case, M3, the number of transitions in the model is more than tripled. Nevertheless, this large number of changes is nicely structured in subprocesses: between 2 and 10 subprocesses were added per log, the largest subprocess had 37 transitions, the average subprocess had 6-13 transitions. We identified alternatives, concurrent actions, and loops in the different subprocesses. Yet,

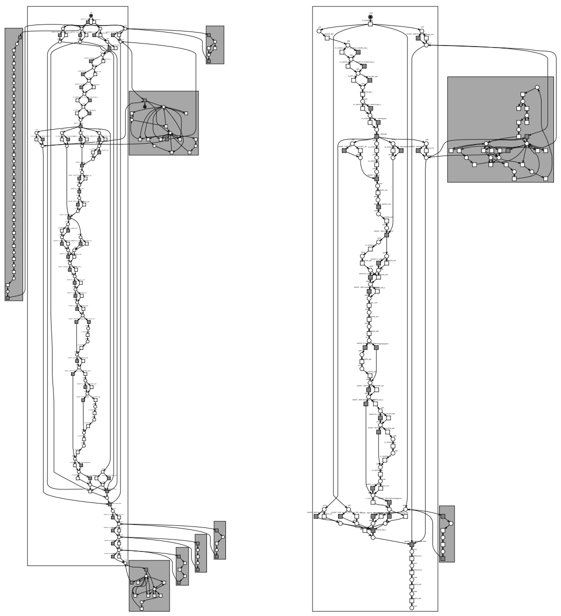


Fig. 5. Result of repairing Fig. 1(left) w.r.t. M1 (left) and M2^f (right)

simplification [11] ensured a simple structure in all subprocesses, i.e., graph complexity between 1.0 and 2.0. Model repair also allowed 25%-30% of the original transitions to be skipped by new τ-transitions; only few original transitions could be removed entirely.

To measure similarity, we computed the graph similarity distance [10] between repaired model and original model, and between a completely rediscovered model and the original model. The rediscovered model was obtained with the ILP miner [24] (ensuring fitness) and subsequently simplified by the technique of [11] using the same settings as for subprocess simplification. The similarity distance, roughly, indicates the fraction of the original model that has to be changed to obtain the repaired/rediscovered model, i.e., 0.0 means identical models. We observed that original model is significantly more similar to the repaired models (.074-.199) than the original model to the rediscovered models (.473-.543). This indicates that model repair indeed takes the original model structure by far more into account than model repair. The numbers also match the observations one can make when comparing Fig. 1(middle) to Fig. 1(right). Finally, the simpler models and fewer changes required for the filtered logs compared to the raw logs indicate that log preprocessing, as discussed in Sect. 3, has a significant impact on model repair.

6 Related Work

The model repair technique presented in this paper largely relates to two research streams: conformance checking of models and changing models to reach a particular aim.

Various conformance checking techniques that relate a given model to an event log have been developed in the past. Their main aim is to quantify *fitness*, i.e., how much the model can replay the log, and if possible to highlight deviations where possible [2,4,5,8,21]. The more recent technique of [2,4] uses alignments to relate log traces to model executions which is a prerequisite for the repair approach presented in this paper. Besides fitness, other metrics [2,9,13,18,19,23] (*precision*, *generalization*, and *simplicity*) are used to describe how good a model represents reality. Precision and generalization are currently considered in our approach only as a side-effect and not a leading factor for model repair. Incorporating these measure into model repair is future work. Simplicity is considered in our approach in the sense that changes should be as tractable as possible, which we could validate experimentally.

A different approach to enforcing similarity of repaired model to original model could be model transformation incorporating an edit distance. The work in [15] describes similarity of process model variants based on edit distance. Another approach to model repair is presented in [12] to find for a given model a most similar sound model (using local mutations). [16] considers repairing incorrect service models based on an edit distance. These approaches do not take the behavior in reality into account. Other approaches to adjust a model to reality, adapt the model at runtime [22,20], i.e., an individual model is created for each process execution. This paper repairs a model for multiple past executions recorded in a log. The approach of [11] uses observed behavior to structurally simplify a given model obtained in process discovery.

7 Conclusion

This paper addressed, for the first time, the problem of repairing a process model w.r.t. a given log. We proposed a repair technique that preserves the original model structure and introduces subprocesses into the model to permit to replay the given log on the repaired model. We validated our technique on real-life event logs and models and showed the approach is effective and the resulting model allows to understand the changes done to the original model for repair.

Our proposed technique of model repair covers the entire problem space of model repair between confirming conformance and complete rediscovery. In case of complete fitness, the model is not changed at all. In case of an entirely unfitting model (no synchronous move), the old model is effectively replaced by a rediscovered model. In case of partial fitness, only the non-fitting parts are rediscovered. This allows to apply our technique also in situations where the given model is understood as a *partial model* (created by hand) that is then completed using process discovery on available logs.

The technique can be configured. The cost-function influences the best-fitting alignment found, grouping of traces into sublogs and identifying sublocations for inserting new subprocesses allows for various solutions. Any process discovery algorithm can be used to discover subprocesses; the concrete choice depends on the concrete conformance notion addressed.

In our future work we would like to consider other conformance metrics such as generalization and precision. Moreover, in our current approach we abstract from extra logging information such as the resource executing or initiating the activity and the timestamp of the event. We would like to incorporate this information when repairing the model. For example, resource information can give valuable clues on for repair.

Acknowledgements. We thank M. Kunze and R.M. Dijkman for providing us with an implementation of the graph similarity distance and the anonymous reviewers for their fruitful suggestions. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n^o 257593 (ACSI).

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. WIREs Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time Prediction Based on Process Mining. Information Systems 36(2), 450–475 (2011)
4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking using Cost-Based Fitness Analysis. In: EDOC 2011. IEEE Computer Society (2011)
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards Robust Conformance Checking. In: zur Muehlen, M., Su, J. (eds.) BPM 2010 Workshops. LNBIP, vol. 66, pp. 122–133. Springer, Heidelberg (2011)
6. Bose, R.P.J.C., van der Aalst, W.M.P.: Analysis of Patient Treatment Procedures. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 165–166. Springer, Heidelberg (2012)
7. Bose, R.P.J.C., van der Aalst, W.M.P.: Process diagnostics using trace alignment: Opportunities, issues, and challenges. Inf. Syst. 37(2), 117–141 (2012)
8. Calders, T., Guenther, C., Pechenizkiy, M., Rozinat, A.: Using Minimum Description Length for Process Mining. In: SAC 2009, pp. 1451–1455. ACM Press (2009)
9. Cook, J.E., Wolf, A.L.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology 8(2), 147–176 (1999)
10. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
11. Fahland, D., van der Aalst, W.M.P.: Simplifying Mined Process Models: An Approach Based on Unfoldings. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 362–378. Springer, Heidelberg (2011)
12. Gambini, M., La Rosa, M., Migliorini, S., ter Hofstede, A.H.M.: Automated Error Correction of Business Process Models. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 148–165. Springer, Heidelberg (2011)
13. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009)
14. van der Aalst, W., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B.F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D.R., Gaaloul, W., van Geffen, F., Goel, S., Günther, C., Guzzo, A., Harmon, P., ter Hofstede, A., Hoogland, J., Ingvaldsen, J.E., Kato, K., Kuhn, R., Kumar, A., La Rosa, M., Maggi, F., Malerba, D., Mans, R.S., Manuel, A., McCreesh, M., Mello, P., Mendling, J., Montali, M., Motahari-Nezhad, H.R., zur Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Seguel Pérez,

H., Seguel Pérez, R., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., Wynn, M.: Process Mining Manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)

15. Li, C., Reichert, M., Wombacher, A.: Discovering Reference Models by Mining Process Variants Using a Heuristic Approach. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 344–362. Springer, Heidelberg (2009)
16. Lohmann, N.: Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer, Heidelberg (2008)
17. Medeiros, A., Weijters, A., Aalst, W.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
18. Muñoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer, Heidelberg (2010)
19. Muñoz-Gama, J., Carmona, J.: Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In: CIDM 2011. IEEE, Paris (2011)
20. Reichert, M., Dadam, P.: ADEPTflex-Supporting Dynamic Changes of Workflows Without Losing Control. JIIS 10(2), 93–129 (1998)
21. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
22. Sadiq, S., Sadiq, W., Orlowska, M.E.: Pockets of Flexibility in Workflow Specification. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 513–526. Springer, Heidelberg (2001)
23. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: CIDM 2011, pp. 148–155. IEEE (April 2011)
24. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae 94, 387–412 (2010)

FNet: An Index for Advanced Business Process Querying

Zhiqiang Yan, Remco Dijkman, and Paul Grefen

Technology University of Eindhoven
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
{z.yan,r.m.dijkman,p.w.p.j.grefen}@tue.nl

Abstract. Nowadays, more and more organizations describe their operations in terms of business processes. Consequently, it is common for organizations to have collections of hundreds or even thousands of business process models. This calls for techniques to quickly retrieve business process models that satisfy a given query. Some advanced techniques for querying a collection business process models exist. However, these techniques mainly focus on the expressive power of the query language, and performing an advanced business process query using these techniques can take considerable time. Consequently, querying a collection of models can take considerable time. To solve this problem, this paper proposes an efficient technique, using feature nets. Experiments show that on average the technique performs two orders of magnitude faster than existing techniques.

1 Introduction

Nowadays, business process management becomes more and more important in managing organizations. To increase the flexibility and controllability of the management of organizations, business processes are used to describe their operations. As a result, it is common to see collections of hundreds or even thousands of business process models. For example, the collection of SAP reference models consists of more than 600 business process models [15], and the collection of the reference models for Dutch Local Government contains a similar number of models [8]. As business process model collections increase in size, business process model repositories are developed to provide process-specific functions managing them. Querying a collection of business process models is one of these functions [23].

Querying a collection of business process models is done by providing a business process model fragment as a query. The querying technique then returns all business process models from the collection that contains that fragment. For example, if *query a* from Fig. 1 is provided to the querying technique, then the first three models should be returned, because they all contain that fragment; *graph 4* should be returned for *query b*.

This paper focuses on *advanced* business processes queries, which are queries that contain advanced query modeling elements, e.g., *query b* from Fig. 1. There

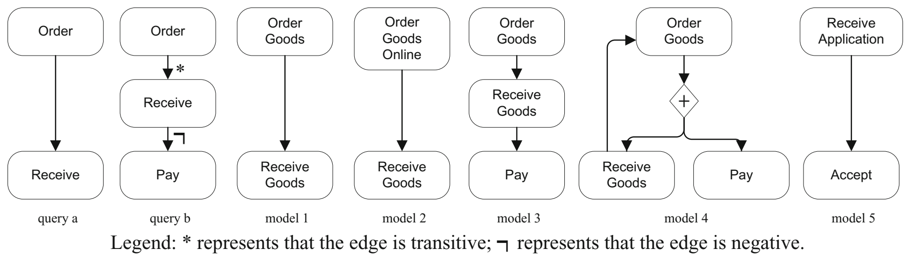


Fig. 1. Querying a Collection of Business Process Models

are four advanced query elements, which will be defined in section 3: wildcard nodes, transitive edges, negative edges, and negative transitive edges. To the best of our knowledge, three advanced query languages exist [1,3,6]. However, performing an advanced business process query using these techniques can take considerable time. For example, it on average takes 5s to run a query with a collection of 500 process models, using BPMN-Q [2,16]. While users of a search engine typically expect a response within milliseconds.

Therefore, our goal is to make advanced business process querying more efficient. To this end, this paper introduces the concept of feature net (FNet for short) and an efficient technique for querying based on feature nets. In this way, while both *advanced* query languages and techniques for efficiently performing *non-advanced* queries, the contribution of this paper is a technique for efficiently performing *advanced* queries.

The rest of the paper is organized as follows. Section 2 introduces the concept of process graph, which we use as the underlying formalism to define our efficient querying technique. Section 3 introduces query process graphs. Section 4 introduces the concept of feature of a process (query) graph. Features are the elements of an FNet. Section 5 presents the FNet and shows how it can be constructed for a collection of process models and how it can be used to make advanced querying more efficient. Section 6 evaluates the performance of the FNet. Section 7 introduces related work and Section 8 concludes the paper.

2 Process Graph

We define our querying technique on process graphs. A process graph is a graph-based representation of a process model [7,13]. The benefit of using a graph-based representation is that it can be used to represent the structure of existing (graph-based) business process modeling languages. In this way, techniques that are defined for process graphs can be generically applied to models that are constructed with multiple business process modeling languages. As an example, Fig. 2 shows the business process graphs for the models from Fig. 1. As shown in *graph 4* of Fig. 2, we assign each gateway node a unique label to represent its routing function, e.g., *'And-Split'* and *'Xor-Join'*.

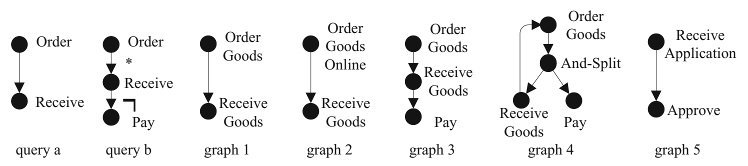


Fig. 2. Querying a Collection of Business Process Graphs

Definition 1 (Process Graph, Pre-set, Post-set). *Let $\mathcal{L}$ be a set of labels. A process graph describes a (business) process as a tuple (N, E, λ), in which:*

- *N is the set of nodes.*
- *$E \subseteq N \times N$ is the set of edges.*
- *$\lambda : N \to \mathcal{L}$ is an injective function that maps nodes to labels.*

Let $G = (N, E, \lambda)$ be a process graph and $n \in N$ be a node: $\bullet n = \{m|(m, n) \in E\}$ is the pre-set of n, while $n\bullet = \{m|(n, m) \in E\}$ is the post-set of n.

A path is a sequence of edges. For example, in *graph 4* of Fig. 2, there is a path from node *'Order Goods'* to node *'Receive Goods'*.

Definition 2 (Path). *Let $G = (N, E, \lambda)$ be a process graph and $n_1, n_s \in N$ be nodes of G. There is a path from n_1 to n_s if and only if there exists nodes $\{n_1, n_2, n_3, \ldots, n_s\} \subseteq N$ $(s > 1)$, and $\{(n_1, n_2), \ldots, (n_{s-1}, n_s)\} \subseteq E$.*

3 Query Process Graph

Being a fragment of a business process, a business process query can contain notational elements from the business process modeling notation in use. We refer to notational elements as basic elements. To make querying more powerful, advanced business process query languages [1,3,6] also use the following types of nodes and edges: wildcard node, transitive edge, negative edge, and neg-transitive edge. A *wildcard node* matches any node in a business process graph. A *transitive edge* matches a path from its source node to its target node. A *negative edge* matches if there is no edge between its source node and its target node. A *neg-transitive edge* is transitive and negative at the same time, matching if there is no path from its source node to its target node.

Definition 3 (Query Process Graph). *Let $\mathcal{L}$ be a set of labels. A query process graph is a process graph that can contain advanced elements besides basic ones, defined as a tuple $Q = (N, E, \lambda, \Theta, \theta)$, in which:*

- *N is the set of nodes.*
- *$E \subseteq N \times N$ is the set of edges.*
- *$\lambda : N \to \mathcal{L}$ is an injective function that maps a node to a label.*
- *$\Theta : N \to \{basic, wildcard\}$ is an injective function that determines whether a node is a basic or a wildcard query node.*

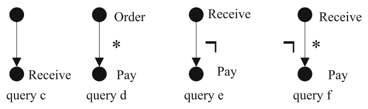


Fig. 3. Query Process Graphs with Advanced Nodes or Edges

– *$\theta : E \rightarrow \{basic, transitive, negative, neg\text{-}transitive\}$ is an injective function that determines whether an edge is a basic, a transitive, a negative, or a neg-transitive edge.*

For example, as shown in Fig. 3, *query c* has a wildcard node, which is denoted as a node without any label; *query d* has a transitive edge, $\theta(Order, Pay) = transitive$, which is represented as an edge with '$*$' as its label; *query e* has a negative edge, $\theta(Receive, Pay) = negative$, which is represented as an edge with '$\neg$' as its label; *query f* has a neg-transitive edge, $\theta(Receive, Pay) = neg\text{-}transitive$, which is represented as an edge with both '$\neg$' and '$*$' as its label. Besides these queries with advanced elements, a query process graph can also be a basic query like *query a* in Fig. 2.

Using the definition of a business process graph and a query graph, querying is done by finding business process graphs that match a given query graph.

Definition 4 (Querying). *A business process graph $G = (N_G, E_G, \lambda_G)$ matches a query graph $Q = (N_Q, E_Q, \lambda_Q, \Theta_Q, \theta_Q)$, if and only if there exists a mapping $M : N_Q \rightarrow N_G$, such that:*

– *for each $(n_Q, n_G) \in M$, either $\Theta(n_Q) = wildcard$ or $\omega(\lambda_Q(n_Q)) \subseteq \omega(\lambda_G(n_G))$, where $\omega(l)$ denotes the set of words that appear in a label l* [1];
– *if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = basic$ then $(M(n_Q), M(m_Q)) \in E_G$;*
– *if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = negative$ then $(M(n_Q), M(m_Q)) \notin E_G$;*
– *if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = transitive$ then there exists a path from $M(n_Q)$ to $M(m_Q)$ in G;*
– *if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = neg\text{-}transitive$ then there does not exist a path from $M(n_Q)$ to $M(m_Q)$ in G.*

For example, in Figure 2 and 3, node *'Order'*, is matching with nodes, *'Order Goods'* and *'Order Goods Online'*; *'Receive'* is matching with *'Receive Goods'* and *'Receive Application*; a wildcard node, *''*, is matching with all nodes; *'Pay'*, is matching with *'Pay'*; *query a*, '*Order*'$\rightarrow$'*Receive*', is matching with *graph 1,2,3*; *query c*, *''*$\rightarrow$*'Receive'*, is matching with *graph 1,2,3,4*; *query d*, *'Order'*$\xrightarrow{*}$*'Pay'*, is matching with *graph 3,4*; *query b*, *'Order'*$\rightarrow$*'Receive'*$\not\rightarrow$*'Pay'*, *query e*, *'Receive'*$\not\rightarrow$*'Pay'*, and *query f*, *'Receive'*$\overset{*}{\not\rightarrow}$*'Pay'*, are matching with *graph 4*.

[1] Label matching can be measured in a number of different ways [7]. For illustration purposes, we perform label matching by considering words of a label. This also allows users to query with only words under their concerns (like Google). It can be easily replaced by other metrics for label matching.

4 Features

To efficiently query a collection of business process graphs, we break both the process graphs and the query process graphs up into features. Features should be small and representative. Since they are small, they can be used for efficient processing. Since they are representative, results of a query feature are candidates of results of a query graph. After also defining an index on features in section 5, we can use them for fast query processing. This section presents how to perform feature-based querying.

Taking the criteria for selecting features (small and representative) into account, we only consider features based on the most common workflow patterns: sequence, split, join, and loop. Besides that we also consider single nodes as a feature, because we want to construct an index based on node labels. We name these features basic features.

Definition 5 (Basic Feature). *Let $G = (N, E, \lambda)$ be a process graph. A feature F of G is a subgraph of G. The size of a feature is the number of edges it contains. A feature is a:*

- *node feature consisting of node n, if and only if $N_F = \{n\} \wedge E_F = \emptyset$ (its size is 0);*
- *sequence feature of size $s-1$ consisting of nodes $\{n_1, n_2, n_3, \ldots, n_s\}$, if E_F is the minimal set containing $(n_1, n_2), (n_2, n_3), \ldots, (n_{s-1}, n_s)$, for $s \geq 2$;*
- *split feature of size s consisting of a split node n and a set of nodes $\{n_1, n_2, \ldots, n_s\}$, if and only if E_F is the minimal set containing $(n, n_1), (n, n_2), \ldots, (n, n_s)$, for $s \geq 2$;*
- *join feature of size s consisting of a join node n and a set of nodes $\{n_1, n_2, \ldots, n_s\}$, if and only if E_F is the minimal set containing $(n_1, n), (n_2, n), \ldots, (n_s, n)$, for $s \geq 2$;*
- *loop feature of size s consisting of nodes $\{n_1, n_2, \ldots, n_s\}$, if E_F is the minimal set containing $(n_1, n_2), \ldots, (n_{s-1}, n_s), and (n_s, n_1)$, for $s \geq 1$.*

For example, for *graph 4* in Fig. 2, the set of basic node features consists of nodes *'Order Goods', 'Receive Goods', 'And-Split', and 'Pay'*; the set of the basic sequence features of size 1 consists of sequences *'Order Goods'→'And-Split', 'And-Split'→'Receive Goods', 'And-Split'→'Pay', and 'Receive Goods'→'Order Goods*; the basic split feature set consists of the feature with split node *'And-Split'* and the set of nodes *'Receive Goods', 'Pay'*; and the basic loop feature set consists of the loop feature with three basic nodes *'Order Goods', 'And-Split', and 'Receive Goods'.*

More features are used other than sequences in this paper, which helps filter more graphs that are not matching with a given query [21]. For example, given a split query $a \to \{b, b\}$, a sequence $a \to b$ is filtered using split features.

A process graph contains only basic features. However, as explained in Section 3, a query process graph can also contain wildcard nodes, transitive edges, negative edges, and neg-transitive edges. Consequently, we need advanced features to be able to break up a query process graph into query features.

Definition 6 (Advanced Feature). *Let $Q = (N, E, \lambda, \Theta, \theta)$ be a query process graph. An advanced feature $F = (N_F, E_F, \lambda, \Theta, \theta)$ of Q is a subgraph of Q. The advanced feature is a:*

- *wildcard feature consisting of node n, if and only $N_F = \{n\}$ and $\Theta(n) =$ wildcard (its size is 0);*
- *transitive feature consisting of nodes $\{n_1, n_2\}$, if and only if $E_F = \{(n_1, n_2)\}$ and $\theta((n_1, n_2)) =$ transitive (its size is 1);*
- *negative feature consisting of nodes $\{n_1, n_2\}$, if and only if $E_F = \{(n_1, n_2)\}$ and $\theta((n_1, n_2)) =$ negative (its size is 1);*
- *neg-transitive feature consisting of nodes $\{n_1, n_2\}$, if and only if $E_F = \{(n_1, n_2)\}$ and $\theta((n_1, n_2)) =$ neg-transitive (its size is 1);*
- *basic feature, if and only if $\forall e \in E_F, \theta(e) =$ basic, and it is a feature according to Definition 5.*

The basic features in Definition 6 also includes basic features with wildcard nodes besides features in Definition 5, which can be used as a query feature. For example, in Fig. 3, *query c* is a basic query feature; *query d* is a transitive query feature; *query e* is a negative query feature; *query f* is a neg-transitive query feature. Definition 4 is used to measure whether an advanced feature is matching with a basic feature.

In order to optimally benefit from indexes that can be constructed for features, we allow features to be constructed hierarchically, such that we can use a multi-level (hierarchical) index. To this end, Definition 7 defines a hierarchical relation between features.

Definition 7 (Parent Feature, Child Feature). *If a feature f can generate feature cf by adding a single edge and at most one node, feature f is a direct parent feature of feature cf and feature cf is a direct child feature of feature f. It is denoted as $f \in DPFS(cf)$ or $cf \in DCFS(f)$, where $DPFS$ ($DCFS$) is a function that maps a feature to its direct parent (child) feature set. The parent and the child relation are the transitive closure of the direct parent and the direct child relation.*

For example, for *graph 4* in Figure 2, the direct child feature set of the node feature 'Receive Goods' is the set consisting of sequence features *'And-Split'→'Receive Goods'* and *'Receive Goods'→'Order Goods'*.

Feature based querying is done by first finding matching graphs for each of the features and subsequently determining whether the matching graphs also match the query as a whole. For a graph to match the query as a whole, it must meet three requirements. First, it must be a match for all of the basic query features. Second, the mappings that create the matches for each basic query feature, must not contradict each other (i.e.: if a node from the query graph is mapped to a node from the process graph for one feature, it must be mapped to the same node for another feature). Third, the advanced features must be matching with the graph for the given mappings according to Definition 4.

More precisely, feature-based querying is defined as follows.

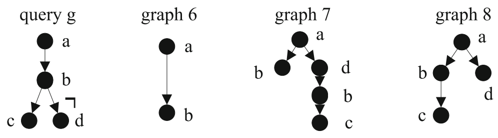


Fig. 4. Example of feature-based querying

Definition 8 (Feature-based Querying). *Given a business process graph g, a query graph qg and a decomposition of the query graph into a set of basic query features $\{f_1, f_2, \ldots, f_n\}$ and a set of advanced query features $\{af_1, af_2, \ldots, af_n\}$ (as defined in Definition 6). The business process graph matches the query graph, if and only if for each of the features there exists a corresponding mapping $M_1, M_2, \ldots, M_n$, such that:*

- *each mapping M_i creates a match of query f_i to g according to Definition 4;*
- *there exists the mapping M such that for each node $n \in N_Q$, for each basic feature f_i, if n is a node of f_i, then $M(n) = M_i(n)$ $(1 \leq i \leq n)$;*
- *for the mapping M, each advanced query feature af_i is matching with the process graph g according to Definition 4.*

For example, Figure 4 shows a query process graph and three process graphs. If we use only nodes and sequences of size 1 as features, then *query g* has four basic nodes features, *a, b, c* and *d*, the basic sequence features, $a \rightarrow b$ and $b \rightarrow c$, and one negative feature, $b \not\rightarrow d$. First, these basic query features are queried through indexes, and retrieved matching features are used to check which graphs contain matches for all basic query features. In this example, *graph 7* and *graph 8* have matching features for basic query features, while *graph 6* do not have and therefore is not a match for the query graph. Second, for each graph satisfying the first requirement, whether the same query node maps to the same node in the graph in all the mappings for features is checked. In this example, *graph 7* is also not a match. Although it is a match for all of the basic features, the only possible way in which to make both features match, causes a contradiction in the mappings. In particular the node labeled '*b*' from the query graph must be mapped to two different nodes in *graph 7* to match all basic features. Third, for each graph satisfying the first two requirements, whether advanced features are matching with the graph is checked based on the node mappings above. In this example, *graph 8* is a match for the query graph, because it does not have an edge between nodes *b* and *d*.

5 Feature Net

In order to speed up the querying operation, we must be able to quickly determine which process graphs in a collection contains *all* the features of a query graph without contradictions, because, according to Definition 8, those process graphs are the results for the query. In order to determine this, we construct an index of all basic features and transitive sequence features that process graphs in the collection contain. We call this index the feature net (FNet).

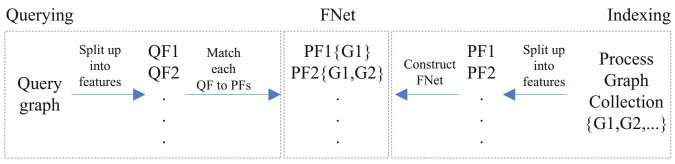


Fig. 5. Basic operations of an FNet

Figure 5 shows how this works. There are two operations that can be performed on the FNet: indexing and querying. When indexing a collection of process graphs $\{G_1, G_2, \ldots\}$, an FNet is constructed that consists of the process features, $PF_1, PF_2, \ldots$, and a mapping to the graphs that have those features. When querying, the query graph must be split up into query features, $QF_1, QF_2, \ldots$. For each of those features, the matching process features (if any) are then determined. Subsequently, those process graphs are returned that are a match for *all* of the features.

In this section we describe in more detail how an index of features, an FNet, can be constructed and how that FNet can be queried.

5.1 Constructing an FNet

An FNet consists of a directed graph, in which each node corresponds to a feature and edges relate each feature to its direct children. Node features are the smallest possible features and, as a consequence, are not the child of any feature. However, in the FNet we relate nodes to the words that are used in their labels, using an inverted index [12]. The inverted index does not contain stop-words, e.g., ‘a’, ‘an’, ‘the’, ‘one’, ... and uses lower case versions of the words. The FNet maps each feature to the process graphs of which it is a feature and each node of a feature to the process graph node that it represents. This mapping is required to decide which feature nodes also represent the same graph node, which must be checkable according to the second requirement of definition 8.

More precisely, an FNet is defined as follows.

Definition 9 (Feature Net (FNet)). *Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes. The feature net of $\mathcal{D}$, denoted $FNet(\mathcal{D})$, is a tuple (W, F, RW, RF, υ), in which:*

- $W = \bigcup_{G\in\mathcal{D}} \bigcup_{n\in N_G} \omega(\lambda(n))$ *is the set of all words in $\mathcal{D}$.*
- $F = \bigcup_{G\in\mathcal{D}} \Gamma(G)$ *is the set of all basic features of graphs in $\mathcal{D}$, where $\Gamma(G)$ returns the basic feature set of G.*
- $RW = \{(w, f) | w \in W, f \in \bigcup_{G\in\mathcal{D}} N_G, w \in \omega(\lambda(f))\}$ *is the relation between words and node features in $\mathcal{D}$ that contain that word.*
- $RF = \{((f_1, f_2)) | f_1, f_2 \in F \wedge f_1 \in DPFS(f_2)\}$ *is the direct parent-child relation between features as defined in Definition 7.*
- $\upsilon : (\bigcup_{f\in F} N_f) \to \mathbb{P}(\bigcup_{G\in\mathcal{D}} N_G)$ *is the function that maps each feature node to the graph nodes from which the feature is derived.*

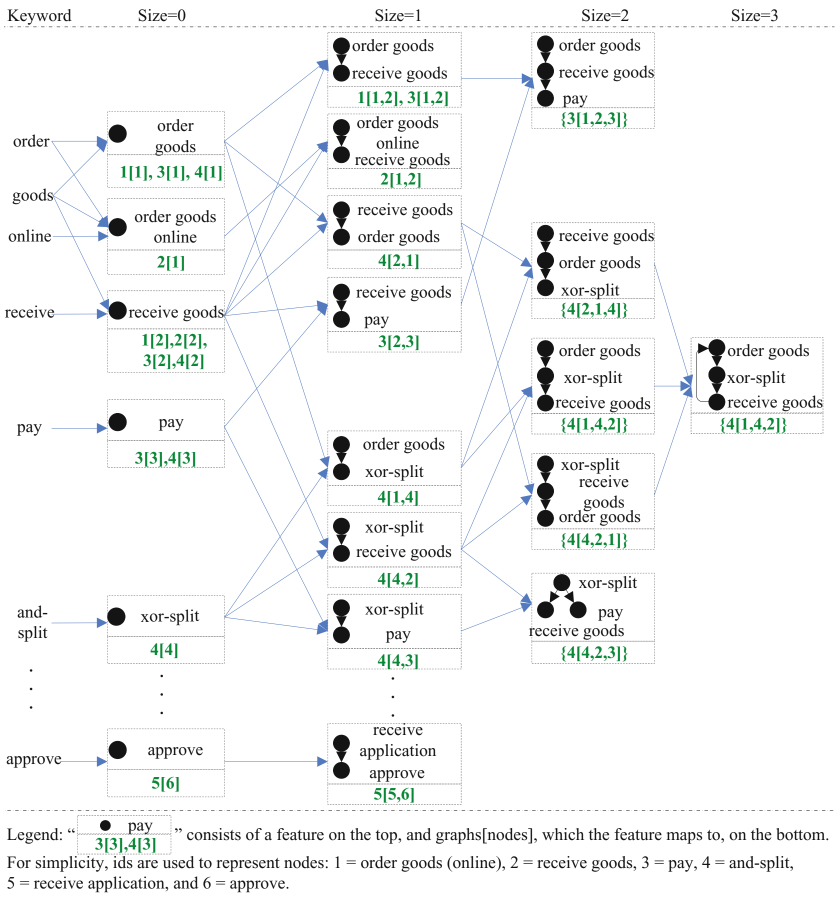


Fig. 6. Example of an FNet

Figure 6 shows an example of an FNet. The FNet is generated based on *graph 1-5* in Fig. 2. From left to right, five columns are listed, in which the first one lists the words (W) that appear in labels and the latter four list features (F) ordered by their sizes in the ascending order. Between columns, relationships (arrows) are drawn to connect a word to a node feature (RW) and connect a feature to its child features (RF). Each feature is associated with both graphs containing the feature and lists of nodes of which the feature consists (υ). For the sake of simplicity, not all of the features are shown in the figure.

5.2 Querying an FNet

To query a collection of process graphs for a given query graph through an FNet, four steps are performed: retrieving nodes, generating features, retrieving features, and checking graphs.

Firstly, for each query node, the words in its label are looked up, using the inverted index, to retrieve matching node features. A node feature is returned if all words in the query node's label return that node feature. If the query node is a wildcard node, then all the node features in the FNet are returned. The querying proceeds if and only if all the query nodes have matching node features in the FNet; otherwise there is no graph from the collection that satisfies the query. Taking *query a* and *query b* in Fig. 2 and the FNet in Figure 6 as an example, query node '*Order*' has two matching features, '*order goods*' and '*order goods online*'; query node '*Receive*' has one matching feature, '*receive goods*'; query node '*Pay*' has one matching feature, '*pay*'.

Secondly, features are generated from the query graph, by breaking up the query graph into subgraphs that are advanced features, as defined in Definition 6. In the example, *query a* contains a basic sequence feature, *'Order'*$\rightarrow$*'Receive'*; *query b* contains a transitive feature, *'Order'*$\overset{*}{\rightarrow}$*'Receive'*, and a negative feature, *'Receive'*$\overset{\neg}{\rightarrow}$*'Pay'*.

Thirdly, the FNet is used to retrieve matching features for each basic query graph feature, as defined in Definition 4. Given a query feature of size 0 (a query node), matching features were already retrieved using the inverted index in the first step. Given a query feature of size 1 (a query sequence), we already have the matching features of size 0 (the query nodes) that are this query feature's parents. For both these query nodes, we determine the sets of sequences that are their direct children. A sequence in the intersection of these two sets is a match of the query feature, if it connects the nodes in the same manner as the query feature (i.e.: has the same source and target). Given a query feature of size n ($n > 1$), which is a basic feature according to Definition 6, we already have the matching features of size $n - 1$ that are this query feature's parents. For both these parents, we determine the sets of features that are their direct children. A feature in the intersection of these sets is a match of the query feature, if it connects the same nodes as the query feature, connects these nodes in the same manner. Continuing with the example, *query a* has a basic sequence feature *'Order'*$\rightarrow$*'Receive'* has two matches in the FNet, *'order goods'*$\rightarrow$*'receive goods'*, and *'order goods online'*$\rightarrow$*'receive goods'*. The feature, *'receive goods'*$\rightarrow$*'order goods'*, is not a match since its source (target) node matches with the target (source) node of the query feature.

Fourthly, after getting the matching features for all basic query features, the mapping between features and graphs maintained in the FNet is used to retrieve and check matching graphs for the query graph, as defined in Definition 8. A graph is potentially matching with the query graph, if it has matching features for all query features. For such graphs, a check of contradictions is performed. If a node in a query graph matches more than one node in a graph, there is a contradiction. If there are advanced features in the query process graph, for each graph without contradictions, a check of advanced feature matching is performed. Graphs matching with all advanced features are returned as the final result set. Continue with the example, for *query a*, *graph 1,2,3* contain matching features for each basic query feature; no contradictions exist between feature mappings;

no advanced features are in *query a*; therefore, *graph 1,2,3* are matching with *query a*. For *query b*, *graph 3,4* contain matching features for each basic query feature; no contradictions exist between feature mappings; *graph 3 and 4* are matching with basic and transitive features of *query b*; *graph 4* is matching with the negative feature, while *graph 3* is not.; Therefore, *graph 3* is matching with *query b*.

6 Evaluation

This section shows how the use of the technique affects process querying in terms of performance and quality. We implemented the technique with Java. An FNet is constructed and manipulated in memory. We provided more details about the implementation and algorithms in a technical report [25].

The evaluation was performed on the collection of SAP reference models. This is a collection of 604 business process models (described as EPCs) that capture the business processes that are supported by SAP [15]. On average each process model in the collection contains 20.3 nodes with a minimum of 3 and a maximum 130 nodes. The average size of node labels is 3.8 words.

All the experiments were run on a laptop with an Intel Core2 Duo T7500 CPU (2.2GHz, 800MHz FSB, 4MB L2 cache), 4 GB DDR2 memory, the Windows Vista operating system and the SUN Java Virtual Machine version 1.6.

To evaluate the performance and to determine what influences the performance of the technique, two groups of queries were designed.

In the first group, as shown in Figure 7, five queries were adapted from the evaluation of BPMN-Q [1]. Query a, b, d, e, and g in the evaluation of [1] were selected. Query c and f were not selected, because they test specific types of elements in BPMN-Q that are simply considered nodes in this paper and are not handled differently from other types of nodes. Each node of the queries was labeled by one or two words from the original labels of these SAP reference models, such that the queries have matches in the collection of SAP reference models. More precisely, *query 1* is a sequence of size 1, composed of three basic nodes and two basic edge; *query 2* is a sequence of size 1, composed of a basic node, a wildcard node and a basic edge; *query 3* is a join of size 2, composed of three basic nodes and two transitive edges; *query 4* added two neg-transitive edges in *query 3* between the two nodes before joining; *query 5* is a loop of size 1, composed of a wildcard node and a transitive edge.

The second group were large models that were chosen to test the effect of query size on the performance of the FNet. The models in this group were designed by randomly selecting 5 from the collection of SAP reference models; then the models were adapted such that each query in this group contains as many advanced constructs as the corresponding query in the first group. *query 6* is composed of 9 basic nodes and 8 basic edges; *query 7* contains 1 wildcard nodes besides 22 basic nodes and 22 basic edges; *query 8* contains 4 transitive edges besides 22 basic nodes and 17 basic edges; *query 9* contains 2 neg-transitive edges besides 27 basic nodes and 26 basic edges; besides 20 basic nodes and 23

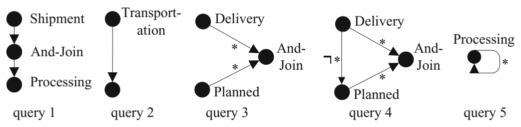


Fig. 7. First Group of Queries

basic edges, *query 10* contains 1 transitive edge with the same source and target node (a loop of size 1).

Table 1 shows the results of performing the queries on the collection of SAP reference models. The columns in the table show the execution time of each query and the average total time over the 5 queries in each group. The rows in the table show the features that are used to construct the FNet. In the first row process querying is performed based on node features of size 0 (Node(0)), sequence features of size 1 (Seq(1)), and loop features of size 1 (Loop(1)). In the second row process querying is performed based on features in the first row (1) plus sequence features of size 2 (Seq(2)) and loop features of size 2 (Loop(2)). Features of the rows below are described similarly.

In Table 1, we can see that on average a query in *Group 1* is performed in 0.03 second and a query in *Group 2* is performed in 0.06 second. The execution time of the first group is faster than the second group on average. This is because in the second group, there are more basic nodes and edges in the query models and more words in the node labels, therefore more feature comparisons are required. The second group has fewer matches than the first group. On average, for each query in the first group there are 10 hits from the collection of SAP reference models; while on average there are 1.6 positive results for each query in the second group.

Table 1. Execution Time

Group 1						
Features(size)	query1	query2	query3	query4	query5	T_{avg}
1:Node(0)+Seq(1)	0.04s	0.07s	0.01s	0.01s	0.02s	0.03s
2:1+Seq(2)	0.04s	0.08s	0.01s	0.01s	0.02s	0.03s
3:1+Split(2)	0.04s	0.07s	0.004s	0.01s	0.02s	0.03s
4:3+Split(3)	0.04s	0.09s	0.01s	0.01s	0.03s	0.03s
5:1+Join(2)	0.04s	0.07s	0.01s	0.01s	0.02s	0.03s
6:5+Join(3)	0.04s	0.07s	0.01s	0.01s	0.02s	0.03s
Group 2						
Features(size)	query6	query7	query8	query9	query10	T_{avg}
1:Node(0)+Seq(1)	0.002s	0.09s	0.03s	0.06s	0.12s	0.06s
2:1+Seq(2)	0.002s	0.10s	0.04s	0.06s	0.13s	0.07s
3:1+Split(2)	0.002s	0.10s	0.03s	0.06s	0.13s	0.06s
4:3+Split(3)	0.004s	0.19s	0.03s	0.06s	0.92s	0.24s
5:1+Join(2)	0.002s	0.10s	0.03s	0.06s	0.13s	0.06s
6:5+Join(3)	0.001s	0.09s	0.03s	0.06s	0.13s	0.06s

To better evaluate the performance of our technique, we compared it with the performance of BPMN-Q [2]. BPMN-Q on average takes 5s to perform a query with a collection of 500 process models (each model on average has 12 nodes) on a PC (2.8 GHz processors and 4GB memory); while an FNet on average takes 0.05s to perform a query with a collection of 604 process models (each model on average has 20.3 nodes) on a laptop (2.2 GHz processors and 4GB memory). The queries in this paper have similar characteristics as in [2] in terms of the number and type of advanced query elements. From the comparison we conclude that on average the technique performs two orders of magnitude faster than BPMN-Q.

To evaluate the scalability of an FNet, we generated a collection of 6040 synthetic process graphs and executed the first group of queries with the collection. On average, a synthetic process graph contains 20.3 nodes and 24.1 edges. The labels is the queries are adapted to make sure there are matching process graphs in the collection. The average execution time of a query is 0.10s, which is still within milliseconds. While using BPMN-Q [2], it on average takes about 6s to execute a query with 1000 models. We also find out that more execution time is required if a query has more hits in a collection or a query contains a wildcard node. More details about the synthetic process graph generator and the experiments with the synthetic collection are provided in a technical report [25].

To evaluate the quality of the technique, the first group of queries were used to run an experiment. The collection was developed in two steps. First, twenty SAP reference models were selected. We manually checked each pair of query and selected SAP reference model to see whether the selected SAP reference model is a positive or negative result for the query. We make sure that each query at least have one positive result within the two models. Second, for each query, three models were artificially made to check if the technique in this paper works well in terms of result quality. One of the model is a positive result for the query and the other two are negative results for the query. The experiment result shows that both precision and recall of the technique are 1.

7 Related Work

The work presented in this paper is related to business process querying, business process similarity search, general graph querying and general model querying.

Three groups of researchers have been working on advanced business process querying [2,3,6]. Awad [2] develops BPMN-Q, a language to query business processes, by extending the BPMN notation and implements BPMN-Q on top of relational databases. Beeri et al. [3] propose BP-QL, a language to query business processes modeled in BPEL. Choi et al. [6] propose IPM-EPDL, a query language for a proprietary process modeling notation based on XML. The difference between this paper and the above work is that this paper focuses on developing indexing techniques to make advanced business process querying more efficient. Jin et al. [9] develop efficient indexing techniques for basic business process querying, using sequences in the process models. The differences between this paper and [9] are as follows. Firstly, the technique in this paper supports advanced business process querying besides basic business process querying. Secondly, more features besides sequences are evaluated in this paper. Besides the work on querying business process models, there also exists work on querying executions of business processes [4,5].

The technique in this paper relates to the topic of business process similarity search. Process querying and similarity partly share the same techniques, e.g., for node matching. Similarity search is used to retrieve process models that are similar to a query model instead of exactly matching and thus uses different techniques to perform the search. The feature-based indexing has also been applied on business process similarity search to improve the efficiency of similarity search [22,24]. The main difference between this paper and [22,24] is that the metrics for matching (features of) process models are different because of the differences of process similarity search and querying, therefore retrieving features through the indexes are also different. In addition to that, features used in this paper are different, i.e., the loop feature is used for process querying, while the start and stop features are used for process similarity search. Other work on improving the efficiency of similarity search includes the work by Kunze et al. [11], who propose a metric that enables the use of an MTree index on process models. Qiao et al. [14] use clustering techniques to search process models efficiently.

General graph querying has been applied in various application domains, including fingerprint, DNA and chemical compound search. Willett et al. [20] describe a feature-based similarity search algorithm for searching in a chemical compound databases. ShaSha et al. [17] propose a path-based approach; Yan et al. [21] use discriminative frequent structures to index graphs. The main difference between the work that has been done in this area and the work in this paper is the different nature of business process graphs as compared to graphs in other domains. In particular, there is practically no restriction to the number of possible node labels in a business process graph and matching nodes do not necessarily have the identical labels. The selection of features is also different. In this paper common workflow patterns are used due to the characteristics of process models.

General model querying also relates to the topic of business process similarity search. Query languages [18,19] have been designed based on UML (e.g., class diagrams) instead of process modeling notations (e.g., BPMN and EPCs). Therefore, these query languages are for querying general software models instead of process models.

8 Conclusion

This paper presents a technique for improving the efficiency of advanced business process querying, which can be used to efficiently retrieve specific business models from the large sets of business process models that we encounter nowadays in practice. The technique works by breaking up a process model into small sub-models, which can also be used to build an index, called a feature net. In particular, the technique can also deal with advanced querying structures, such as paths of edges. Experiments show that a feature net can be used to retrieve results two orders of magnitude faster than querying techniques that is built on top of traditional RDBMS [2].

There are several directions for improving the feature net. Firstly, the technique in this paper focuses on tasks and relations between tasks. However, process models often contain other information that may be exploited, e.g., resources. This information can be integrated into an FNet by add more dimensions into features, e.g., resources and relations between resources and tasks. Querying on the basis of this information is left for future work. Secondly, label matching is based on matching identical words. However, equivalent tasks can be labeled differently, e.g., due to the use synonyms and different levels of verbosity. Therefore, we applied more advanced metrics for label matching that consider synonyms [7] and domain ontologies [10]. The integration of these advanced metrics into the technique described in this paper is also left for future work. Finally, comparing with the performance of the algorithm in this paper with that of general graph databases is also left for future work.

Acknowledgement. The research is supported by China Scholarship Council.

References

1. Awad, A.: BPMN-Q: A language to query business processes. In: Proceedings of EMISA 2007, Nanjing, China, pp. 115–128 (2007)
2. Awad, A., Sakr, S.: Querying Graph-Based Repositories of Business Process Models. In: Yoshikawa, M., Meng, X., Yumoto, T., Ma, Q., Sun, L., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 6193, pp. 33–44. Springer, Heidelberg (2010)
3. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: Proceedings of VLDB 2006, Seoul, Korea, pp. 343–354 (2006)
4. Beeri, C., Eyal, A., Milo, T., Pilberg, A.: Monitoring business processes with queries. In: Proceedings of VLDB 2007, Vienna, Austria, pp. 603–614 (2007)
5. Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H.R., Sakr, S.: A Query Language for Analyzing Business Processes Execution. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 281–297. Springer, Heidelberg (2011)
6. Choi, I., Kim, K., Jang, M.: An xml-based process repository and process query language for integrated process management. Knowledge and Process Management 14(4), 303–316 (2007)
7. Dijkman, R.M., Dumas, M., van Dongen, B., Uba, R., Mendling, J.: Similarity of Business Process Models: Metrics and Evaluation. Information Systems 36(2), 498–516 (2011)
8. Documentair structuurplan, `http://www.model-dsp.nl`
9. Jin, T., Wang, J., Wu, N., La Rosa, M., ter Hofstede, A.H.M.: Efficient and Accurate Retrieval of Business Process Models through Indexing. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 402–409. Springer, Heidelberg (2010)
10. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling, Ballarat, Victoria, Australia, pp. 71–80 (2007)

11. Kunze, M., Weidlich, M., Weske, M.: Behavioral Similarity – A Proper Metric. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 166–181. Springer, Heidelberg (2011)
12. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
13. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., Garcia-Banuelos, L.: APROMORE: an advanced process model repository. Expert Systems with Applications 38(6), 7029–7040 (2011)
14. Qiao, M., Akkiraju, R., Rembert, A.J.: Towards Efficient Business Process Clustering and Retrieval: Combining Language Modeling and Structure Matching. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 199–214. Springer, Heidelberg (2011)
15. Curran, T.A., Keller, G.: SAP R/3 Business Blueprint - Business Engineering mit den R/3-Referenzprozessen. Addison-Wesley, Bonn (1999)
16. Sakr, S., Awad, A.: A framework for querying graph-based business process models. In: Proceedings of WWW 2010, pp. 1297–1300 (2010)
17. ShaSha, D., Wang, J., Giugno, R.: Algorithmics and applications of tree and graph searching. In: Proceedings of the 21th PODS, pp. 39–53 (2002)
18. Stein, D., Hanenberg, S., Unland, R.: Query Models. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 98–112. Springer, Heidelberg (2004)
19. Störrle, H.: VMQL: A Visual Language for Ad-hoc Model Querying. Journal of Visual Languages and Computing 22, 3–29 (2011)
20. Willett, P., Barnard, J., Downs, G.: Chemical similarity searching. J. Chem. Inf. Comput. Sci. 38, 983–996 (1998)
21. Yan, X., Yu, P.S., Han, J.: Graph Indexing: A Frequent Structure-based Approach. In: Proceedings of the 2004 ACM SIGMOD, pp. 335–346 (2004)
22. Yan, Z., Dijkman, R., Grefen, P.: Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 60–77. Springer, Heidelberg (2010)
23. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Business Process Model Repositories - Framework and Survey. Information and Software Technology 54(4), 380–395 (2012)
24. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Fast Business Process Similarity Search. Distributed and Parallal Databases 30(2), 105–144 (2012)
25. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: FNet: An Index for Advanced Business Process Querying. Beta Working Paper-385, Technology University of Eindhoven, The Netherlands (2012)

Where Did I Misbehave? Diagnostic Information in Compliance Checking

Elham Ramezani, Dirk Fahland, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{e.ramezani,d.fahland,W.M.P.v.d.aalst}@tue.nl

Abstract. Compliance checking is gaining importance as today's organizations need to show that operational processes are executed in a controlled manner while satisfying predefined (legal) requirements. Deviations may be costly and expose the organization to severe risks. Compliance checking is of growing importance for the business process management and auditing communities. This paper presents a *comprehensive compliance checking approach based on Petri-net patterns and alignments*. 55 control flow oriented compliance rules, distributed over 15 categories, have been formalized in terms of Petri-net patterns describing the compliant behavior. To check compliance with respect to a rule, the event log describing the observed behavior is aligned with the corresponding pattern. The approach is *flexible* (easy to add new patterns), *robust* (the selected alignment between log and pattern is guaranteed to be optimal), and allows for both a *quantification of compliance* and *intuitive diagnostics* explaining deviations at the level of alignments. The approach can also handle resource-based and data-based compliance rules and is supported by *ProM* plug-ins. The applicability of the approach has been evaluated using various real-life event logs.

Keywords: compliance checking, process mining, conformance checking, Petri-nets.

1 Introduction

Business processes need to comply with regulations and laws set by both internal and external stakeholders. Failing to comply may be costly, therefore, organizations need to continuously check whether business processes are executed within the boundaries set by managers, governments, and other stakeholders. Deviations of the observed behavior from the specified behavior may point to fraud, malpractice, risks, and inefficiencies. Five types of compliance-related activities can be identified [23,30,19,28]:

- *compliance elicitation*: determine the constraints that need to be satisfied (i.e., rules defining the boundaries of compliant behavior),
- *compliance formalization*: formulate precisely the compliance requirements derived from laws and regulations in compliance elicitation,
- *compliance implementation*: implement and configure information systems such that they fulfil compliance requirements,
- *compliance checking*: investigate whether the constraints will be met (forward compliance checking) or have been met (backward compliance checking), and

– *compliance improvement*: modify the processes and systems based on the diagnostic information in order to improve compliance.

There are two basic types of conformance checking: (1) *forward compliance checking* aims to design and implement processes where conformant behavior is enforced and (2) *backward compliance checking* aims to detect and localize non-conformant behavior. This paper focuses on backward compliance checking based on event data.

Compliance checking is gaining importance because of the availability of event data and new legislations. Major corporate and accounting scandals including those affecting Enron, Tyco, Adelphia, Peregrine and WorldCom have fueled the interest in more rigorous auditing practices. Legislation such as the Sarbanes-Oxley (SOX) Act of 2002 and the Basel II Accord of 2004 was enacted as a reaction to such scandals. At the same time, new technologies are providing opportunities to systematically observe processes at a detailed level by recording all process relevant events.

Process mining techniques [1] offer a means to more rigorously check compliance and ascertain the validity and reliability of information about an organization's core processes. The core challenge is to compare the prescribed behavior (e.g., a process model or set of rules) to observed behavior (e.g., audit trails, workflow logs, transaction logs, message logs, and databases). For example, in [3] it is shown how constraints expressed in terms of Linear Temporal Logic (LTL) can be checked with respect to an event log. In [25] both LTL-based and SCIFF-based (i.e., abductive logic programming) approaches are used to check compliance with respect to a declarative process model and an event log. Dozens of approaches have been proposed to check conformance given a Petri-net and an event log [2,8,6,7,11,13,20,26,27,31,38]. Approaches such as in [31] replay the event log on the model while counting "missing" and "remaining" tokens. The former indicates observed, but disallowed behavior, and the latter indicate non-observed, but required behavior. State-of-the-art techniques in conformance checking retrieve this information by computing *optimal alignments* [2,8] between traces in the event log and "best fitting" paths in the model.

Existing approaches to backwards compliance checking have two main problems. First of all, the *elicitation of compliance rules is not supported well*. End users need to map compliance rules onto expressions in temporal logic or encode the rules into a Petri-net-like process model. Second, existing checking techniques can discover violations but *do not provide useful diagnostics*. While forward compliance checking techniques [10,18] employ pattern matching to highlight compliance violations in a model, such techniques are not applicable in backwards checking where not a model, but a log is given. Here, LTL-based checkers will classify a trace as non-compliant without providing detailed diagnostics and discard the remainder of the trace when the first deviation is detected.

To address these limitations we provide a *comprehensive collection of control flow related compliance rules*. We identify 55 rules distributed over 15 categories. These compliance rules are formalized in terms of Petri-net patterns. We apply the alignment technique developed in [2,8] to analyze if the process execution (log) has been compliant with the compliance rules (Petri-net patterns). If the observed behavior is consistent with the compliance rule, then the optimal alignment shows that all moves of the log can be mimicked by the corresponding Petri-net pattern and vice versa. If this is not possible because the compliance rule is violated, then the alignment shows the root cause of the deviation. This way, we are able to show *detailed diagnostics without false negatives* (non-conformant behavior remains undetected) and *false positives* (conformant behavior is classified as non-conforming because of an incorrect alignment of log and model/rule). The approach is *extendible*, i.e., to add a new type of rule, one just needs to add the Petri-net pattern to our repository. Moreover, as shown in this paper, our approach can be used to support resource-based and data-based compliance rules.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 explains the notion of alignments to relate observed and modeled behavior. Our compliance rule framework is introduced in Section 4. Although the primary focus of this paper is on control flow compliance rules, Section 5 illustrates that the approach also supports the other perspectives (e.g. resources and data). In Section 6 the approach is validated using a case study and the implementation in ProM is showcased. Section 7 concludes the paper.

2 Related Work

The importance of compliance management has been pointed out by various authors [5]. In [30] a life cycle is introduced to structure the process of compliance management. A comparative analysis over different compliance management solution frameworks is provided in [23].

Compliance management has gained wide interest from the Business Process Management (BPM) community. Compliance checking approaches can be mapped onto two main categories [22]:

– *Forward* compliance checking aims at ensuring compliant process executions. Processes can be constructed to be compliant [32] or verified whether they are compliant [24]. Alternatively, compliance requirements can be transformed into monitoring rules [12] or model annotations which then are used to enforce compliant process executions [17,39]. Diagnostic information is obtained by pattern matching [10,18].
– *Backward* compliance checking evaluates in hindsight whether process executions did comply to all compliance rules or when and where a particular rule was violated. A variety of conformance checking techniques have been proposed to quantify conformance and detect deviations based on an event log and process model (e.g., a Petri-net) [2,8,6,7,11,13,20,26,27,31,38]. Also approaches based on temporal logic [3,25] have been proposed to check compliance

In this paper we focus on backward compliance checking and assume an event log to be present. Compared to existing approaches we provide a comprehensive collection of compliance rules. Moreover, we focus on providing diagnostic information in backwards compliance checking.

3 Conformance Checking Based on Alignments

As will be shown in this paper, we provide a large repository of Petri-net patterns modeling typical compliance rules. These rules can be instantiated for a particular process, i.e., the abstract activities in the pattern are replaced by concrete activities also recorded in the event log. The log *complies* to the rule if each log trace is described by the Petri-net pattern. In case a trace is not described, we want to locate *where* the trace deviates from the pattern. This section recalls basic notions and a recent technique [2,8] for finding deviations between log traces and formal specification (a Petri-net).

An *event log* is a *multiset* of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) as a sequence of *events*. An event often refers to the *activity* executed. However, event logs may store additional information about events. For example, many process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order).

From a formal point of view a trace σ_L is a sequence over an alphabet Σ_L, i.e., $\sigma_L \in {\Sigma_L}^*$. An event log L is a multiset of traces, i.e., $L \in \mathbb{B}({\Sigma_L}^*)$. The alphabet Σ_L is typically the set of activity names. However, when including additional perspectives, the alphabet may be extended to also contain information about data and resources. For example, $(\mathit{prepare\ decision}, \mathit{start}, \mathit{John}, \mathit{gold}, \mathit{50\ euro}) \in \Sigma_L$ may refer to an event describing the start of activity "prepare decision" by John for a gold customer claiming 50 euro. The choice of Σ_L depends on the compliance rule that needs to be checked, e.g., for most control flow related rules it is sufficient to record the activity name.

A Petri-net pattern is essentially a specification prescribing compliant traces in a concise way. Technically, a *specification* $S \subseteq {\Sigma_S}^*$ is a finite set of traces over an alphabet Σ_S together with a mapping $\ell : \Sigma_S \to 2^{\Sigma_L} \cup \{\tau\}$ that relates each specification event in Σ_S to a set of log events in Σ_L or to τ. In this paper, S is the set of firing sequences of a Petri-net and ℓ is the function that labels each transition with an activity name. S and ℓ can be described by other formalisms such as temporal logics as well.

We use the alignment approach described in [2,8] to relate traces in the log (i.e., observed behavior) to traces of the specification (i.e., prescribed behavior). An optimal alignment of σ_L to S, roughly speaking, is a trace σ_S that is possible according to S and that is *as similar to* σ_L *as possible*. By comparing σ_L and σ_S, a business analyst gains an understanding on what has been done wrong in σ_L and what instead should have been done (to behave as shown in σ_S).

A given trace $\sigma_L \in L$ will be related to a trace $\Sigma_S \in S$ by pairing events in σ_L to events of Σ_S. Formally, a *move* (of σ_L and S) is a pair $(x, y) \in (\Sigma_L \cup \{\gg\}) \times (\Sigma_S \cup \{\gg\}) \setminus \{(\gg, \gg)\}$. For $x \in \Sigma_L, y \in \Sigma_S$, we call $(x, \gg)$ a *move on log*, $(\gg, y)$ a *move on specification* S, and if $x \in \ell(y)$, then (x, y) is a *synchronous move*.

An *alignment* of a trace $\sigma_L \in {\Sigma_L}^*$ to S is a sequence $\gamma = \langle (x_1, y_1) \ldots (x_n, y_n) \rangle$ of moves (of σ_L and S) such that the projection $x_1 \ldots x_n$ to Σ_L is the original trace σ_L, i.e., $\langle x_1 \ldots x_n \rangle|_{\Sigma_L} = \sigma_L$, and the projection $\langle y_1 \ldots y_n \rangle|_{\Sigma_S} = \sigma_S \in S$ is described by the specification.

For example, for a specification $S = \{\langle a, b, c, d \rangle, \langle a, c, b, d \rangle\}$ with $\ell(x) = \{x\}$ the trace $\sigma_L = \langle a, c, c, d \rangle$ has (among others), the following two alignments with events of σ_L shown at the top and events of S shown at the bottom: $\gamma_1 = \begin{array}{c|c|c|c|c} a & c & c & \gg & d \\ \hline a & c & \gg & b & d \end{array}$ and $\gamma_2 = \begin{array}{c|c|c|c|c|c} a & \gg & \gg & c & c & d \\ \hline a & c & b & \gg & \gg & d \end{array}$.

Both alignments yield the same specified trace $\sigma_S = \langle a, c, b, d\rangle \in S$. However, γ_1 is preferable over γ_2 as it maximizes the number of synchronous moves. The conformance checking problem in this setting is to find for a given trace σ_L and specification S an *optimal* alignment γ of σ_L to S s.t. no other alignment has fewer non-synchronous moves (move on log only or move on specification only). The technique of [2,8] finds such an optimal alignment using a cost-based approach: a cost-function κ assigns each move (x, y) a cost $\kappa(x, y)$ s.t. a synchronous move has cost 0 and all other types of moves have cost > 0. Then an A*-based search on the space of (all prefixes of) all alignments of σ_L to S is guaranteed to return an optimal alignment for σ_L and S.

In such an optimal alignment, a move on log $(x, \gg)$ indicates that the trace σ_L had an event x that was not supposed to happen according to the specification S whereas a move on specification $(\gg, y)$ indicates that σ_L was missing an event $\ell(y)$ that was expected according to S. As the alignment preserves the position relative to the trace σ_L, we can locate the exact position where σ_L had an event too much or missed an event compared to S.

In the remainder of this paper, we show how to leverage this approach to compliance checking. The optimal alignments between log and specification provide excellent diagnostic information and can be used to robustly quantify conformance. Thereby, the specification S will formally capture all traces that comply to a given compliance constraint. The alignment of a log trace σ_L to S can then clearly show where and how often σ_L deviates from the constraint.

4 Expressing Compliance Rules as Petri-net Patterns

In this section, we provide a framework for compliance rules together with an extensive list of control flow rules in 15 different categories. Each rule has a comprehensive description and is formalized as a Petri-net pattern. In Sect. 5 we generalize this approach to data- and resource-related rules.

4.1 Compliance Rule Framework

A compliance rule prescribes how an internal or cross-organizational business process has to be designed or executed. It originates in explicitly stated regulations and can refer to the individual perspectives of a business process (control flow, data flow, organizational aspects) or a combination of several perspectives. We reviewed existing literature on compliance [4,15,9,14,21,36,19,33,35], collected the rules described in these papers, and categorized them. We found that a single rule usually is not concerned with only one perspective of a process, but with several perspectives. Based on this observation, we identified six orthogonal dimensions of compliance rules, into which each of the rules could be categorized. For example, the rule "After a claim of more than 3000 EUR has been filed, two different employees need to check the validity of the claim independently." is composed of 3 basic rules that refer to (1) *control flow* ("After a claim has been filed, validity must be checked."), (2) *data flow* ("A claim over 3000 EUR requires two validity checks."), and (3) the *organization* ("Multiple validity checks are carried out by different employees.").

Furthermore, a compliance rule can (4) impose *time-related* constraints (e.g., "Within 6 months the claim must be decided.") or can be *untimed*,(5) prescribe properties of a *single case* or of *multiple* cases (e.g., "20% of all claims require a detailed check."), and (6) prescribe properties of the *process design* (e.g., "The claim process must have a time-out event handler.") or properties of the process executions, which can be *observed* (i.e., recorded in an event log).

These six dimensions give rise to the framework shown in Fig. 1. In this paper, we present compliance rules for control flow, data flow as well as organizational aspects, where we focus on untimed, observation-based properties of individual cases. The remainder of this section presents control flow compliance rules and their formalization. Section 5 presents data flow rules and organizational rules and their combination with control flow rules.

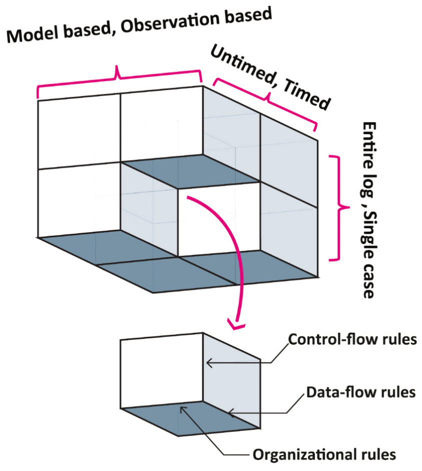


Fig. 1. Compliance Rule Framework

4.2 Control Flow Compliance Rules

Eliciting and *formalizing* compliance rules for a business process comprise determining the laws and regulations that are relevant for this process and formulating these compliance rules in an unambiguous, yet understandable manner [30]. Typically, this involves expressing a given informal requirement in a formal notation: a task an end user may not be capable of. To support elicitation, we provide end users with an *extensive library* of comprehensive compliance rules. Each rule has an informal, precise description and is accompanied by a mathematical formalization. The end user just has to pick the rule(s) that describe the given compliance requirement best; the accompanying formalization is then used for compliance checking.

We collected from literature [4,15,9,14,21,36,19,33,35] 55 compliance rules that concern the control flow perspective of a process, and classified them further into 15 categories; see Tab. 1. Each category includes several compliance rules. For example, the *Existence* category defines 2 rules in total, e.g., "In each process execution, task A should be executed" and "In each process execution, Task A should not be executed." Each rule is parameterized over tasks (e.g., Task A) or numeric parameters (e.g., governing bounds for repetitions etc.).

To formalize these rules we need to use a concrete formalism. Some compliance rules prescribe behaviors that are easier to express in terms of logical formulas (each A is followed by a B), and some rules prescribe behaviors that are easier to express in a more operational model (A, B, and C happen twice directly in sequence with no other event in between). Our literature survey found both kinds of rules to be relevant, temporal logics (e.g., LTL) against operational models (e.g., Petri-nets). Because of tool support for conformance checking [8], we decided to formalize rules as *parameterized*

Table 1. Categorization of the 55 Control Flow Compliance Rules

Category (Rules)	Description
Existence (2)	Limits the occurrence or absence of a given event A within a scope. [4],[15],[9], [14],[21],[36],[33]
Bounded Existence (6)	Limits the number of times a given event A must or must not occur within a scope. [15],[14]
Bounded Sequence (5)	Limits the number of times a given sequence of events must or must not occur within a scope. [15],[14]
Parallel (1)	A specific set of events should occur in parallel within a scope. [33]
Precedence (10)	Limits the occurrence of a given event A in precedence over a given event B. [15],[33],[14],[36],[9],[19],[21],[4],[33]
Chain Precedence (4)	Limits the occurrence of a sequence of events $A_1, \ldots, A_n$ over a sequence of events $B_1, \ldots, B_n$. [15],[14],[21]
Response (10)	Limits the occurrence of a given event B in response to a given event A. [33],[14],[21],[15],[37],[9],[19]
Chain Response (4)	Limits the occurrence of a sequence of events $B_1, \ldots, B_n$ in response to a sequence of events $A_1, \ldots, A_n$. [15]
Between (7)	Limits the occurrence of a given event B between a sequence of events A and C. [14]
Exclusive (1)	Presence of a given event A mandates the absence of an event B. [15]
Mutual Exclusive (1)	Either a given event A or event B must exist but not none of them or both. [15],[34]
Inclusive (1)	Presence of a given event A mandates that event B is also present. [15]
Prerequisite (1)	Absence of a given event A mandates that event B is also absent. [15]
Substitute (1)	A given event B substitutes the absence of event A. [15]
Corequisite (1)	Either given events A and B should exist together or to be absent together. [15]

Petri-net patterns. Although being an unusual choice, we could formalize operational rules as well as declarative rules in a systematic and understandable way. The complete collection of compliance rules and their Petri-net patterns is described in [29]. In the following we present a few characteristic rules and their formalization in terms of Petri-net patterns.

4.3 Petri-net Patterns for Compliance Rules

Bounded Existence of a Task (from category Bounded Existence). Description: "Task A should be executed exactly k times." If A occurs less than or more than k times, the rule is violated. For instance, for $k = 2$, the trace $\langle BCADBCAD \rangle$ complies to this rule and $\langle BCADBCAAD \rangle$ violates the rule.

Figure 2 shows the Petri-net pattern that formalizes this rule. Task A is expressed as an A-labeled transition. Occurrences of any other transition than A are mimicked by the Ω-labeled transition. This way, the pattern abstracts from all other trace events that are not described in the compliance rule. The transition F expresses that the end of the trace has been reached, i.e., it occurs *after* all other events of the trace occurred.

The pre-place P_k of A is initially marked with k tokens. As each occurrence of A consumes one token from P_k, A can occur at most k times. Also each occurrence of A produces one token on $Count$. By consuming k tokens from place $Count$, the final transition F can only occur (i.e., the trace can only complete) if A has occurred k times.

Figure 2 also illustrates the basic principles of Petri-net patterns for compliance rules. Each pattern has a dedicated place $Final$ that defines the final marking of the pattern. A trace σ *complies* to the pattern (its rule) iff after executing σ, final transition F is enabled, and its occurrence leads to the *final marking* that puts 1 token on place *final* and all other places are empty. In Fig. 2, the arc from F to $Initial$ ensures that once F occurs, no other tokens remain in the net and A and Ω cannot occur anymore.

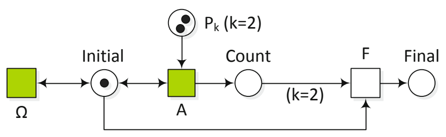


Fig. 2. Petri-net pattern for rule "Bounded Existence of a Task"

We use the alignment-based approach of Sect. 3 for checking whether a trace σ complies to this rule, by aligning σ to an occurrence sequence σ_S of the net of Fig. 2 where F is the last event of σ_S. To this end, we first have to map transitions of the pattern to events of σ by the labeling function ℓ. For example, the trace $\langle BCADBCAD \rangle$ could have the mapping $\ell(A) = \{A\}$, $\ell(\Omega) = \{B, C, D\}$, and $\ell(F) = \{\tau\}$ (the final transition F is always regarded as silent and not mapped to any trace event). For this labeling, the approach of Sect. 2 aligns $\sigma_1 = \langle BCADBCAD \rangle$ by $\gamma_1 = \begin{array}{c|c|c|c|c|c|c|c|c} B & C & A & D & B & C & A & D & \gg \\ \hline \Omega & \Omega & A & \Omega & \Omega & \Omega & A & \Omega & F \end{array}$ and $\sigma_2 = \langle BCADBCAAD \rangle$ by $\gamma_2 = \begin{array}{c|c|c|c|c|c|c|c|c|c} B & C & A & D & B & C & A & A & D & \gg \\ \hline \Omega & \Omega & A & \Omega & \Omega & \Omega & A & \gg & \Omega & F \end{array}$ and $\sigma_3 = \langle BCADBCD \rangle$ by $\gamma_3 = \begin{array}{c|c|c|c|c|c|c|c|c} B & C & A & D & B & C & \gg & D & \gg \\ \hline \Omega & \Omega & A & \Omega & \Omega & \Omega & A & \Omega & F \end{array}$. Alignment γ_1 only contains synchronous moves of σ_1 and Petri-net (except for final transition F) which means that σ_1 complies to the rule. In contrast, γ_2 contains a move on log $(A, \gg)$ on the third A. This move on log not only indicates that σ_2 violates the compliance rule, but the move $(A, \gg)$ also tells the exact location of the deviation, which is the third A. γ_3 contains a move on model $(\gg, A)$ that is required to get the Petri-net pattern into its final marking. This move on model also indicates a violation of the rule as a missing second A in the σ_3. This first rule constrains a single task. The following rules constrain *orderings* of several tasks and also shows the importance of the Ω-transition in the Petri-net patterns.

Direct Precedence of a Task (from category Precedence). Description: "Every time B occurs, it should be directly preceded by A." If B occurs without a directly preceding A, the rule is violated. For instance, $\langle ACCAAC \rangle$ and $\langle ABCAAB \rangle$ comply to the rule, whereas $\langle ABACB \rangle$ violates the rule.

The pattern of Fig. 3(top left) formalizes this rule. It basically describes a cycle of A and B, so that B can only occur if A has preceded it. As there is no Ω-transition adjacent to place p, A *directly precedes* any B (occurrences of any other transition but B are excluded).

This pattern also demonstrates the power of the alignment-based approach and for checking whether a trace complies the rule. For instance, in the compliant trace $\sigma = \langle ABCAABA \rangle$, the second and the last occurrence of A are aligned to the left-most transition A of Fig. 3 and only the first and third A that directly precede a B are aligned to enter the cycle. This capability of the alignment-based approach allows to design patterns with non-deterministic choices such as in Fig. 3, which gives greater flexibility when formalizing compliance rules.

Our approach also allows to derive *variants* of patterns. For instance, Fig. 3(top right) formalizes that "Every occurrence of B should be preceded by A (also several steps earlier)." The patterns presented so far assumed an occurrence of a task A to be represented as an atomic event A in the log. Fig. 3(bottom) formalizes the direct precedence rule for the case that task A is represented by two events $A-$ *start* (A_{st}) and $A-$ *complete* (A_{cmp}) indicating the start and completion of an ongoing activity. All Petri-net patterns of our collection rules come in these two flavors and can be picked based on the setting. The next rule demonstrates that in this way, also intricate ordering constraints can be formalized with Petri-nets in an intuitive way.

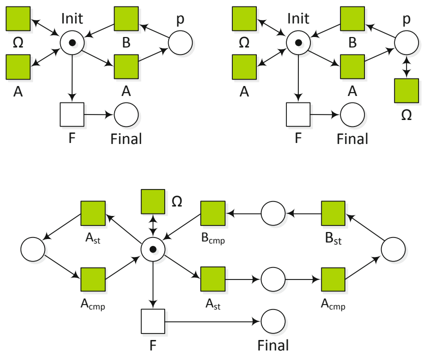


Fig. 3. Petri-net patterns for precedence

Direct Precedence or Simultaneous Occurrences of Tasks. "Task A must always be executed simultaneously or directly before task B." In case of atomic tasks this rule is identical to the preceding rule, in case of ongoing tasks, A and B can overlap in time. For example, the trace $\langle A_{st}B_{st}CA_{cmp}DB_{cmp} \rangle$ complies to this rule whereas $\langle A_{st}A_{cmp}CB_{st}DB_{cmp} \rangle$ violates this rule.

The Petri-net pattern of Fig. 4 formalizes this rule. The case where A strictly precedes B (A ends before B starts) is formalized by the lower cycle of the net. More interestingly, the case where A and B occur simultaneously is formalized by the upper cycle (white transitions are silent). There is no cycle that permits B without a preceding or simultaneous A. If there is no B or B just occurred, any events but B_{st} and B_{cmp} may occur. This is also the situation when the pattern may terminate. The replay-based approach of Sect. 3 aligns traces to this pattern as explained for "Direct Precedence of a Task."

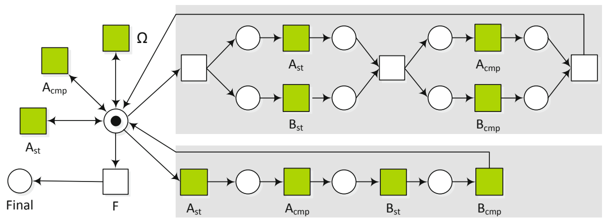


Fig. 4. Petri-net pattern for "Direct Precedence or Simultaneous Occurrences of Tasks"

The preceding rules constrained control flow in one specific dimension (ordering or number of occurrences). The next rule shows that also mixed rules occur, and how to formalize mixed rules by reusing concepts of the preceding patterns.

Bounded Existence of Sequence of Tasks (from category Bounded Sequence). Description: "The direct sequence of tasks $\langle AB \rangle$ (B exactly after A) should not occur more than k times." If $\langle AB \rangle$ occurs for the $k+1$-st time, the rule is violated. For instance, for $k = 2$, $\langle CABACBABC \rangle$ complies to this rule and $\langle CABBCABABC \rangle$ violates the rule.

The pattern of Fig. 5 formalizes this rule by combining concepts of "Bounded Existence" (Fig. 2) with concepts of "Direct Precedence" (Fig. 3). A consecutive sequence $\langle AB \rangle$ is expressed as a cycle in the pattern. The complete cycle may occur at most k times because of pre-place P_k of B. A $k+1$-st occurrence of A is permitted as it does not complete $\langle AB \rangle$, yet. After A occurred, there may be arbitrary further occurrences of A; a subsequent B still yields a direct sequence $\langle AB \rangle$, any other transition interrupts this sequence (Ω brings the token back to $Initial$).

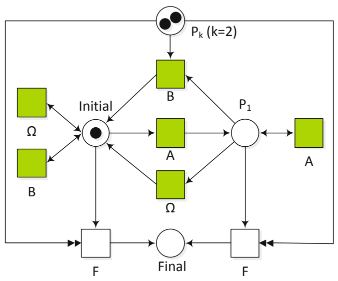


Fig. 5. Petri-net pattern for "Bounded Existence of Sequence of Tasks"

The pattern can go to the final marking at any point (that is, if there are no other events to be executed). Here, the *reset arcs* from P_k to the final transitions F ensure that all pending tokens are removed from the net (e.g., if $\langle AB \rangle$ never occurred).

Aligning the violating trace $\sigma = \langle CABBCABABC \rangle$ to this pattern yields alignment $\gamma = \frac{C|A|B|B|C|A|B|A|B|C|\gg}{\Omega|A|B|B|\Omega|A|B|A|\gg|\Omega|F}$ where the move on log $(B, \gg)$ at the third occurrence of $\langle AB \rangle$ arises because there is no token on P_k left (and hence B is not enabled in the pattern), i.e., this move on log indicates the violation. These patterns demonstrate some of the key concepts and basic building blocks that we used to formalize all 55 compliance rules in terms of Petri-net patterns.

5 Compliance to Data and Organizational Aspects

So far we presented a comprehensive collection of control flow compliance rules and their formalization as Petri-net patterns. These rules cover the *control flow* dimension of the compliance rule framework introduced in Fig. 1. In this section, we show how the pattern-based approach to compliance rules of Sect. 5 can also be applied to check compliance with respect to *data* and to *organizational aspects*, which constitute two other dimensions of the framework. As before, we consider single-case observation-based untimed compliance rules.

5.1 Data Flow Compliance Rules

A typical example of a data flow compliance rule is to **Restrict data values permitted for a task**. For example, "A discount of 10% is granted if the customer is a gold customer; 5% are granted if the customer is a silver customer." A rule of this kind prescribes that task *grant* refers to 2 attributes e.g., *customer status* and *percentage*. When task *grant discount* occurs, these attribute values need to be logged in the corresponding event such as $(grant, John, gold, 10\%)$ (see Sect. 3); otherwise compliance cannot be checked in hindsight.

When checking compliance to this rule, it is not just sufficient to check whether $grant$ occurred, but we need to check whether $grant$ occurred with the right attribute values. To this end slight changes in actual Petri-net pattern and labeling ℓ that relates Petri-net transitions to events are required. Figure 6(top) shows the Petri-net pattern for this rule. It contains two transitions *grant* that are further distinguished by the attribute value combinations that are permitted by this task.

Recall from Sect. 3 that each pattern also has a labeling function $\ell(.)$ that maps transitions to sets of events. In contrast to Sect. 4, a transition is not mapped to an event name, but to a *combination of name and attribute values*. For instance, the mapping $\ell(grant10\%gold) = \{(grant, x, y, z) \mid y = gold, z = 10\%\}$ maps transition $grant10\%gold$ only to $grant$ events which have $gold$ and 10% as their attribute values, correspondingly for $grant5\%silver$. Other occurrences of *grant* (with other attribute value combinations) are *disallowed* by mapping Ω only to events other than *grant*, e.g., $\ell(\Omega) = \{(a, x, y, \ldots) \mid a \neq grant\}$. This mapping ℓ and the pattern of Fig. 6(bottom) together formalize the compliance rule. For example, trace $\langle(\mathit{add\ item}, x, 10EUR)$ $(\mathit{add\ item}, y, 32EUR)$ $(\mathit{grant}, \mathit{Joe}, \mathit{gold}, 10\%)\rangle$ complies to this rule whereas aligning trace $\langle(\mathit{add\ item}, x, 10EUR)$ $(\mathit{add\ item}, y, 32EUR)$ $(\mathit{grant}, \mathit{Jim}, \mathit{silver}, 10\%)\rangle$ to the this rule yields a move on log $((\mathit{grant}, \mathit{Jim}, \mathit{silver}, 10\%), \gg)$ indicating that Jim was granted a wrong discount.

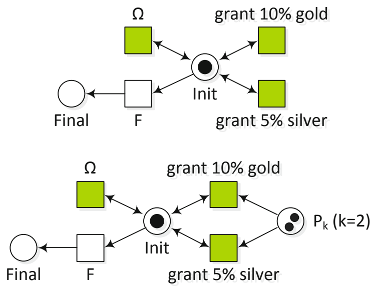


Fig. 6. Petri-net pattern to 'Restrict data values permitted for a task'

Note that data flow compliance is essentially formalized by further distinguishing transitions in the Petri-net patterns, and by defining the right mapping from transitions to events. This permits to *combine control flow rule and data flow compliance rules* also formally, e.g., the pattern of Fig. 6(bottom) formalizes that "A discount (of 10% for gold customers and 5% for silver customers) is given at most twice per case."

5.2 Compliance to Organizational Aspects

Separation of Duty (4-eyes principle). The perhaps best known compliance rule states that "Of two sequential tasks A and B, if A was performed by user R, then B must not be performed by R." Here, each task has a particular attribute *performed by* (or role) which takes as values user names or roles. Technically, the role attribute is a special data attribute: a log event $(check, Sue)$ describes that Sue performed activity $check$. A trace $\sigma_1 = \langle(receive, Tom)(check, Sue)(notify, Sue)(pay, Tom)\rangle$ complies to the 4-eye principle for tasks *check* and *pay* whereas $\langle(receive, Tom)(check, Sue)(notify, Sue)$ $(pay, Sue)\rangle$ violates the principle.

Figure 7 shows the Petri-net pattern that formalizes this compliance rule. It distinguishes two cases (as indicated by the upper and lower grey-shaded rectangle. Each case describes one compliant role assignment to tasks A and B, either A is performed by R, then B not by R, or vice versa. In this compliance rule, once R performed A, it always has to perform A and may never perform B (within the same trace). Hence the choice for either case is permanent in the pattern as well. The pattern may terminate at any point in time, and all other tasks (except for A and B with the chosen role assignments) may occur at any point in time. As for data flow compliance, the labeling $\ell(.)$ is crucial to relate patterns of the transitions to events: $\ell(\mathit{A,R}) = \{(x,y) \mid x = A, y = R\}, \ell(\mathit{A,not\text{-}R}) = \{(x,y) \mid x = A, y \neq R\}, \ell(\mathit{B,R}) = \{(x,y) \mid x = B, y = R\}, \ell(\mathit{B,not\text{-}R}) = \{(x,y) \mid x = B, y \neq R\}, \ell(\Omega) = \{(x,y) \mid x \notin \{A,B\}, y \neq R\}$.

Each user gives rise to a different labeling that has to be checked separately from other labelings. When checking compliance of trace σ_2 given above w.r.t. tasks *check*, *pay*, and user *Joe*, the alignment-based approach of Sect. 3 returns a move on log $((\mathit{pay}, \mathit{Sue}), \gg)$ indicating that the *pay* task should not have been performed by *Sue* (as it is not allowed by the pattern).

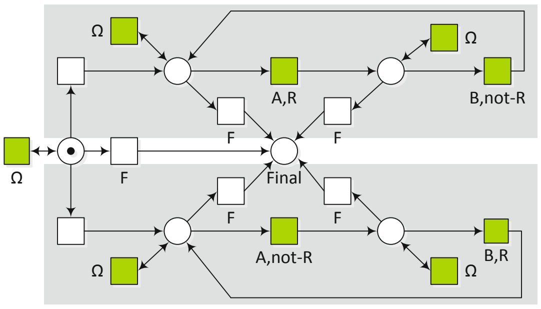


Fig. 7. Petri-net pattern for "Separation of Duty"

Altogether, compliance to data flow and to organizational aspects is orthogonal to control flow compliance and builds on mapping Petri-net transitions to events based on a combination of event name and attributes. This also allows to formalize and check rules that depend on mixture of control flow, data flow and organizational aspects.

6 Experimental Results

We have evaluated the technique on real-life log taken from the financial system of a large Dutch hospital. The log contained over 150000 events from over 700 different activities in 1150 cases, each case representing a patient. The log was obtained from financial system of the hospital in the period of 2005 to 2008. Beside anonymizing the log, all other data in log is preserved including event names, involved resources etc. We first describe the implementation used in this evaluation and then report on some compliance rules relevant to this process and the results we obtained for them.

6.1 Implementation in ProM

The presented technique is implemented in the *Compliance* package of the Process Mining Toolkit ProM 6, available from `http://www.processmining.org/`. The packet provides Petri-net patterns for the control flow compliance rules discussed in this paper. The "*Check Compliance*" plugin takes a log as input. Then the user can pick from a list of available compliance rules, those rules against which the log shall be checked. For each rule to check, the user then configures its parameters, mostly by mapping events to task names of the rule. Then the conformance checker of Sect. 3 is called to align the log to the rule's Petri-net pattern. The resulting alignment is shown to the user. Each aligned trace is shown in a separate row and deviations are highlighted: a move on log indicates an event occurred which did not comply to the rule, a move on model indicates which event skipped in log such that log does not comply to the rule. Several figures in the next section show these alignments.

To ease presentation of our results, we abstract long sequences of events that are not relevant to the compliance rule (i.e., which are mapped to Ω-transitions), to shorter sequences. This way, order and relative position of compliance-relevant events are preserved while irrelevant details are abstracted from.

6.2 Case Study Constraints and Results

In the case study we followed the standard use case for compliance checking: (1) check relevant regulations and elicit respective compliance requirements, (2) for each requirement, identify the patterns that precisely express the requirement from the rule collection in Tab. 1, (3) take the corresponding Petri-net pattern and map its transitions to the events in the given log, and (4) run the conformance checker.

In the following, we describe our findings for three compliance requirements that were derived from the financial department's internal policies and medical guidelines.

Compliance Requirement 1. "The hospital should register each visiting patient and prevent duplicate registrations for a patient." This requirement is formalized by the compliance rule "Event *First Visit Registration* should occur exactly once per case." from the category '*bounded existence*' of Table 1. The corresponding pattern is shown in Fig. 2 (left). To obtain a reliable result, we needed to filter the data for patients who started their treatment between 2005 and 2008. We checked compliance for 640 cases and identified 622 compliant and 18 non-compliant cases.

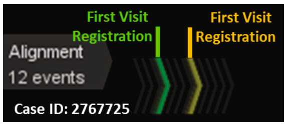


Fig. 8. Non-compliant case for '*First Visit Registration* should occur exactly once'

Figure 8 shows diagnostic information for a non-compliant case. As described above, ProM maps the trace on to the compliance-influencing events: *First Visit Registration* occurs twice, where the first occurrence is compliant (highlighted green) and the second one should not have been in the trace (highlighted yellow, *move on log*).

Compliance Requirement 2. The patients are sent to the hospital for specialized treatment. Therefore a basic X-ray scan has to be performed after a patient was registered. This requirement is formalized by two rules: "Event *x-ray* should occur at least once per case." and "Every time the event '*x-ray*' occurs, '*First Visit Registration*' should have happened before *x-ray*." Figure 9 shows compliance results for the second rule, which is formalized by Fig. 3(top right); we found 104 compliant cases out of 640.

An example of a non-compliant case is shown in Fig. 9. The relevant sequence of events in this case is $\langle \dots \textit{x-ray} \dots \textit{First Visit Registration} \dots \rangle$. The compliance checker identified the *x-ray* event as an event that should not have happened (highlighted yellow, *move on log*) because it occurred before the *First Visit Registration* event). In addition, *the checker highlights the position where x-ray should have occurred* as a move on model (highlighted purple).

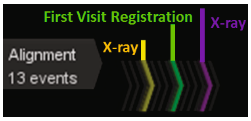


Fig. 9. Non-compliant case for '*First Registration Visit* precedes *x-ray*'

Compliance Requirement 3. "For safety reasons, either a *CT-Scan* or an *MRI test* of an organ should be taken from a patient but not both." The corresponding compliance rule from the *Exclusive* category has the Petri-net pattern of Fig. 10(top).

We checked this rule and identified 1092 compliant cases out of 1150. Fig. 10(bottom) shows diagnostic information for one non-compliant case. The relevant sequence of events for this case is $\langle \dots CT.MRI \dots CT \dots MRI \dots \rangle$. The occurrence of *CT* together with *MRI* is a violation. In addition, the diagnostic information provided by the checker clearly shows *several* violations due to multiple occurrences of *CT* (*move on log*, highlighted yellow).

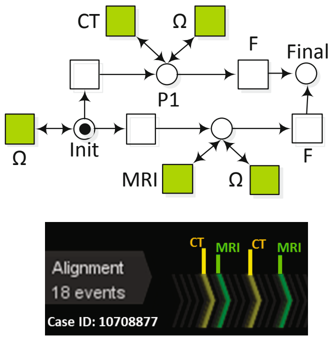


Fig. 10. Petri-net pattern and a non-compliant case for 'CT-Scan and MRI Test exclude each other'

Altogether we could identify and precisely locate various compliance deviations from given compliance rules in a real-life log.

7 Conclusion

Today's organizations need to comply to an increasing set of laws and regulations. Compliance requirements are often described in natural language which makes checking compliance a difficult task. In this paper we provided a first comprehensive collection of control flow compliance rules which allow to formally capture a large set of compliance requirements. Moreover we presented a robust technique for backwards compliance checking which enables us to provide diagnostic information in case of violations. The technique is also applicable to check compliance of artifact-centric processes [16]. The approach is supported by ProM plugins and we tested our techniques using real-life logs and compliance requirements.

The unusual choice of formalizing compliance rules as Petri-nets rather than logics posed no difficulties. Yet, we can foresee benefits from a mixed formalization of declarative rules by logics and operational rules by Petri-nets. Note that in no situation, the end user is confronted with the formalization of the rule, but picks rules by their informal description.

We showed that our approach can also handle organizational rules and data flow rules to constrain individual tasks. Handling constraints across several tasks requires to generalize the technique, in particular the underling conformance checker [8]. Also, the mapping between event attributes and transitions is cumbersome and currently specified at a technical level using concrete values; a more user-friendly approach to specify organizational and data flow rules is required.

Thus, future work aims at exploring the compliance rule framework (Fig. 1) further and extending the compliance rule set (Table 1) for other dimensions, i.e., with collections of compliance rules restricting data flow, process resource, and process time.

Acknowledgements. We thank R.Mans, A.Adriansyah and C.Stahl for their substantial support in writing this paper. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 257593 (ACSI).

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. 2, 182–192 (2012)
3. 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. (ed.) OTM 2005, Part I. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
4. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M., Kumar, A., Verdonk, M.: Conceptual Model for Online Auditing. Decision Support Systems 50(3), 636–647 (2011)
5. Abdullah, N.S., Sadiq, S.W., Indulska, M.: Information systems research: Aligning to industry challenges in management of regulatory compliance. In: PACIS 2010, p. 36. AISeL (2010)
6. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards Robust Conformance Checking. In: zur Muehlen, M., Su, J. (eds.) BPM 2010 Workshops. LNBIP, vol. 66, pp. 122–133. Springer, Heidelberg (2011)
7. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based Fitness in Conformance Checking. In: ACSD 2011, pp. 57–66. IEEE (2011)
8. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: EDOC 2011, pp. 55–64. IEEE (2011)
9. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
10. Awad, A., Weske, M.: Visualization of Compliance Violation in Business Process Models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 182–193. Springer, Heidelberg (2010)
11. Calders, T., Guenther, C., Pechenizkiy, M., Rozinat, A.: Using Minimum Description Length for Process Mining. In: SAC 2009, pp. 1451–1455. ACM Press (2009)
12. Christopher Giblin, S.M., Pfitzmann, B.: Research report: From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Tech. rep., IBM Research GmbH, Zurich Research Laboratory, Switzerland (2006)
13. Cook, J., Wolf, A.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology 8(2), 147–176 (1999)

14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE 1999, pp. 411–420 (1999)
15. Elgammal, A., Turetken, O., van den Heuvel, W.-J., Papazoglou, M.: Root-Cause Analysis of Design-Time Compliance Violations on the Basis of Property Patterns. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 17–31. Springer, Heidelberg (2010)
16. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011)
17. Fötsch, D., Pulvermüller, E., Rossak, W.: Modeling and verifying workflow-based regulations. In: ReMo2V 2006. CEUR Workshop Proceedings, vol. 241 (2007)
18. 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)
19. Giblin, C., Liu, A.Y., Müller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models (realm). In: Frontiers in Artificial Intelligence and Applications, JURIX 2005, vol. 134, pp. 37–48. IOS Press (2005)
20. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009)
21. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electr. Notes Theor. Comput. Sci. 153(2), 117–133 (2006)
22. Kharbili, M.E., de Medeiros, A.K.A., Stein, S., van der Aalst, W.M.P.: Business process compliance checking: Current state and future challenges. In: MobIS 2008. LNI, vol. 141, pp. 107–113. GI (2008)
23. Kharbili, M.: Business process regulatory compliance management solution frameworks: A comparative evaluation. In: APCCM 2012. CRPIT, vol. 130, pp. 23–32. ACS (2012)
24. Lu, R., Sadiq, S., Governatori, G.: Compliance Aware Business Process Design. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer, Heidelberg (2008)
25. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1), 1–62 (2010)
26. Muñoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer, Heidelberg (2010)
27. Muñoz-Gama, J., Carmona, J.: Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In: CIDM 2011. IEEE (2011)
28. Pitzmann, B., Powers, C., Waidner, M.: Ibm's unified governance framework (ugf). Tech. rep., IBM Research Division, Zurich (2007)
29. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Diagnostic information in compliance checking. Tech. rep., BPM Center Report BPM-12-11, BPMcenter.org (2012)
30. Ramezani, E., Fahland, D., van der Werf, J.M., Mattheis, P.: Separating Compliance Management and Business Process Management. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part II. LNBIP, vol. 100, pp. 459–464. Springer, Heidelberg (2012)
31. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
32. Sadiq, S., 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)

33. Schleicher, D., Grohe, S., Leymann, F., Schneider, P., Schumm, D., Wolf, T.: An approach to combine data-related and control-flow-related compliance rules. In: SOCA 2011, pp. 1–8. IEEE (2011)
34. Schleicher, D., Anstett, T., Leymann, F., Schumm, D.: Compliant Business Process Design Using Refinement Layers. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 114–131. Springer, Heidelberg (2010)
35. Schleicher, D., Fehling, C., Grohe, S., Leymann, F., Nowak, A., Schneider, P., Schumm, D.: Compliance domains: A means to model data-restrictions in cloud environments. In: EDOC, pp. 257–266 (2011)
36. Schumm, D., Leymann, F., Ma, Z., Scheibler, T., Strauch, S.: Integrating compliance into business processes: Process fragments as reusable compliance controls. In: MKWI 2010. Universitätsverlag Göttingen (2010)
37. Schumm, D., Turetken, O., Kokash, N., Elgammal, A., Leymann, F., van den Heuvel, W.-J.: Business Process Compliance through Reusable Units of Compliant Processes. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 325–337. Springer, Heidelberg (2010)
38. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: CIDM 2011, pp. 148–155. IEEE (2011)
39. Wolter, C., Meinel, C.: An approach to capture authorisation requirements in business processes. Requir. Eng. 15(4), 359–373 (2010)

Using Mapreduce to Scale Events Correlation Discovery for Business Processes Mining

Hicham Reguieg[1], Farouk Toumani[1], Hamid Reza Motahari-Nezhad[2], and Boualem Benatallah[3]

[1] LIMOS, CNRS, Blaise Pascal University, Clermont-Ferrand, France
`{reguieg,ftoumani}@isima.fr`
[2] HP Labs, Palo Alto, USA
`hamid.motahari@hp.com`
[3] CSE, UNSW, Sydney, Australia
`boualem@cse.unsw.edu.au`

Abstract. In this paper, we present a scalable data analysis technique to support efficient event correlation for mining business processes. We propose a two-stages approach to compute correlation conditions and their entailed process instances from event logs using MapReduce framework. The experimental results show that the algorithm scales well to large datasets.

1 Introduction

As a key-step in any process discovery method, *event correlation discovery* consists in analyzing event logs or interactions among processes entities in order to find relationships between events that belong to the same business process execution instance [1,5]. This task is recognized as challenging for at least two main reasons: (i) *big data is a fact of the modern world*. Indeed, modern infrastructures supporting large scale enterprise applications record more and more information about the history of business processes, and (ii) event correlation discovery is in essence a data-intensive task. It consists of various repetitive data-intensive computations (e.g., aggregation of events, intersection and join, computing transitive closures, etc) on a sheer large amount of data. This trend is particularly supported by the unprecedented computation opportunities offered by the emerging service-oriented cloud computing infrastructures, which open new possibilities for process mining in cross-organizational and/or multi-tenants settings [8].

In this paper, we investigate the application of modern large scale data analysis techniques, and in particular the MapReduce approach, to support efficient event correlation discovery in process mining activities. Mapreduce [2] has emerged recently as a promising approach for processing huge amounts of data on a multitude of machines in a cluster. It provides a simple programming framework that enables harnessing the power of very large data centers, while hiding low level programming details related to parallelization, fault tolerance, and load balancing.

We propose a two-stages approach to compute correlation conditions and their entailed process instances from event logs. The first stage is devoted to the computation of simple correlation conditions (called atomic conditions). The second stage deals with composite correlation conditions (conjunctive and disjunctive conditions) and associated process instances. Each stage is implemented as a map and reduce step. The main difficulties encountered when designing our approach are related to log partitioning and redistribution in order to generate efficient parallel computations.

The paper is organized as follows. Section 2 gives an overview on the event correlation approach used in this paper and introduces the main MapReduce concepts. Section 3 presents our approach to compute interesting process instances by analyzing process events logs. We conclude and draw future research directions in section 4.

2 Preliminaries

Overview on Event Correlation Discovery. This paper uses as a basis the event correlation discovery approach presented in [5]. The pursued goal in that work is to analyze web service interaction logs in order to identify correlation between events that belong to the same process execution instance. The extracted knowledge is then exploited to build views that represent parts of the original business processes. The global approach proposed in [5] consists in three main steps: (i) finding correlation between events (transform messages in the log into a set of process instances), (ii) using an algorithm for process mining to discover the process model for each set of the discovered process instances, and (iii) organizing the process views into a process map. Our aim is to investigate the implementation of this approach in a MapReduce Framework. We consider as input a web services interaction log L. A log L can be viewed as a relation over a relational schema $\mathcal{L}(id, A_1, \ldots, A_n)$, where $U = \{A_1, \ldots, A_n\}$ is a set of attributes used in messages parameters and id a special attribute denoting message identifiers. Elements of L are called messages. Since typically a message $m \in L$ contains only a subset of attributes of U, therefore m may have several undefined attributes in L (i.e., having a *null* value). Correlated messages are identified using *correlation conditions*. A correlation condition, denoted by $\psi(m_l.A_i, m_p.A_j)$, is a boolean predicate over the attributes A_i and A_j of respectively the two messages m_l and m_p. Conditions of the form $m_l.A_i = m_p.A_j$ are called *atomic conditions* and denoted ψ_{A_i,A_j}. A *conjunctive* (respectively, *disjunctive*) condition consists of conjunction (respectively, disjunction) of atomic conditions. Criteria and measures, based on the number of distinct values as well as the number of discovered process instances, have been proposed in [5] to prune non-interesting conditions. Based on that measures, we present in section 3, a MapReduce-based approach to discover interesting correlation conditions and associated process instances from an events log.

Overview on the MapReduce. A MapReduce framework provides a simple programming constructs to perform a computation over an input file f through two primitives: a *map* and a *reduce* functions [2]. It operates exclusively on $\langle key, value \rangle$ pairs and produces as output a set of $\langle key, value \rangle$ pairs. A *map* function takes as input a data set in form of a set of key-value pairs, and for every pair $\langle k, v \rangle$ of the input returns zero or more intermediate key-value pairs $\langle k', v' \rangle$. The *map* outputs are then processed by *reduce* function. A *reduce* function takes as input key-list a pair $\langle k', list(v') \rangle$, where k' is an intermediate key and $list(v')$ is the list of all the intermediate values associated with k', and returns as final result zero or more key-value pairs $\langle k'', v'' \rangle$. Several instantiations of the *map* and *reduce* functions can operate simultaneously. A given reduce execution requires all the intermediate values associated with a same intermediate key k' (i.e., for a given intermediate key k', all the pairs $\langle k', v' \rangle$ produced by the different map tasks **must be** processed by the same *reduce* task). Typically, MapReduce programs are executed on clusters of several nodes and both their inputs and outputs are files in a distributed file system.

3 Event Correlation Discovery Using Elastic MapReduce

In this section, we describe our approach for computing correlation conditions, and the process instances entailed by these conditions, from event logs. The general approach, depicted at figure 1, consists in two main stages: (i) a first map and a reduce step to compute atomic correlation conditions and their associated process instances, and (i) a second step that implements a MapReduce levelwise-based algorithm to compute conjunctive conditions and associated instances. Disjunctive correlation conditions discovery, not discussed here for space reasons, can be processed following a similar approach. We briefly describe below each of these two steps. More technical details are available in [7].

Computing Atomic Correlation Conditions. Given an events log L, the goal is to compute the interesting atomic correlation conditions as well as the

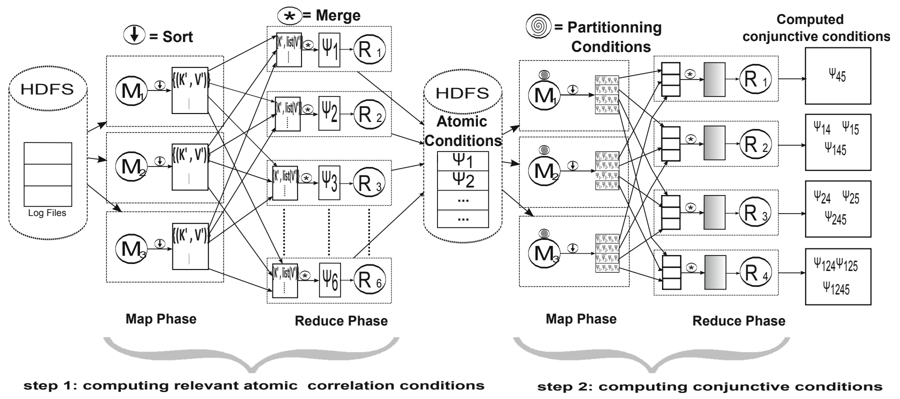


Fig. 1. The proposed two-steps approach

process instances entailed by these conditions. One of the encountered difficulty is to decide how data and computations should be partitioned in order to efficiently execute the operations entailed by this task. The main idea is to use several instantiations of the reduce functions in order to process in parallel candidate conditions ψ_{A_i,A_j} associated to a couple of attributes A_i, A_j. Hence, a map phase will hash-partitions the content of the events log across the network based on keys in such a way that every pair of attributes A_i, A_j (with $i, j \in [1, n]$) of L will be allocated to a unique reducer (i.e., identified by a unique key value). To achieve this task, the map function will be in charge of projecting the log L on each pair A_i, A_j (with $i, j \in [1, n]$), of its attributes, and then assigning a same unique key to the projected record of A_i, A_j. More precisely, a map function takes as input a key-value pair (k, v) with $k = null$ and $v = m$ a message in L and for each pair of values $m.A_i$ and $m.A_j$ of m, with $i, j \in [1, n]$, will emit two intermediate key-value pairs (k'_{ij}, v'_i) and (k'_{ij}, v'_j) where: k'_{ij} is a unique key identifying a couple (i, j), and each value $v'_l = l\text{-}m.A_l\text{-}m.id$, with $l \in \{i, j\}$, corresponds to the message value $m.A_l$ tagged with both the attribute position l, as a prefix, and the message identifier $m.id$, as a postfix. The map function ensures that: (i) a given pair of attributes A_i and A_j is allocated to only one reducer, and (ii) a given reducer, in charge of the attributes A_i and A_j, will receive all the values of these attributes appearing in L (i.e., the values of the projections $\pi_{A_i}(L)$ and $\pi_{A_j}(L)$ are tagged and sent to the same reducer). Once a reducer node has received all intermediate data (k'_{ij}, v'_i) and (k'_{ij}, v'_j) related to the attributes A_i and A_j, it merges the data to produce a single pair $\langle k'_{ij}, list(v') \rangle$. Note that, during the shuffle and sort phase, MapReduce sorts intermediate key-value pairs by the keys but not by the values. However, it is very convenient for our purposes to also sort the intermediated values since the computations inside the reducer will take benefits from such an order. Therefore, instead of implementing an additional secondary sorting within the reducer, we used the *value-to-key conversion* design pattern [4], which is known to provide a scalable solution for secondary sorting. This is achieved by moving intermediate values into the intermediate keys, during the map phase, to form composite keys, and then we let the execution framework handle the sorting. Therefore, a given reducer will process as input a pair $\langle k'_{ij}, list(v') \rangle$ where $list(v')$ is sorted by the intermediate values. Moreover, since the values are prefixed by the attribute position, then if $i < j$, the values of the attribute A_i will appear in $list(v')$ before the values of A_j. A reduce function then takes as input an intermediate key-list of values pair $\langle k'_{ij}, list(v') \rangle$ and then performs some computations, essentially counting distinct values, computing intersections and transitive closures, corresponding to the measures defined in [5] and used to prune non-interesting atomic conditions. Finally, a reducer writes the final result to the distributed file system. To efficiently implement these tasks, we use appropriate data structures that enables compact storage of sorted values of the corresponding attributes (c.f., [7] for more technical details).

Computing Conjunctive Conditions. Conjunctive conditions are computed using conjunctive operator on atomic conditions: let ψ_1 and ψ_2 be two atomic conditions elicited during the previous step, then the goal is to compute the process instances entailed by the condition $\psi_1 \wedge \psi_2$, noted ψ_{12}. More generally, given a set AC of atomic conditions, the goal is to identify the set of *minimal* atomic conditions that partition the log into interesting process instances. As explained in [5], such a task can be achieved using a levelwise-like approach where, roughly speaking, each level is determined by the length, in terms of number of conjuncts, of the considered conditions. Starting from atomic conditions (level 1), the discovery process consists in two main parts: (i) generating candidate conditions of level k from candidates of level $k-1$, and (ii) pruning non interesting conditions. At each level, the process instances associated with each generated candidate condition are computed and used to prune, if any, the considered candidate condition. To cast such a levelwise-like algorithm into a mapreduce framework, the main issue to deal with is the specification of how to distribute the candidates among reducers such that the generation and pruning computations are effectively parallelized. We propose to partition the space of candidates in such a way that a given partition can be handled by a unique reducer. This enables to avoid multiple mapreduce steps in order to compute conjunctive conditions. We proceed as follows to compute the partitions (c.f., step 2 at figure 1). Let AC be a set of n atomic conditions and let $P = \{\psi_1, \ldots, \psi_l\} \subseteq AC$ be a subset of AC containing l atomic conditions, hereafter called the partitioning conditions. The main idea is to define partition of the space of candidate conditions with respect to the presence or absence of partitioning conditions. We annotate a partition with a condition ψ_i to indicate that this partition is made of candidates that contain the subscript i and with $\bar{\psi_i}$ to indicate that the partition is made of candidates that do not contain such a subscript. Consequently, given AC and P defined as previously, the set $\mathcal{P}$ of partitions of the space of candidates using P is obtained as follows: $\mathcal{P} = \{\psi_1, \bar{\psi_1}\} \times \ldots \times \{\psi_l, \bar{\psi_l}\}$. Each element $(\phi_1, \ldots, \phi_n) \in \mathcal{P}$, with $\phi_i \in \{\psi_i, \bar{\psi_i}\}$, for $i \in [1, l]$, forms a partition of the space of candidate conditions. Note that the obtained partitions are balanced (i.e., they have the same number of candidate conditions) and form a partition of the initial space of candidates. It is also worth noting that each partition can be treated separately from the others in order to compute the corresponding interesting conditions. Roughly speaking, the proposed mapreduce based levelwise algorithm works as follows: it takes as input a set CA of atomic conditions and a set P, subset of CA of partitioning conditions. Then, the map function generates the different partitions and sends each condition, and its associated data, to the corresponding reducer. The reduce function implements: (i) a generation of candidate conditions procedure which is suitable for parallelization, and (ii) pruning tasks using the measures defined in [5].

We conducted preliminary experiments to evaluate the overhead due to the use of the MapReduce framework. We considered both real world datasets and randomy generated datasets with different log sizes. Two main observations can be derived from the first results: (i) they suggest that the extension of the proposed approach with additional map-reduce steps is potentially interesting since it will increase the level of parallelization, and hence improving scalability, without impacting too negatively the global performances, and (ii) the implementation of stage 2 can be improved using compact data structures and specialized Apriori-like algorithms.

4 Conclusions and Future Work

Since its introduction in [2], MapReduce has been used in several application domains such as data management [4], data analysis [3,6], text processing [4], etc. To the best of our knowledge, this is the first work that uses MapReduce in processes discovery. In this paper, we studied the problem of event correlation discovery using the MapReduce framework. We proposed a two-stages approach to compute correlation conditions and their corresponding process instances from service interactions event logs. We described efficient methods to partition an events log across cluster nodes in order to balance the workload related to atomic and conjunctive conditions computation while reducing data transfer. First experimental results show that the overhead introduced by the Mapreduce approach is small compared to gain in performance and scalability. This suggests future research work on investigation of additional implementation strategies, e.g., by increasing the number of MapReduce steps or using different partitioning techniques that may increase the overhead while improving the scalability of the proposed approach.

References

1. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)
2. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004, vol. 6, p. 10 (2004)
3. Li, B., Mazur, E., Diao, Y., McGregor, A., Shenoy, P.: A platform for scalable one-pass analytics using mapreduce. In: SIGMOD 2011, pp. 985–996. ACM, New York (2011)
4. Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers (2010)
5. Motahari Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. VLDB J. 20(3), 417–444 (2011)
6. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: SIGMOD 2009, pp. 165–178 (2009)
7. Reguieg, H., Toumani, F., Motahari-Nezhad, H.R., Benatallah, B.: Using mapreduce to scale events correlation discovery for business processes mining. Hewlett Packard Laboratories Technical Report (2012)
8. van der Aalst, W.M.P.: Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 8–25. Springer, Heidelberg (2010)

A Framework for Behavior-Consistent Specialization of Artifact-Centric Business Processes

Sira Yongchareon[1], Chengfei Liu[1], and Xiaohui Zhao[2]

[1] Faculty of Information and Communication Technologies
Swinburne University of Technology, Victoria, Australia
{syongchareon,cliu}@swin.edu.au
[2] Faculty of Information Sciences and Engineering
University of Canberra, Australia
xiaohui.zhao@canberra.edu.au

Abstract. Driven by complex and dynamic business process requirements, there has been an increasing demand for business process reuse to improve modeling efficiency. Process specialization is an effective reuse method that can be used to customize and extend base process models to specialized models. In the recent years, artifact-centric business process modeling has emerged as it supports a more flexible process structure compared with traditional activity-centric process models. Although, process specialization has been studied for the traditional models by treating a process as a single object, the specialization of artifact-centric processes that consist of multiple interacting artifacts has not been studied. Inheriting interactions among artifacts for specialized processes and ensuring the consistency of the processes are challenging. To address these issues, we propose a novel framework for process specialization comprising artifact-centric process models, methods to define a specialized process model based on an existing process model, and the behavior consistency between the specialized model and its base model.

1 Introduction

Complex business process requirements from different customer needs, government regulations, outsourcing partners, etc., result in frequent changes and revision to business processes. Therefore, reusability of business processes is highly sought after to improve process modeling efficiency. In this background, organizations strive for a more efficient and systematic approach to flexibly define and extend their business processes. Business process reuse aims to support on-demand customization and extension of existing business processes by establishing a modular and a repository of process components [18]. Business process specialization is deemed as one of main mechanisms that can be used to construct a specific business process by extending a generic reference process model. With specializations, processes can be reported at different levels of generality, and can be compared across the specializations [19].

Current activity-centric modeling approaches focus on the conformation of tasks and the control-flows among tasks according to specific logics. Intuitively, constructing processes with sequenced activities leads to highly-cohesive and tightly-coupled process structures; therefore, process componentization and extension are difficult to be achieved in a natural way [17]. In recent years, *artifact-centric* approaches to business process modeling have emerged and been widely studied [1, 2, 3, 7, 11, 13, 16, 17, 19]. These approaches naturally lend themselves well to both object-orientation and service-orientation design principles, as they focus on the design of both business artifacts involved in a process and services (a.k.a. tasks) performing on such artifacts. Owning to the object-oriented nature, the artifact-centric models support higher level of flexibility, extensibility, and reusability.

The existing approaches for the specialization of business processes treat a process as a single object [8, 9, 10]; hence, traditional object specialization techniques in object-oriented analysis and design can be applied (e.g., from [4]). For artifact-centric processes, specializations should not only apply on each individual artifact but also on their interactions. Some works have initiated the study of object lifecycles and their interactions within (or between) business processes in various areas, e.g., process adaptation and dynamic changes [20], design compliance [6, 13], conformance checking [16], and contract for inter-org processes [12]. However, a specialization mechanism that takes into account the interactions of objects and the guarantee of behavior consistency between a specialized process and its base process brings in technical challenges and requires further study. To address these challenges, we propose a novel framework for behavior-consistent process specialization that consists of artifact-centric process model, methods to define a specialized process model based on an existing model, and the behavior consistency between the specialized model and its base model.

The remainder of this paper is organized as follows. Section 2 presents an artifact-centric process model and an approach to process specialization. Section 3 discusses the behavioral properties and the consistency between a specialized model and its base model. Section 4 reviews and discusses related works. Finally, the conclusion and future work are given in Section 5.

2 Specialization of Artifact-Centric Business Process Model

To begin with, we briefly introduce an *artifact-centric business process model* (*ACP model*) (e.g., in [2, 3]). The ACP model constitutes of three sets: *artifact classes*, *tasks*, and *business rules*. An *artifact class* (or *artifact* if the context is clear), containing its relevant attributes and states, is a key business entity involved in business processes. A *task* performs read/write operations on some artifact(s). A *business rule* is defined in a *Condition-Action* style to associate task(s) with artifact(s). It describes on what *pre-condition* a particular task is executed, and what *post-condition* that the effect (after performing such task) must satisfy. We can say that a (complete) set of business rules defined in a process model specifies the control logic of the whole process from its beginning to its end. Now, we use two simplified product ordering processes to illustrate and motivate the artifact-centric process specialization, as shown in Fig. 1.

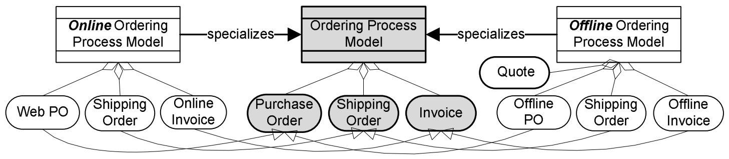


Fig. 1. Generic ordering process with its two specializations

The example depicts a generic ordering process model with its two possible specialized process models: *Online* and *Offline* ordering processes. The former offers a service to only retail customers on the web, while the latter accepts both retail and wholesale customers. The *Online* ordering process has each artifact specializes its base artifact in the ordering process, e.g., *Web PO* specializes *Purchase Order*. Not only the internal behavior of *Web PO* is specialized, but it is possible that some synchronization between *Web PO* and the other artifact(s) may also need to be modified due to the specialization. Now, consider the *Quote* artifact that is added into the *Offline* ordering process. Extending this artifact, of course, requires some synchronization with the other artifact(s), e.g., *Offline PO*. Next, Section 2.1 explains more details about how to define an ACP model and Section 2.2 introduces our approach to define a specialized ACP model based on an existing ACP model.

2.1 Syntax of ACP Model

First, we begin with the definitions of an artifact. *Artifact schema* $Z = \{C_1, C_2 ..., C_x\}$ is a finite set of *artifact classes*. Each *artifact* $C_i \in Z$ ($1 \le i \le x$) can be defined as a tuple (A, s^{init}, S, S^f) where, set $A = \{a_1, a_2, ..., a_y\}$, and each $a_j \in A(1 \le j \le y)$ is a name-value pair attribute; set $S = \{s_1, s_2, ..., s_z\}$ contains the possible states of the instances of class C_i; s^{init} is the *initial* state, and $S^f \subseteq S$ is a set of *final* states. For example in Fig. 1, the *Purchase Order* (*PO*) artifact can be defined as ({*OrderID*, *SupplierID*, *GrandTotal*, *SubmitDate, CompleteDate*}, *init*, {*created, confirmed, canceled, ready to ship, dispatched, billed, closed*}, {*closed, canceled*}), and the *Shipping Order* (*SO*) artifact can be defined as ({*ShippingID*, *OrderID*, *SubmitDate, ShipDate, CompleteDate*}, *init*, {*scheduled, in transit, arrived, completed*}, {*completed*}). Next, we define *business rules* (in a *condition-action* style) to capture the processing control logic of artifacts in a process. A *business rule*, denoted as *r*, is a tuple (λ, β, v) where λ and β are a *pre-condition* and *post-condition*, respectively; v is a task that performs read/update operations on the attributes and the processing states of some artifact(s). In this paper, we do not focus on the task-level information, i.e., the specification of task is omitted; and, for the simplification, we restrict both pre- and post-conditions to be expressed by a conjunctive normal form (CNF). This form contains two types of propositions over *Z*: (1) *state proposition* (the *instate* predicate) and (2) *attribute proposition* (the *defined* and scalar comparison operators). We write: *defined*(*C, a*) if attribute $a \in C.A$ of artifact of class *C* has a value; *instate*(*C, s*) if state $s \in C.S$ of artifact of class *C* is active. Table 1 shows an example (incomplete) set of business rules that are used in our generic ordering process.

Table 1. Example of business rules

r_1 : Buyer confirms *Purchase Order* po to the selected Supplier	
Pre-condition	*instate*(po, *sent_to_supplier*) ∧ *defined*(po, *OrderID*) ∧ *defined*(po.*SupplierID*)
Task	*confirmPO*(po)
Post-condition	*instate*(po, *confirmed*) ∧ *defined*(po.*SubmitDate*)
r_2 : Supplier creates *Shipping Order so* for *Purchase Order* so	
Pre-condition	*instate*(po, *confirmed*) ∧ *defined*(po.*SupplierID*) ∧ *instate*(so, *init*)
Task	*createSO*(po, so)
Post-condition	*instate*(po, *ready_to_ship*) ∧ *instate*(so, *scheduled*) ∧ *defined*(so.*ShippingID*) ∧ *defined*(so.*OrderID*)

Definition 1: (Artifact-Centric Process Model or ACP model). An *ACP model*, denoted as Π, is a tuple (Z, V, R), where Z is an artifact schema, V is a set of tasks, and R is a set of business rules over Z.

Note that we call a *synchronization (sync) rule* for a business rule that is used to induce multiple state changes (of different artifacts), e.g., r_2 in Table 1. We also define two auxiliary functions: function $pre\_s(r, C)$ returns a set of states $\{s_1, s_2, \ldots, s_x\}$ where business rule $r \in Z$ and state $s_i \in C.S (1 \leq i \leq x)$ is defined in the $instate$ predicate of the pre-condition of r; and function $post\_s(r, C)$ returns a set of states of artifact C appearing in the post-condition of r.

2.2 Approach to Artifact-Centric Business Process Specialization

Intuitively, we can adopt a single object specialization in the traditional object-oriented (OO) approaches (e.g., [4, 8-10]) for an individual artifact class in our model. Apart from the specialization of artifacts in a process, we investigate the specialization of their interactions. In the design and modeling phase, we propose that the specialization of ACP models can be achieved by two construction methods: *artifact refinement* and *artifact extension*.

– *Artifact refinement*. Process modelers decide to inherit an artifact from a base model by refining (adding/modifying) some corresponding business rules and states to the specialized model. The pre-condition and post-condition of a modified rule may have a state of the supertype refined into new state(s) in the subtype. Note that the refinement can be performed on a single business rule that is used to synchronize two or more artifacts.
– *Artifact extension*. Process modelers decide whether there is a need of any additional artifact for the specialized process. Adding new artifacts to a process implies that the process requires not only new business rules (of such artifact) but also sync rules between the new artifact and existing artifact(s).

Fig. 2 shows an example of specialized *Offline* ordering process and its base ordering process. The dash line linked between transitions of different artifacts indicates a sync rule that is used between such artifacts. The shaded artifacts represent *extended* artifacts. Similarly, for an existing artifact, a set of gray-shaded states and their corresponding transitions represents the refinement of its base artifact. In Fig. 2 (b), *Offline PO* is specialized by applying *artifact extension* (*Quote*, *Picking List*, and *Shipping List* are added with additional sync rules) and *artifact refinement* (the *created* state and the transition from the *confirmed* state to *ready to ship* state are refined with more details). Similarly, *Offline invoice* is specialized by applying *artifact refinement* on the transition from the *issued* state to the *cleared* state. Next, we define a process specialization function that maps a specialized ACP model to its supertype, called *ACP specialization*.

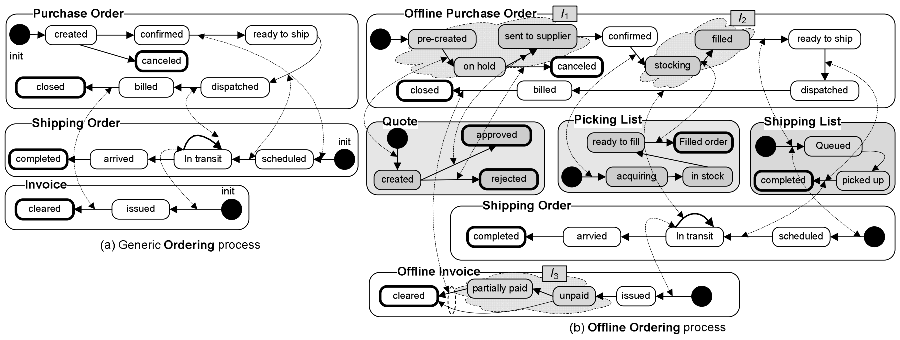


Fig. 2. Example of ACP specialization

Definition 2: (ACP specialization). Given two ACP models $\Pi = (Z, V, R)$ and $\Pi' = (Z', V', R')$, we define a specialization relation between Π and Π' by *ACP specialization function* $ps_{\Pi' \to \Pi} : Z' \cup V' \cup R' \to Z \cup V \cup R \cup \{\varepsilon\}$ such that *ps* is a total function mapping from each element in specialized model Π' onto the element in Π or *empty element* ε. The specialization methods can be expressed by ps as follows.

– *Artifact refinement.* Let artifact $C' \in Z'$ refine artifact $C \in Z$, a set of business rules $R_x' \subseteq R'$ refine business rule $r \in R$, a set of tasks $V_y' \subseteq V'$ refine tasks $v \in V$, and a set of states $S_z' \subseteq C'.S$ refine state $s \in C.S$. The following statements hold: (a) $ps(C') = C$; (b) $\forall r_i' \in R_x', ps(r_i') = r$; (c) $\forall v_j' \in V_y', \exists r_i' \in R_x',\ ps(v_j') = v \wedge v_j' \in r_i'.V$; (d) $\forall s_k' \in S_z', \exists r_i' \in R_x', ps(s_k') = s \wedge\ s_k' \in pre\_s(r_i', C') \cup post\_s(r_i', C')$.
– *Artifact extension.* Let new artifact $C' \in Z'$ be added in Π' with a set of new business rules $R_x' \subseteq R' \setminus R$, a set of new tasks $V_y' \subseteq V' \setminus V$. The following statements hold: (a) $ps(C') = \varepsilon$; (b) $\forall r_i' \in R_x', ps(r_i') = \varepsilon$; (c) $\forall v_j' \in V_y', \exists r_i' \in R_x',\ ps(v_j') = \varepsilon$.

Note that if artifact, business rule, or task x: $x \in Z \cup V \cup R$ remains unchanged in the specialized model, then $ps(x) = x$.

3 Behavior-Consistent Process Specialization

3.1 Behavioral Properties of ACP

We first classify behavioral properties of ACP models into *intra-behavior* and *inter-behavior*. The *intra-behavior* of an artifact describes how an artifact changes its state throughout its lifecycle. Here, we use a deterministic finite state machine to capture the *lifecycle* of an individual artifact. The *inter-behavior* describes how a lifecycle of one artifact depends on the counterpart of another artifact, and it can be represented as state dependency (i.e., via a *sync rule*) between artifacts. Then, we discuss about the *soundness* property of individual artifact and the entire process which is constructed by composing all of its artifact lifecycles.

Definition 3: (Lifecycle). Let artifact class $C_i = (A_i, s_i^{init}, S_i, S_i^f)$ be in ACP model Π. The *lifecycle* of C_i, denoted as $\mathcal{L}_{C_i}$, can be defined as a tuple $(S, s^{init}, \Rightarrow)$ where set $S = S_i \cup S_i^f$, $s^{init} = s_i^{init}$, and transition relation $\Rightarrow \subseteq S \times R_i \times G_i \times S$ where $R_i \subseteq \Pi.R$ is a set business rules that are used to induce a state transition of artifact C_i, and G_i (guards) is a union set of state preconditions of each business rule in R_i such that each precondition references to a state of other artifact in Π.

Definition 4: (Sync rule). Given ACP model Π, a set of *sync rules* between lifecycles of artifacts $C_x \in \Pi.Z$ and $C_y \in \Pi.Z$ is denoted as $\varphi(\mathcal{L}_{C_x}, \mathcal{L}_{C_y}) = \{r \in \Pi.R \mid \exists(s_i, r, g, s_j) \in \mathcal{L}_{C_x}.\Rightarrow \wedge\ \exists(s_m, r, g, s_n) \in \mathcal{L}_{C_y}.\Rightarrow\}$.

Next, we define *ACP lifecycle* for describing the behavioral aspect of an ACP model consisting of synchronized lifecycles of artifacts. We adapt a state machine composition technique presented in [5] for generating the lifecycle of ACP.

Definition 5: (Lifecycle composition and composed lifecycle). Given ACP model Π, two lifecycles $\mathcal{L}_i = (S_i, s_i^{init}, \Rightarrow_i)$, and $\mathcal{L}_j = (S_j, s_j^{init}, \Rightarrow_j)$ can be composed, denoted as $\mathcal{L}_i \oplus \mathcal{L}_j$, into *composed lifecycle* $\mathcal{L}_c = (S_c, s_c^{init}, \Rightarrow_c)$, where $S_c \subseteq \mathcal{L}_i.\mathrm{S} \times \mathcal{L}_j.\mathrm{S}$ is a set of composed states, $s_c^{init} = (\mathcal{L}_i.s^{init}, \mathcal{L}_j.s^{init})$ is the initial state, and $\Rightarrow_c \subseteq S_c \times \Pi.R \times G_c \times S_c$ is transition relation where G_c is a set of guards. To formulate $\Rightarrow_c$ of *composed lifecycle* $\mathcal{L}_c$, a following set of three inference rules are required. Let $g[s_{\mathcal{L}_i}^x / state(\mathcal{L}_i,\ s_x)]$ denote that state $s_{\mathcal{L}_i}^x$ in guard g is substituted by true or false (of state predicate) depending on whether the local state of $\mathcal{L}_i$ is s_x.

$$\frac{(s_{\mathcal{L}_i}^x, r, g_1, s_{\mathcal{L}_i}^y) \in \Rightarrow_i}{\left(\left(s_{\mathcal{L}_i}^x, s_{\mathcal{L}_j}^x\right), r, g_c, \left(s_{\mathcal{L}_i}^y, s_{\mathcal{L}_j}^x\right)\right) \in \Rightarrow_c,\ g_c = g_1\left[s_{\mathcal{L}_j}^x / state(\mathcal{L}_j, s_x)\right]} \tag{1}$$

$$\frac{(s_{\mathcal{L}_j}^x, r, g_2, s_{\mathcal{L}_j}^y) \in \Rightarrow_j}{\left(\left(s_{\mathcal{L}_i}^x, s_{\mathcal{L}_j}^x\right), r, g_c, \left(s_{\mathcal{L}_i}^x, s_{\mathcal{L}_j}^y\right)\right) \in \Rightarrow_c,\ g_c = g_2\left[s_{\mathcal{L}_i}^x / state(\mathcal{L}_i, s_x)\right]} \tag{2}$$

$$\frac{(s_{\mathcal{L}_i}^x, r, g_1, s_{\mathcal{L}_i}^y) \in \Rightarrow_i \ \wedge\ (s_{\mathcal{L}_j}^x, r, g_2, s_{\mathcal{L}_j}^y) \in \Rightarrow_j}{\left(\left(s_{\mathcal{L}_i}^x, s_{\mathcal{L}_j}^x\right), r, g_c, \left(s_{\mathcal{L}_i}^y, s_{\mathcal{L}_j}^y\right)\right) \in \Rightarrow_c,\ g_c = g_1\left[s_{\mathcal{L}_j}^x / state(\mathcal{L}_j, s_x)\right] \wedge g_2\left[s_{\mathcal{L}_i}^x / state(\mathcal{L}_i, s_x)\right]} \tag{3}$$

Rule (1) and Rule (2) are applied when a business rule is fired on $\mathcal{L}_i$ and $\mathcal{L}_j$, respectively. Rule (3) is applied when a *sync rule* is fired on both $\mathcal{L}_i$ and $\mathcal{L}_j$.

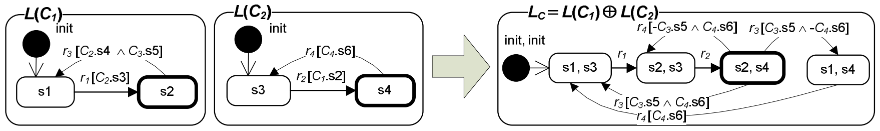


Fig. 3. An example of a lifecycle composition

Fig. 3 shows the composition between the lifecycle of artifact C_1 and the lifecycle of artifact C_2. We attach label $r_i[g]$ to a transition to mean that the transition is fired when both the attribute proposition in the pre-condition of business rule r_i holds and all state propositions (of external lifecycles) in g hold. We denote the counter state condition of $C.s_x$ by symbol $-C.s_x$ in the guard. Next, we define the lifecycle of ACP by using *lifecycle composition*. Given ACP model Π, an *ACP lifecycle* of Π, denoted as $\mathcal{L}_\Pi$, can be generated by iteratively performing *lifecycle composition* of every artifact in Π. For both artifact lifecycle and ACP lifecycle, we define *lifecycle occurrence* to refer to a particular sequence of states occurring from the *init* state to one of the *final* states of the lifecycle. Based on this, we define a *soundness* property to describe the desired and correct behavior of an artifact lifecycle and a process. Given ACP model Π, and *lifecycle* $\mathcal{L}$ (of either an artifact class or Π), a *lifecycle occurrence* is denoted as $\sigma = (s^{init}, ..., s_f)$ such that for every state $s \in \sigma$, there exists final state $s_f \in \mathcal{L}.S$ and s_f can be reached from s^{init} through s by a particular firing sequence of some business rules in R.

Definition 6: (Safe, Goal-reachable, and Sound lifecycle). Given ACP model Π, lifecycle $\mathcal{L} = (S, s^{init}, \Rightarrow)$ (of either an artifact class in $\Pi.Z$ or Π), we define sets of lifecycle states $S = \mathcal{L}.S \cup \{\mathcal{L}.s^{init}\}$ and final states $S^f \subseteq S$. Lifecycle $\mathcal{L}$ is said to be: (1) *safe* iff there exists business rule $r \in \Pi.R$ such that r induce one and only one transition in $\mathcal{L}$; (2) *goal-reachable* iff, for every non-final state s, there exists s in some lifecycle occurrence; and, for every final state $s_f \in S^f$, there exists occurrence σ such that s_f is the last state of σ; (3) *sound* iff $\mathcal{L}$ is *safe* and *goal-reachable.*

In the rest of paper, we restrict our discussion only to the *sound* behavior of artifacts and ACP based on their lifecycles (not the changes of artifact's data). However, discussions and formal approaches to data verification can be found in [2].

3.2 Behavior Consistent Specialization

In this section, we discuss the behavior consistency between a specialized ACP model and its base model when applying two methods of ACP specialization introduced in Section 2.2: *artifact refinement* and *artifact extension*. In object-oriented design approaches, the consistency of (dynamic) object behaviors between subtype and its supertype can be divided into observation consistency and invocation consistency. Observation consistency ensures that if features added at a subtype are ignored and features refined at a subtype are considered unrefined, any processing of an artifact of the subtype can be observed as correct processing from the view of the supertype. The invocation consistency refers to the idea that instances of a subtype can be used in the same way as instances of the supertype. More detailed discussion about object's behavior consistencies can be found in [4, 10]. In this article, we restrict our discussion of business process specialization to observation consistency (for both artifact and process). On one hand, in the viewpoint of structure, it is ensured that the current processing states of artifact and process are always visible at the higher (abstracted) organizational role. On the other hand, to preserve the behavior consistency it is guaranteed that business rules added at a subtype do not interfere with the business rules inherited from its supertype. Particularly, dealing with changes of synchronization dependencies between artifacts is a major technical issue of *ACP specialization*. Here, we consider *ACP specialization* for an entire process as the product of (1) the specialization of individual lifecycle (*lifecycle specialization*) and (2) the specialization of synchronizations (*sync specialization*).

Definition 7: (Lifecycle specialization). Let ACP model Π' be a specialization of ACP model Π with *ACP specialization* $ps_{\Pi' \to \Pi}$. Given lifecycle $\mathcal{L} = (S, s^{init}, \Rightarrow)$ (of artifact in Π or Π) and lifecycle $\mathcal{L}' = (S', s^{init'}, \Rightarrow')$ (of artifact in Π' or Π'), we define lifecycle specialization relation between $\mathcal{L}$ and $\mathcal{L}'$ based on $ps_{\Pi' \to \Pi}$ by *lifecycle specialization* (*total*) *function* $ls_{\mathcal{L}' \to \mathcal{L}} : S' \cup s^{init'} \cup \Rightarrow' \ \to S \cup s^{init} \cup \Rightarrow \cup \{\varepsilon\}$.

For ACP specialization method by the *refinement* of artifact class, a single state (or a transition) is refined into a set of sub states and sub transitions. Here, we define a fragment of lifecycle, called *L-fragment*, which contains a set of sub states and sub transitions for capturing the refinement, e.g., in Fig. 2, L-fragment l_3 refines a transition of *Invoice* and L-fragment l_1 refines the *created* state of *Purchase Order*. Then, we use *lifecycle specialization function* ls to project every state and transition of a fragment in a specialized lifecycle onto a state or a transition of its base lifecycle.

Definition 8: (L-fragment). Given ACP model Π, L-fragment $\ell^{\mathcal{L}x}$ of lifecycle $\mathcal{L}_x$ is a nonempty connected sub-lifecycle of $\mathcal{L}_x$. It can be defined as $\ell^{\mathcal{L}x} = (S, \Rightarrow, \Rightarrow^{\boldsymbol{in}}, \Rightarrow^{\boldsymbol{out}})$ where,

- $S \subseteq \mathcal{L}_x.S \setminus \{s^{init}\} \cup S^f$ is a non-empty set of states of $\ell^{\mathcal{L}x}$, where S^f is a set of *final* states of $\mathcal{L}_x$,
- $\Rightarrow \subseteq S \times R^{\mathcal{L}x} \times G^{\mathcal{L}x} \times S \subseteq \mathcal{L}_x.\Rightarrow$ is a set of transitions of $\ell^{\mathcal{L}x}$, where $R^{\mathcal{L}x}$ and $G^{\mathcal{L}x}$ are subsets of business rules and guards, respectively,
- $\Rightarrow^{in} = \mathcal{L}_x.\Rightarrow \cap \ ((\mathcal{L}_x.S \backslash S) \times R^{\mathcal{L}x} \times G^{\mathcal{L}x} \times S))$ is a set of *entry transitions*,
- $\Rightarrow^{out} = \mathcal{L}_x.\Rightarrow \cap \ (S \times R^{\mathcal{L}x} \times G^{\mathcal{L}x} \times (\mathcal{L}_x.S \backslash S))$ is a set of *exit transitions*,
- such that, for every state s in $\ell^{\mathcal{L}x}.S$, there exists a sequence of transitions from some entry transition in $\Rightarrow^{in}$ to s and from s to some exit transition in $\Rightarrow^{out}$.

Now, we apply *L-fragment* to the *lifecycle specialization*. Let lifecycle $\mathcal{L}' = (S', s^{\boldsymbol{init'}}, \Rightarrow')$ be a specialization of lifecycle $\mathcal{L} = (S, s^{init}, \Rightarrow)$ by *lifecycle specialization* $ls_{\mathcal{L}' \to \mathcal{L}}$. We denote a set of *refined L-fragments* that are used to refine $\mathcal{L}$ as $lf(\mathcal{L}' \to \mathcal{L}) = \{\ell_1, \ell_2, \ldots, \ell_y\}$ where $\ell_i (1 \leq i \leq y)$ is an L-fragment in $\mathcal{L}'$ such that ℓ_i does not exist in $\mathcal{L}$.

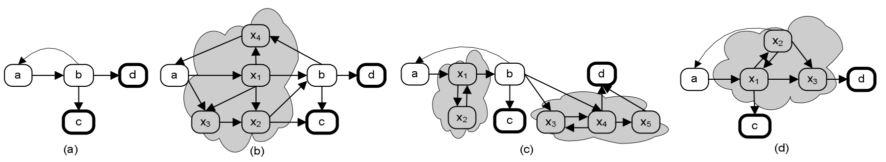


Fig. 4. Examples of L-fragments

For example in Fig. 4, lifecycles (b), (c), and (d) are different specializations of lifecycle (a). Lifecycles (b) and (c) refine some transitions of lifecycle (a), while lifecycle (d) refines only state b of lifecycle (a). Next, we want to check whether the behavior of specialized lifecycle $\mathcal{L}_y$is consistent to the behavior of its base lifecycle $\mathcal{L}_x$. It is understandable that if every L-occurrence of $\mathcal{L}_x$, disregarding the states and transitions in a refined L-fragment, is observable as the same sequence as of $\mathcal{L}_y$, then the behavior of $\mathcal{L}_y$ is *consistent* to the behavior of $\mathcal{L}_x$. Here, we define *behavior-consistency* (*B-consistency*) property between two lifecycles to describe the condition to preserve the consistency between them. Our *B-consistency* relation between two lifecycles can be defined by adopting the notion of bi-simulation equivalence relation in process algebras. By replacing a *silent* (τ) *action* for a refined L-fragment in the specialized lifecycle, we can apply weak bi-simulation to compare two lifecycles.

Definition 9: (*B-consistent*). Let lifecycle $\mathcal{L}_y = (S_y, {s_y}^{init}, \Rightarrow_y)$ and lifecycle $\mathcal{L}_x = (S_x, {s_x}^{init}, \Rightarrow_x)$ such that $\mathcal{L}_y$ specializes $\mathcal{L}_x$ with *lifecycle specialization* $ls_{\mathcal{L}_y \rightarrow \mathcal{L}_x}$, and $S_{x\cap y} = S_x \cap S_y$ be a set of states that exist in both $\mathcal{L}_x$ and $\mathcal{L}_y$. We have $\mathcal{L}_y$ *B-consistent* to $\mathcal{L}_x$ iff$\forall s_i, s_j \in S_{x\cap y}, \exists (s_i, r, g, s_j) \in \Rightarrow_x, \forall s_k \in S_y \backslash S_{x\cap y}, s_i \overset{*}{\Rightarrow}_y s_k \wedge s_k \overset{*}{\Rightarrow}_y s_j$, where $\overset{*}{\Rightarrow}$ is denoted for a reflexive transitive closure of $\Rightarrow$. We also say that $ls_{\mathcal{L}_y \rightarrow \mathcal{L}_x}$ is *B-consistent.*

For instance, the lifecycle in Fig. 4 (b) is not *B-consistent* to the lifecycle in Fig. 4 (a). This is because, in some lifecycle occurrences of lifecycle (b), state *a* can reach state *c* (through state x_2) without passing state *b*; and, state *a* can reach itself via state x_4 without passing state *b*. In contrast, we can see that $ls_{\mathcal{L}_c \rightarrow \mathcal{L}_a}$in Fig. 4 (c) and $ls_{\mathcal{L}_d \rightarrow \mathcal{L}_a}$ in Fig. 4 (d) are *B-consistent.* Note that we can also apply *B-consistency* for the case of L-fragments. Now, we define L-fragment with a single entry and a single exit state as *atomic L-fragment* (*AL-fragment*) and show how it is considered for the behavioral consistency between two lifecycles.

Definition 10: (AL-fragment). Given ACP model Π and *L-fragment* $\ell = (S, \Rightarrow, \Rightarrow^{in}, \Rightarrow^{out})$ of lifecycle $\mathcal{L}_x = (S, s^{init}, \Rightarrow)$, ℓ_i is called *AL-fragment* iff for every entry transition $\Rightarrow_m \in \ell.\Rightarrow^{in}$, $\Rightarrow_m$ is fired from same source state $s_x \in \mathcal{L}.S \backslash \ell.S$; and, for every exit transition $\Rightarrow_n \in \ell.\Rightarrow^{out}$, $\Rightarrow_n$ is fired to same target state $s_y \in \mathcal{L}.S \backslash \ell.S$.

Theorem 1: Let lifecycle $\mathcal{L}'$ be a specialization of lifecycle $\mathcal{L}$ with a set of *refined L-fragments* $lf(\mathcal{L}' \rightarrow \mathcal{L})$, $ls_{\mathcal{L}' \rightarrow \mathcal{L}}$ is *B-consistent* if, for every $\ell_i \in lf(\mathcal{L}' \rightarrow \mathcal{L})$,

- if ℓ_i refines transition $s_x \Rightarrow_t s_y \in \mathcal{L}.\Rightarrow$ then ℓ_i is an *AL-fragment*; or,
- if ℓ_i refines state $s \in \mathcal{L}.S$ then, for every instate $s_x \in \mathcal{L}.S$ fired to s and for outstate $s_y \in \mathcal{L}.S$ fired from s, s_x can reach s_y in some *L-occurrences* of ℓ_i.

Revisiting Fig. 2, *Offline PO* (with L-fragments ℓ_1 and ℓ_2) and *Offline Invoice* (with L-fragment ℓ_3) are *B-consistent* to *Purchase Order* and *Invoice*, respectively. Next, we define *B-consistent specialization* for both an artifact and a process.

Definition 11: (B-consistent specialization). Given ACP model Π' be a specialization of ACP model Π with ACP specialization $ps_{\Pi' \to \Pi}$, Π' is a *B-consistent specialization* of Π iff $ls_{\mathcal{L}'_\Pi \to \mathcal{L}_\Pi}$ is *B-consistent*. Similarly, we say artifact C' in Π' is a *B-consistent specialization* of artifact C in Π iff $ls_{\mathcal{L}'_C \to \mathcal{L}_C}$ is *B-consistent*.

3.3 Specialization of Synchronization Dependencies

This section discusses how changes of artifact interactions (through their synchronization dependencies) affect the behavior of the process in their specialization at both the artifact level and the process level. We classify specialization of synchronizations into two methods: *sync extension* and *sync refinement*. First, *sync extension* is a method of synchronizing new artifact with an existing artifact without refining any existing sync rule. However, it is achieved by adding a new defined set of sync rules, called *extended sync rules*. Second, *sync refinement* is a method to decompose an individual existing sync rule in the base process to a new set of refined sync rules in the specialized process. A specialized sync rule can be used to synchronize between existing artifacts or between existing artifact(s) and new (extended) artifact(s) added to the specialized process. Fig. 5 shows an abstracted example of results after applying different sync specialization methods to the base process (a). More discussions on this example will appear through the rest of the paper.

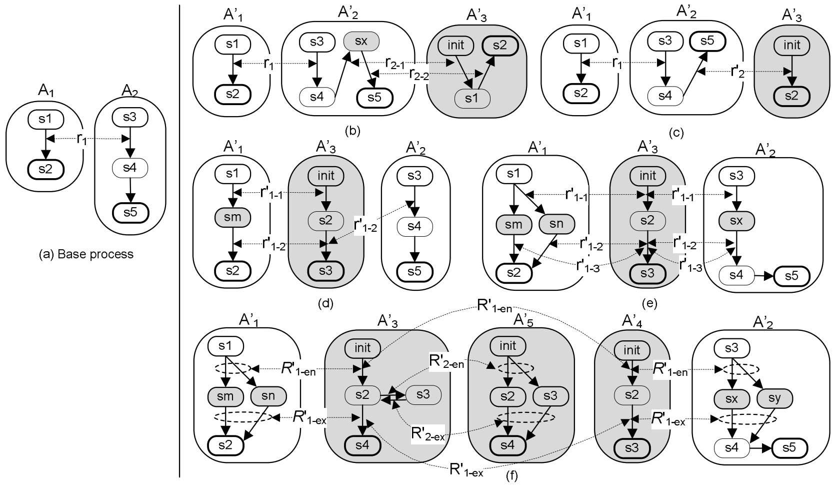


Fig. 5. Examples of sync specializations between artifacts

The consistency of synchronization dependencies in process specialization means whatever changes made to the synchronization of artifacts the behavior of the *composed lifecycle* of such specialized artifacts must be consistent to their composition in the base process. Particularly, adding a new artifact into a specialized process results unobservable behavior of itself in the base process. However, it is desirable that the overall behaviors of such base and specialized processes with added artifacts remain consistently observable. For instance, in Fig. 2, the *Quote* artifact added to the *Offline* ordering process should not interfere with the behavior of the *Purchase Order, Shipping Order* and *Invoice* artifacts and their interactions in its base ordering process. Next, we define the specialization of the synchronization between two lifecycles followed by detailed discussion on how synchronization is consistently handled when applying each of the two sync operations.

Definition 12: (Sync specialization). Let artifact lifecycles $\mathcal{L}'_{C_x}$ and $\mathcal{L}'_{C_y}$ in specialized ACP model Π' be a *B-consistent specialization* of artifact lifecycles $\mathcal{L}_{C_x}$ and $\mathcal{L}_{C_y}$ in base ACP model Π, respectively. We define *sync specialization* $ss^{\Pi' \to \Pi}{}_{(\mathcal{L}'_{C_x}, \mathcal{L}'_{C_y}) \to (\mathcal{L}_{C_x}, \mathcal{L}_{C_y})} : \varphi(\mathcal{L}'_{C_x}, \mathcal{L}'_{C_y}) \to \varphi(\mathcal{L}_{C_x}, \mathcal{L}_{C_y}) \cup \{\varepsilon\}$ as a total function that projects a specialized sync rule between $\mathcal{L}'_{C_x}$ and $\mathcal{L}'_{C_y}$ onto its base sync rule between $\mathcal{L}_{C_x}$ and $\mathcal{L}_{C_y}$ or *empty element* ε.

Now, in order to capture and analyze synchronizations between two lifecycles we extend the definition of AL-fragment of isolated lifecycle to *atomic synchronized L-fragment*, called *ASL-fragment*, between two lifecycles.

Definition 13: (*ASL-fragment*). Given ACP model Π, let L-fragment ℓ^{C_x} in $\mathcal{L}_{C_x}$of artifact $C_x \in \Pi.Z$ synchronize with L-fragment ℓ^{C_y} in $\mathcal{L}_{C_y}$of artifact $C_y \in \Pi.Z$ via business rules $R^{sync} \in \Pi.R$. We identify ℓ^{C_x} and ℓ^{C_y} as *ASL-fragments* iff, for every $\ell^{C_i} \in \{\ell^{C_x}, \ell^{C_y}\}$,

– ℓ^{C_i} is an *AL-fragment*,
– $\forall r \in \varphi(\mathcal{L}_{C_x}, \mathcal{L}_{C_y}), \exists s_s \in \ell^{C_i}.S, s_s \overset{r}{\Rightarrow} s \in \mathcal{L}_{C_i}.\Rightarrow \to s_s \overset{r}{\Rightarrow} s \in \ell^{C_i}.\Rightarrow \wedge\ r \in R^{sync}$,
– $\forall r \in \varphi(\mathcal{L}_{C_x}, \mathcal{L}_{C_y}), \exists s_t \in \ell^{C_i}.S,\ s \overset{r}{\Rightarrow} s_t \in \mathcal{L}_{C_i}.\Rightarrow \to s \overset{r}{\Rightarrow} s_t \in \ell^{C_i}.\Rightarrow \wedge\ r \in R^{sync}$,
– $\forall \mathcal{L}_{C_j}(C_j \in \Pi.\mathrm{Z} \backslash \{C_x, C_y\}), \varphi(\ell^{C_i}, \mathcal{L}_{C_j}) = \emptyset$.

Note that the conditions for ASL-fragment are used to restrict two synchronized L-fragments to include every transition and corresponding sync rule that are used for only the synchronization between L-fragments. This is because we want to guarantee that the composition of two ASL-fragments have all entry transition fired from the same (composite) source state and all exit transitions fired to the same (composite) target state, i.e., both the composition and ASL-fragments are atomic.

For example, both L-fragments l_1 and l_2 in Fig. 6 (a) are ASL-fragments containing all related sync rules (r_1 and r_2) used for l_1 and l_2. As such, the composition between l_1 and l_2 is then atomic, as shown in Fig. 6 (c). In contrast, in Fig. 6 (b), we can see that L-fragment l_4 cannot satisfy the property of AL-fragment, and L-fragment l_3 does not include transition $s_2 \overset{r_4}{\Rightarrow} s_9$ where sync rule r_4 exits in entry transition $s_5 \overset{r_4}{\Rightarrow} s_6$ of l_4. Therefore, l_3 and l_4 are not ASL-fragments. Next, we discuss how we use ASL-fragments to induce the *B-consistency* of the specialization of synchronization of such two fragments.

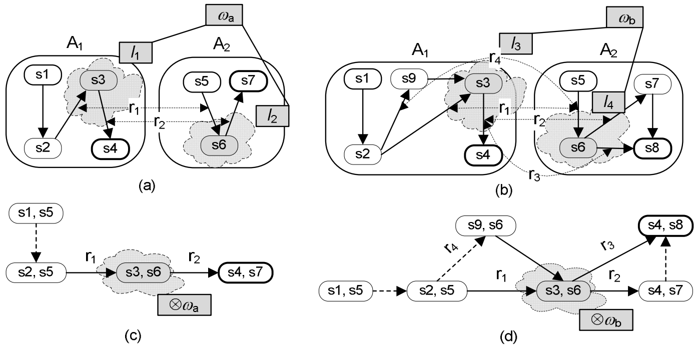


Fig. 6. Examples of the composition of synchronized L-fragments

3.3.1 Sync Extension

With an extension of any new synchronized artifact to the specialized process, we need to guarantee that the consistency is not interfered by the behavior of such artifact. This can be achieved by checking whether a lifecycle of an extended artifact can be completely composed within an embedded lifecycle of an artifact it synchronizes with, as shown in Definition 14 and Lemma 1.

Definition 14: (*ex-lifecycle and* $\sqsupset$). Let lifecycle $\mathcal{L}'_{C_x}$ in specialized ACP model Π' be *B-consistent* to lifecycle $\mathcal{L}_{C_x}$ in base ACP model Π, and lifecycle $\mathcal{L}'_{C_y}$ of extended artifact C'_y in Π' synchronize $\mathcal{L}'_{C_x}$. We say $\mathcal{L}'_{C_y}$ as an *ex-lifecycle* of $\mathcal{L}'_{C_x}$ if there exists refined L-fragment $\ell_i \in lf(\mathcal{L}'_{C_x} \rightarrow \mathcal{L}_{C_x})$ such that $\mathcal{L}'_{C_y}$ has its whole lifecycle synchronized within ℓ_i, denoted $\ell_i \sqsupset \mathcal{L}'_{C_y}$.

Lemma 1: Based on Definition 12, given refined L-fragment $\ell \in lf(\mathcal{L}'_{C_x} \rightarrow \mathcal{L}_{C_x})$ synchronize with extended lifecycle $\mathcal{L}'_{C_y}$, the composed lifecycle between $\mathcal{L}'_{C_y}$ and ℓ is *B-consistent* to ℓ iff $\ell \sqsupset \mathcal{L}'_{C_y}$ and ℓ is an *ASL-fragment.*

For example, extended artifact A'_3 in Fig. 5 (b) has its whole lifecycle synchronized within artifact A'_2. This case can be explained in more detail by using Fig. 6 (a). We can see that L-fragment ℓ'_1 syncrhonizes with ℓ'_2 which represents the whole lifecycle of A'_3 ($\ell'_1 \sqsupset A'_3$); therefore, A'_3 is an *ex-lifecycle* of artifacts A'_2 and we have the composed lifecycle between ℓ'_1 and ℓ'_2 *B-consistent* to ℓ'_1. One can question that what would be the result if an extended artifact is synchronized with more than one existing artifact. For instance, in Fig. 5 (d), where two existing artifacts A'_1 and A'_2 synchronize with extended artifact A'_3. It is possible to see that the lifecycle of A'_3 is an *ex-lifecycle* of the lifecycle of A'_1 while it does not hold for

A'_2. Based on Definition 13, although the condition of ex-lifecycle is satisfied for the synchronization between A'_3 and A'_1, however, it is not held for the synchronization between A'_3 and A'_2. Therefore, the result of iterative composition of such three lifecycles should not satisfy Lemma 1.

3.3.2 Sync Refoinement

– Refinement of synchronization for existing artifacts

We classify specialization patterns of the synchronization between two existing artifacts into two cases. First, one of two artifacts is refined while the other one remains unrefined. Second, both artifacts are refined. With the first case, the effected sync rule(s) of the refinement may have its state condition redefined on either the entry transition or the exit transition of an L-fragment. For the second case, both artifacts have their L-fragment refined. For example, in Fig. 7 (b), sync rules r'_{1-1} and r'_{1-2} are redefined for the exit transition of states s_m and s_n. For the second case, both artifacts have their L-fragment refined, e.g., Fig. 7 (c) and (d).

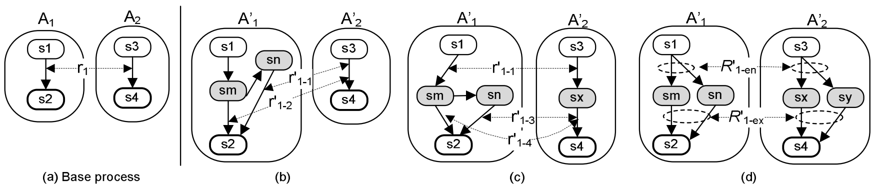


Fig. 7. Sync specializations of existing artifacts

For the refinement of two existing artifacts, we can apply the notion of *ASL-fragments* to check whether the refinement of these artifacts preserves the *B-consistency* of the base process. However, for single artifact refinement, we consider it as a special case since the refinement is applied on a single transition of one artifact not L-fragment. In order to make the transition to qualify L-fragment, so we expand its boundary to cover the source and target states of the transition. Then we can validly apply the ASL-fragments to check *B-consistency*. For instance, in Fig. 7 (b), we have an expanded L-fragment in artifact A'_2, which consists of states s_3 and s_4, synchronizes with L-fragment of A'_2.

Lemma 2: Let artifact lifecycles $\mathcal{L}'_{C_x}$ and $\mathcal{L}'_{C_y}$ in specialized ACP model Π' be *B-consistent* to artifact lifecycles $\mathcal{L}_{C_x}$ and $\mathcal{L}_{C_y}$ in base ACP model Π. Given L-fragment ℓ_m refines transition $\Rightarrow_i$ in $\mathcal{L}_{C_x}$ and L-fragment ℓ_n refines transition $\Rightarrow_j$ in $\mathcal{L}_{C_y}$, if ℓ_m and ℓ_n are *ASL-fragments*, then the composed lifecycle of ℓ_m and ℓ_n is *B-consistent* to the composed lifecycle of $\Rightarrow_i$ and $\Rightarrow_j$ (both $\Rightarrow_i$ and $\Rightarrow_j$ are considered as L-fragments with one transition).

– Refinement of synchronization for extended artifacts

Now, we extend the sync refinement between existing artifacts to be able to consider synchronizations between existing and extended artifacts. Recall sync extension, an extended artifact can be considered as an *ex-lifecycle* of an existing artifact if the lifecycle of the extended artifact is entirely synchronized within such existing artifact. We can say that if extended artifact C is used to refine sync rule r, then each artifact that is synchronized by r must have C as its ex-lifecycle. For example in Fig. 5 (d), artifact A'_3 is used to refine sync rule r_1 (between A'_1 and A'_2), and it can be considered as *ex-lifecycle* of both artifacts. A similar case is shown in Fig. 5 (e).

We now consider the scenario that has to deal with synchronizations for multiple extended artifacts, e.g., extended artifact A'_5 in Fig. 5 (f). Similar to the refinement between an existing artifact and an extended artifact, here we extend the sync extension method and *B-consistency* checking to the synchronization for multiple extended artifacts by introducing *transitivity* of ex-lifecycles. We say $\mathcal{L}'_{C_z}$ as a *transitive ex-lifecycle* of $\mathcal{L}'_{C_x}$ if $\mathcal{L}'_{C_z}$ is an ex-lifecycle of $\mathcal{L}'_{C_y}$ and $\mathcal{L}'_{C_z}$ is an ex-lifecycle of $\mathcal{L}'_{C_x}$. Here, we write $\ell_i \sqsupset^+ \mathcal{L}'_{C_z}$ if there exists refined L-fragment $\ell_i \in lf(\mathcal{L}'_{C_x} \to \mathcal{L}_{C_x})$ such that $\ell_i \sqsupset \mathcal{L}'_{C_y}$ and $\mathcal{L}'_{C_z}$ is an ex-lifecycle of $\mathcal{L}'_{C_y}$. For instance, artifact A'_5 in Fig. 5 (f) has its whole lifecycle synchronized within the lifecycle of artifacts A'_3, and A'_3 is an ex-lifecycle of A'_1; so, we have that A'_5 is a *transitive ex-lifecycle* of A'_1. Now, we show how the *B-consistency* of the refinement for extended artifacts can be preserved in Lemma 3.

Lemma 3: Let artifact lifecycles $\mathcal{L}'_{C_x}$ and $\mathcal{L}'_{C_y}$ in specialized ACP model Π' be *B-consistent* to artifact lifecycles $\mathcal{L}_{C_x}$ and $\mathcal{L}_{C_y}$ in base ACP model Π, and let L-fragment ℓ_m refines transition $\Rightarrow_i$ in $\mathcal{L}_{C_x}$ and L-fragment ℓ_n refines transition $\Rightarrow_j$ in $\mathcal{L}_{C_y}$ such that ℓ_m and ℓ_n are *ASL-fragments*. Let extended lifecycle $\mathcal{L}'_{C_x}$ synchronize with ℓ_m or ℓ_n and a set of extended lifecycles Z^{ex} synchronize with $\mathcal{L}'_{C_x}$. The composed lifecycle of all artifacts in Z^{ex}, $\mathcal{L}'_{C_x}$, ℓ_m, and ℓ_n is *B-consistent* to the composed lifecycle of $\Rightarrow_i$ and $\Rightarrow_j$ iff $\forall \mathcal{L}'_{C_i}\{C'_i \in Z^{ex}\}, \ell_m \sqsupset^+ \mathcal{L}'_{C_i} \wedge \ell_n \sqsupset^+ \mathcal{L}'_{C_i}$.

For example in Fig. 5 (e), the composed lifecycle of artifacts A'_1, A'_2, and , A'_3 is *B-consistent* to the composed lifecycle of A'_1 and A'_2 since A'_3 is an *ex-lifecycle* of both A'_1 and A'_2. More complicated case is shown in Fig. 5 (f) having artifact A'_5 extended to artifact A'_3 which is used for the sync refinement of artifacts A'_1 and A'_2. We can see that A'_5 is considered as a *transitive ex-lifecycle* of A'_1 and A'_2; therefore, this refinement preserves the *B-consistency* of the base process.

3.4 Sync Specialization and B-Consistency

Based on our comprehensive discussion on the two operations of sync specialization and their individual consistency and the *B-consistency* of ACP models, we now are able to define a complete consistency property of sync specialization.

Definition 17: (Synchronization consistent or *S-consistent*). Given ACP model Π' be a specialization of ACP model Π with ACP specialization $ps_{\Pi' \to \Pi}$ and sync specialization $ss_{(\mathcal{L}'_{C_x}, \mathcal{L}'_{C_y}) \to (\mathcal{L}_{C_x}, \mathcal{L}_{C_y})}$, $ss_{(\mathcal{L}'_{C_x}, \mathcal{L}'_{C_y}) \to (\mathcal{L}_{C_x}, \mathcal{L}_{C_y})}$ is said to be

S-consistent iff, Lemma 1 is held for *sync extension*, and Lemmas 2 and 3 are held for *sync refinement.*

Theorem 2: Let ACP model Π' specialize ACP model Π with *ACP specialization* $ps_{\Pi' \to \Pi}$. Then, Π' is a *B-consistent specialization* of Π based on ps iff,

- for every artifact $C'_i \in \Pi'.Z$ such that C'_i specializes $C_i \in \Pi.Z$, $ls_{\mathcal{L}'_{C_i} \to \mathcal{L}_{C_i}}$ is *B-consistent*; and,
- for every artifact C'_x and C'_y in Π', $ss_{(\mathcal{L}'_{C_x}, \mathcal{L}'_{C_y}) \to (\mathcal{L}_{C_x}, \mathcal{L}_{C_y})}$ is *S-consistent.*

Theorem 2 has an importance of being able to assert the overall behavioral consistency between a specialized ACP model and its base model while only perform fragmental consistency checking based on a specialization, i.e., for an individual artifact and for only a synchronization between artifacts that is added or modified in the specialized process. Notably, the model verification can suffer from the state exposition of compositional lifecycle if there are a number of artifacts having many states. Technically, we avoid the state space exposition problem by not composing all artifacts in the model.

4 Related Work and Discussion

The concept of business artifacts was introduced in [1] with the modeling concept of artifact lifecycles. Bhattacharya et al. [2] presented an artifact-centric process model with the study of necessary properties such as reachability of goal states, absence of deadlocks, and redundancy of data. Kuster et al. [6] presented a notion of compliance of a business process model with object lifecycles and a technique for generating the model from such set of lifecycles. Yongchareon and Liu [3, 11] proposed a process view framework to allow role-based customization and inter-org process modeling for artifact-centric business processes. In chorography settings, Van Der Aalst et al. [12] proposed an inter-org process-oriented contract with a criterion for accordance between private view and its public view modelled by open nets (oWFNs). Lohmann and Wolf [7] studied the generation of the interaction model from artifact-centric process models and used artifact composition to validate the model; and later, Lohamnn [13] proposed an approach to generate complaint and operational process model using policies and compliance rules. Fahland et al [16] presented conformance checking technique for interacting artifacts by decomposition into smaller problems so that conventional techniques can apply. Compared to our work, we also use similar composition technique to validate the overall behavior of the model; however, we focus on the fragmental behavior analysis for different methods of sync specializations (extension and refinement).

Schrefl and Stumptner [4] studied the consistency criteria of the inheritance of object life cycles. They proposed necessary and sufficient rules for checking behavior consistency between object lifecycles. Some works have attempted to tackle the specialization of processes using state diagrams [8], the inheritance of (Petri-net based) workflow [10], and the behavior compatibility (consistency) between process models [14, 15]. However, these works only focused on the inheritance of single object lifecycle or workflow model. We extend their study to the synchronization between lifecycles. Although [9] claimed that a specialization of processes cannot be viewed and treated analogously as a specialization of a single object, their work mainly treated the behavior of a process as the behavior of a single (dataflow) diagram. This approach still lacks detailed discussion and analysis of how objects and their interactions are considered in a specialized process, while our work takes into account the specialization of synchronizations between objects. In artifact-centric setting, Calvanese et al. [21] addressed the problem of comparing artifact-centric workflows by proposing a notion of dominance between workflows that captures the fact that all executions of one workflow can be emulated by another workflow. Their work focused on the initial and final snapshots of the workflow execution to be compared and did not take the behavior of artifact and process into account.

5 Conclusion and Future Work

This paper formally proposes the notion of process specialization for artifact-centric business processes with a comprehensive analysis of the behavioral consistency between a specialized process and its base process. For artifact-centric models, not only a local behavior of artifact but also the interaction behavior, which is described by sync business rules, can be specialized. One main outcome of this paper is the formal studies on the conditions for preserving the behavior consistencies of both intra-behavior and inter-behavior of artifacts in a specialized process based on our two proposed specialization methods (extension and refinement). In the future, we will develop an efficient mechanism and a prototype for the consistency checking based on our proposed theorems.

Acknowledgments. This work was partially supported by the Australian Research Council Linkage Project under the grant number LP0990393.

References

1. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
2. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
3. Yongchareon, S., Liu, C.: A Process View Framework for Artifact-Centric Business Processes. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 26–43. Springer, Heidelberg (2010)
4. Schrefl, M., Stumptner, M.: Behavior-Consistent Specialization of Object Life Cycles. ACM Transactions on Software Engineering and Methodology 11, 92–148 (2002)
5. Lind-Nielsen, J., Andersen, H., Hulgaard, H., Behrmann, G., Kristoffersen, K., Larsen, K.: Verification of Large State/Event Systems Using Compositionality and Dependency Analysis. Formal Methods in System Design 18(1), 5–23 (2001)

6. Küster, J.M., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
7. Lohmann, N., Wolf, K.: Artifact-Centric Choreographies. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 32–46. Springer, Heidelberg (2010)
8. Wyner, G.M., Lee, J.: Process Specialization: Defining Specialization for State Diagrams. Computational & Mathematical Organization Theory 8, 133–155 (2002)
9. Wyner, G.M., Lee, J.: Defining specialization for dataflow diagrams. Information Systems 28, 651–671 (2003)
10. van der Aalst, W.M.P., Basten, T.: Inheritance of workflows: an approach to tackling problems related to change. Theoretical Computer Science 270, 125–203 (2002)
11. Yongchareon, S., Liu, C., Zhao, X.: An Artifact-Centric View-Based Approach to Modeling Inter-organizational Business Processes. In: Bouguettaya, A., Hauswirth, M., Liu, L. (eds.) WISE 2011. LNCS, vol. 6997, pp. 273–281. Springer, Heidelberg (2011)
12. van der Aalst, W.M.P., Lohmann, N., Masuthe, P., Stahl, C., Wolf, K.: Multipart Contrats: Agreeing and Implementing Interorganizational Processes. The Computer Journal 53(1), 90–106
13. Lohmann, N.: Compliance by Design for Artifact-Centric Business Processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 99–115. Springer, Heidelberg (2011)
14. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. IEEE Transactions on Software Engineering 37(3), 410–429 (2011)
15. Weidlich, M., Dijkman, R., Weske, M.: Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 78–94. Springer, Heidelberg (2010)
16. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011)
17. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
18. Rosa, M.L., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J.: APROMORE: An advanced process model repository. Expert Systems with Applications 38, 7029–7040 (2011)
19. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
20. 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)
21. Calvanese, D., De Giacomo, G., Hull, R., Su, J.: Artifact-Centric Workflow Dominance. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 130–143. Springer, Heidelberg (2009)

Approximate Clone Detection in Repositories of Business Process Models

Chathura C. Ekanayake[1], Marlon Dumas[2], Luciano García-Bañuelos[2],
Marcello La Rosa[1], and Arthur H.M. ter Hofstede[1,3]

[1] Queensland University of Technology, Australia
{c.ekanayake,m.larosa,a.terhofstede}@qut.edu.au
[2] University of Tartu, Estonia
{marlon.dumas,luciano.garcia}@ut.ee
[3] Eindhoven University of Technology, The Netherlands

Abstract. Evidence exists that repositories of business process models used in industrial practice contain significant amounts of duplication. This duplication may stem from the fact that the repository describes variants of the same processes and/or because of copy/pasting activity throughout the lifetime of the repository. Previous work has put forward techniques for identifying duplicate fragments (clones) that can be refactored into shared subprocesses. However, these techniques are limited to finding exact clones. This paper analyzes the problem of approximate clone detection and puts forward two techniques for detecting clusters of approximate clones. Experiments show that the proposed techniques are able to accurately retrieve clusters of approximate clones that originate from copy/pasting followed by independent modifications to the copied fragments.

1 Introduction

Duplication is a widespread phenomenon in software and model repositories [7,12]. Not surprisingly, significant amounts of duplication can also be found in repositories of business process models used in industrial practice – both in the form of exact duplicates (a.k.a. *exact clones*) [16] and pairs of similar fragments (*approximate clones*) [4].[1] Clones in process model repositories emerge for example as a result of copy/pasting activity, but also when multiple variants of a process co-exist and are described as separate models. For example, a large insurance company typically runs multiple claims handling processes for different types of claims or products. Naturally, these process variants share some commonalities, which manifest themselves in the form of clones.

Detecting clones in process model repositories allows analysts to identify opportunities for standardization and refactoring. For example, given that disbursing occurs in multiple variants of a claims handling process, process fragments corresponding to disbursing can potentially be standardized and encapsulated in a shared subprocess. In previous work, we proposed a technique for identifying exact clones that can be refactored into shared subprocesses [16]. However, when clones emerge as a result of copy/pasting, it is likely that these clones will subsequently undergo independent changes and thereon can no longer be detected using exact clone detection methods.

[1] The term fragment is used to refer to a connected subgraph of a process model.

When designing approximate clone detection methods, a first step is to define what an approximate clone is. Generally, such a definition relies on a similarity or (equivalently) a distance metric. Previous work has shown that graph-edit distance is a suitable proxy for perceived process model dissimilarity [3]. Accordingly, we postulate that a necessary condition for two process model fragments to be approximate clones is that their graph-edit distance is below a user-defined threshold. However, three additional issues ought to be considered when defining a notion of approximate clone.

Firstly, it should be considered that any fragment g_1 is similar to any fragment g_2 such that g_2 contains g_1 or g_1 contains g_2, provided that the difference between g_1 and g_2 falls below the threshold. A definition that would consider two fragments as approximate clones merely because one contains the other would lead to many false positives (e.g. in the SAP reference model there are 8,876 fragments with 13,131 containment relations); this is an issue that has been widely discussed in the field of code and model clone detection [12]. Secondly, given the goal to identify approximate clones for the sake of refactoring them into subprocesses and given that subprocesses are invoked according to a call-and-return semantics, it is necessary that the approximate clones we retrieve are Single-Entry, Single-Exit (SESE) fragments. Thirdly, we are not interested in *trivial* clones consisting of a single activity, since they do not represent an opportunity for subprocess extraction. These considerations lead to the following definition.

Definition 1. *Given a distance metric Dist and a distance threshold* τ*, two non-trivial, SESE process model fragments* g_1 *and* g_2 *are approximate fragments – written* $Approx(g_1, g_2)$ *– iff* $g_1 \not\subset g_2$*,* $g_2 \not\subset g_1$ *and* $Dist(g_1, g_2) \leq \tau$*.*

Armed with this definition, one can retrieve large amounts of approximate clone pairs [4]. However, if the goal is to help analysts to identify opportunities for refactoring and standardization, retrieving all such pairs is of limited use. Instead, given the goal at hand, analysts need to identify sets of fragments C that can be standardized towards a single fragment with a bounded amount of changes on each fragment. Otherwise, some fragments would need to undergo changes during the standardization that would convert them into arbitrarily different fragments. In this respect, we envisage two alternative approaches to standardize a set of fragments:

– A set of fragments can be standardized by taking a given "medoid"[2] fragment as a reference and standardizing all fragments towards this medoid.
– A set of fragments can be standardized by selecting any fragment in the group as a reference and standardizing all other fragments towards this reference fragment.

This leads to the following definition.

Definition 2. *A set of SESE process model fragments* C *is a* cluster of approximate clones *iff one of the following properties holds:*

1. $\exists g \in C\ \forall g' \in C : Approx(g, g')$. *In this case,* g *is called the* cluster medoid.
2. $\forall g, g' \in C : Approx(g, g')$.

[2] In data clustering, a medoid is a representative object of a cluster, i.e. an object whose average dissimilarity to all other objects in the cluster is minimal.

The main contribution of this paper are two techniques (one per standardization approach) for identifying clusters of approximate clones. The proposed techniques are validated in a twofold manner. First, a descriptive analysis of approximate clone clusters in two industrial repositories of process models is undertaken. Secondly, a synthetic experiment is conducted to evaluate the accuracy of the clustering techniques with respect to the task of retrieving clusters of clones that have emanated from a single original fragment via copy/pasting followed by independent changes to the duplicated fragments.

The rest of the paper is structured as follows. Section 2 introduces three techniques for process model parsing, exact clone detection and process model comparison used in this paper. Section 3 then presents the proposed approximate clone clustering techniques. Finally Section 5 discusses related work while Section 6 concludes the paper.

2 Preliminaries

This section introduces the three basic ingredients of the proposed technique: RPST, RPSDAG and process model similarity.

2.1 RPST

The RPST [17,13] is a parsing technique that takes as input a process model and computes a tree representing a hierarchy of single-entry single-exit (SESE) fragments. Intuitively, a process model, represented as a directed graph, is partitioned into sets of edges such that the subgraph induced by each set of edges is a SESE fragment. SESE fragments are organized by subset inclusion to form a rooted tree, where siblings are associated to disjoint sets of edges. As the process graph is partitioned into set of edges, some nodes may be shared in several SESE fragments. The RPST can be computed for any process model in linear time and it is unique [17,13].

A node in an RPST corresponds to a fragment of one out of four types: *trivial*, *polygon*, *bond* or *rigid*. A *trivial* consists of a single edge. A *polygon* represents a sequence of fragments. A *bond* corresponds to a subgraph where all child fragments are adjacent to the entry and exit nodes of the fragment. Any other case is a *rigid* fragment. We use the prefixes T, P, B and R to designate the type of fragment. For example fragment B1 is a bond. This bond appears in three different places (its occurrences are thus exact clones). Meanwhile, bonds B2 and B4 could be considered as approximate clones, depending on the user-defined distance threshold. Similarly, one level above, R1, R2 and R3 could also be considered as approximate clones.

Figures 1(a)–(c) present sample process fragments extracted from models in the SAP Reference Model [6]. For sake of clarity, only SESE fragments with at least four vertices are identified in the figures, surrounded by a dashed rectangle. Moreover, Figure 1(d) shows a simplified (tree) representation of the RPST of each fragment in Figures 1(a)–(c). Consider the process fragment shown in Figure 1(a). We can observe that this fragment contains three bonds, viz. B1, B2 and B3; two non-trivial polygons, viz. P1 and P2; and a rigid fragment, viz. R1. Furthermore, the rigid R1 is the root fragment, having B1, P1, and P2 as children. Finally, polygon P1 is parent of bonds B2 and B3.

The process models and fragments in this paper use EPC as the underlying notation. However, the presented techniques are notation-independent.

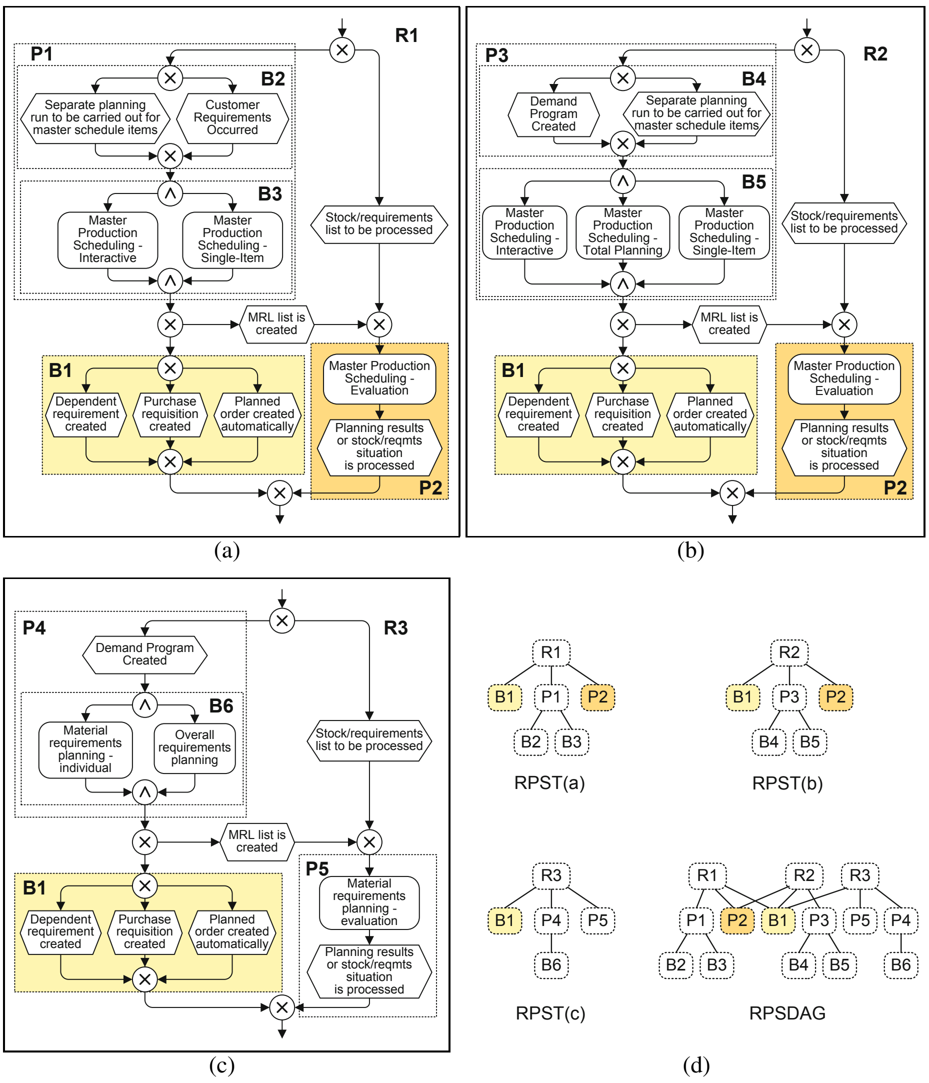

R1
P1
B2
Separate planning run to be carried out for master schedule items
Customer Requirements Occurred
B3
Master Production Scheduling - Interactive
Master Production Scheduling - Single-Item
Stock/requirements list to be processed
MRL list is created
B1
Dependent requirement created
Purchase requisition created
Planned order created automatically
Master Production Scheduling - Evaluation
Planning results or stock/reqmts situation is processed
P2
(a)
R2
P3
B4
Demand Program Created
B5
Master Production Scheduling - Total Planning
(b)
R3
P4
B6
Material requirements planning - individual
Overall requirements planning
P5
Material requirements planning - evaluation
(c)
RPST(a)
RPST(b)
RPST(c)
RPSDAG
(d)

Fig. 1. Process fragments extracted from the SAP Reference Model

2.2 RPSDAG

The RPSDAG [16] is an index structure designed for efficient and accurate identification of exact clones in a collection of process models. Conceptually, it can be thought of as the union of a set of RPSTs. A node in the RPSDAG corresponds to a SESE fragment of a model in the collection, whereas edges encode the containment relation among SESE fragments. Importantly, each fragment only appears once in the RPSDAG. Thus, if a fragment appears multiple times, in the same RPST or in different RPSTs, it is factored out and represented only once in the RPSDAG. For example, Figure 1(d) shows the RPSTs and the RPSDAG of the process fragments presented in Figures 1(a)–(c). Note that fragments B1 and P2 are represented only once in the RPSDAG. A node in the RPSDAG that has more than one parent is an exact clone fragment.

The RPSDAG is built incrementally. When a new process model is added to the collection, the corresponding RPST is computed and merged into the existing RPSDAG. The RPSDAG implementation described in [16] incorporates several optimizations that make it scalable to real-life repositories of process models with hundreds of models. In addition to identifying exact clones, the RPSDAG allows us to determine if a process fragment is contained in another – a feature we will use during clustering.

2.3 Process Model Similarity

The similarity of process models specified in a graph-based notation can be measured on the basis of three complementary aspects: the labels attached to tasks, events and other model elements; their graph structure; or their execution semantics. In this paper, we adopt a measure that combines structural and label similarity and that has been shown to be correlated with perceived similarity [3]. This measure is defined over an abstract representation of process models based on labelled graphs, as follows.

Definition 3 (Process graph). *Let $\mathcal{L}$ be a set of labels. A* (business) process graph *H is a tuple (V, E, λ) where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and $\lambda : V \to \mathcal{L}$ is a function that maps vertices to labels.*

The adopted similarity measure is based on the well-known graph-edit distance [11]. The graph edit distance of two graphs is the minimal set of edit operations required to transform one graph into the other. There are three edit operations: vertex substitution, vertex insertion/deletion and edge insertion/deletion. A vertex substitution refers to the fact that a vertex in one of the graphs is mapped to a vertex in the other graph. To define a valid vertex substitution, we require a notion of vertex similarity. In this respect, we consider that vertices are matched according to their label similarity measured in terms of string-edit distance, denoted as $Sim_{led}(label_1, label_2)$.[3] A vertex substitution is only allowed if the similarity between their labels is above a user-defined threshold (e.g. 0.4). Whenever a vertex in a graph is not matched to any vertex in the other graph, it is considered as either inserted in one graph or deleted in the other one. Similarly, an edge insertion (or deletion) operation is required for each edge that cannot be mapped to an edge in the other graph. This intuition is formalized as follows.

Definition 4 (Normalized process graph edit distance [2]). *Let $H_1 = (V_1, E_1, \lambda_1)$ and $H_2 = (V_2, E_2, \lambda_2)$ be two process graphs. Let $M : V_1 \nrightarrow V_2$ be a partial injective mapping that maps vertices of H_1 to vertices of H_2. Moreover, let $subv$ be the set of substituted vertices, i.e., $\forall v \in subv : v \in dom(M) \cup cod(M)$, $skipv$ the set of skipped vertices, i.e., $\forall v \in skipv : v \notin dom(M) \cup cod(M)$, and $skipe$ the set of skipped edges, i.e., $\forall (v, w) \in skipe : v \notin dom(M) \cup cod(M) \vee w \notin dom(M) \cup cod(M)$. The* normalized graph edit distance *induced by the mapping M is:*

$$Dist^{M}_{GED}(H_1, H_2) = \frac{wskipv \cdot fskipv + wskipe \cdot fskipe + wsubv \cdot fsubn}{wskipv + wskipe + wsubv}$$

[3] Other measures of label similarity (e.g. semantic ones) can be used as discussed in [2].

where $wskipv$*,* $wskipe$ *and* $wsubv$ *are relative weights in the range* $[0..1]$ *assigned to each graph-edit operation,* $fskipv$ *is the fraction of skipped vertices,* $fskipe$ *the fraction of skipped edges, and* $fsubv$ *the average distance between substituted vertices, defined as* $fskipv = \frac{|skipv|}{|V_1|+|V_2|}$*,* $fskipe = \frac{|skipe|}{|E_1|+|E_2|}m$ *and* $fsubv = \frac{2\cdot\sum_{(v,w)\in M} 1-Sim_{led}(\lambda_1(v),\lambda_2(w))}{|E_1|+|E_2|}$*, where* $Dist_{led}$ *is the string-edit distance between vertex labels.*

Finally, the normalized graph-edit distance between H_1 *and* H_2*, written* $Dist_{GED}(H_1, H_2)$*, is the smallest* $Dist^M_{GED}(H_1, H2)$ *across all mappings* M*.*

A $Dist_{GED}$ of 0 means that the process graphs are identical, while a $Dist_{GED}$ of 1 implies that the process graphs are completely dissimilar.

The problem of computing the graph-edit distance is NP-Complete [11]. In this paper, we adopt a fast greedy heuristic described in [2]. Still, despite the fact that we use a greedy heuristic, the computation of the $Dist_{GED}$ is expensive. Accordingly, before computing the actual $Dist_{GED}$ between two graphs, we first calculate a lower-bound of it. When this lower-bound is above threshold τ (cf. Definition 1), we do not need to compute $Dist_{GED}$ to determine if two fragments are approximate clones. In this way, we avoid unnecessary calculations when clustering. The lower-bound is obtained from the following observations. First, we take the largest of the two graphs (i.e. the one with more nodes and more edges). Say that H_1 is larger than H_2 (otherwise we revert the roles). Now, assuming that H_1 is a subgraph of H_2, all vertices of H_1 can be substituted by vertices of H_2, all edges of H_1 are matched with edges of H_2, and no vertices are substituted. The only differences come from the vertices and edges of H_2 that are not in H_1. Thus, $fskipv = \left|\frac{|V_1|-|V_2|}{|V_1|+|V_2|}\right|$, $fskipe = \left|\frac{|E_1|-|E_2|}{|E_1|+|E_2|}\right|$ and $fsubv = 0$. These are lower-bound values. If the assumption that H_1 is not a subgraph of H_2 is violated, then the graph-edit distance will necessarily be greater because it entails additional differences. Thus, we conclude that $Dist_{GED}(H_1, H_2)$ is greater than the one obtained by feeding the above lower-bound values of $fskipv$, $fskipe$ and $fsubv$ into the equation for $Dist^M_{GED}(H_1, H2)$ in Definition 4. Note that if the graphs have equal size, the obtained lower-bound is zero – which is not useful.

3 Approximate Clones Clustering

In order to operationalize the two approaches proposed in the introduction, we reviewed various clustering algorithms and selected two of them which allowed us, with minor adaptations, to fulfill our requirements. These are the *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* [15] for the first approach, and the *Hierarchical Agglomerate Clustering (HAC)* [15] for the second approach. In both algorithms, described below, we assume that the distance between every possible pair of fragments has been pre-computed and stored in a *distance matrix*. This matrix only stores the distance $Dist_{GED}$ of Definition 4 for a pair of fragments if this is within the user-defined threshold τ, and if the two fragments do not contain one another (*non-containment relationship*). For all other fragment pairs, it stores ∞.

3.1 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

In the first approach we propose to standardize a set of clones towards a medoid fragment. Given a cluster, a medoid is an element of the cluster that is the closest to the center of the cluster. The medoid does not necessarily coincide with the center of the cluster (called *centroid*) since in our problem the distance between the medoid and all other cluster elements is not the same, but is bounded by the user-defined threshold. A well-known algorithm that is built upon this principle is DBSCAN. DBSCAN creates clusters based on the density of *neighborhoods*. Given a set of objects O, the neighborhood of an object $o \in O$ is the set of fragments $N_o = \{o_i \in O \mid d(o, o_i) \leq \epsilon\}$, where $d(o, o_i)$ is a distance measure between o and o_i and ϵ is the neighborhood radius. A *core object* is an object whose $|N_o| \geq Size_{min}$, where $Size_{min}$ is the minimum cluster size (we observe that a core object is contained in its neighborhood since its distance with itself is 0). Thus, we have to specify two parameters for this algorithm: neighborhood radius and minimum cluster size. In our case, the neighborhood radius coincides with the used-defined distance threshold τ, whereas we can fix $Size_{min}$ to 2 to retrieve clusters of at least two fragments. Moreover, we use the notion of graph-edit distance $Dist_{GED}$ as the distance measure between two objects.

Standard DBSCAN identifies all core objects of a given dataset and considers their neighborhoods as initial clusters. If two core objects are within each other's neighborhood, their neighborhoods are merged into a single cluster. On the other hand, if an object does not belong to the neighborhood of any core object, it is marked as *noise*. Our adaptation of DBSCAN is described in Algorithm 1. Given the set of process fragments G extracted from the RPSDAG, the algorithm repeats the clustering process (Steps 2–14) until all fragments in G have been checked whether they are core objects. At the beginning of each iteration, a random fragment f is removed from G and marked as "processed". The neighborhood N_f of f is computed (Step 3), and if f is a core object the fragments in N_f are removed from G and from $Noise$ (Step 5), and added to a new cluster C (Step 6). Otherwise f is treated as noise and another fragment is extracted from G. The algorithm then *expands* cluster C by checking whether there are core objects in C whose neighborhoods can be merged with C. This is done by iterating over all fragments in N_f except f, via a set M_C. For a fragment m in M_C that has not been processed, its neighborhood N_m is computed (Step 8) to determine whether m is itself a core object. If so, before merging its neighborhood with C, we check whether there is still a medoid s whose distance with all other fragments of the combined cluster is within τ (Step 10), otherwise we will create clusters whose fragments are far apart from each other to be standardized. In case of merging, the fragments in N_m are removed from G and added, except m, to M_C (Step 11), so that they can be checked whether they are core objects. If N_m cannot be merged with C, m is added back to G so that it can be eventually processed again (Step 12). In fact, N_m may form a cluster by itself or be merged with some other cluster.

A fragment's neighborhood is constructed using the distance matrix. Given the non-containment relation enforced by this matrix, a fragment cannot be in the neighborhood of a core object that contains or is contained by it. Still, it is possible to include two related fragments in a neighborhood if they are both sufficiently similar to the core object. To prevent this, we retrieve the set of all the ascendants and descendants of a fragment

Algorithm 1. DBSCAN Clustering

```
Input: Set $G$ of process fragments.
Output: The sets of clusters ($Clusters$) and noise ($Noise$).

1  Initialize $Clusters$ and $Noise$ to empty sets.
2  Remove a fragment $f$ from $G$ and mark $f$ as "processed".
3  Retrieve the neighborhood $N_f$.
4  If $|N_f| < Size_{min}$, add $f$ to $Noise$, then go to 1.
5  Remove $N_f$ from $G$ and from $Noise$.
6  Initialize a new cluster $C$ in $Clusters$ with $N_f$, and a new set $M_C$ to $N_f \setminus \{f\}$.
7  Remove a fragment $m$ from $M_C$.
8  If $m$ is not "processed", mark $m$ as "processed" and retrieve $N_m$.
9    If $N_m \geq Size_{min}$
10     If there is a fragment $s \in C \cup N_m$ such that for all $p \in C \cup N_m$ $Dist_{GED}(s, p) \leq \tau$
11       Remove $N_m$ from $G$ and $Noise$ and add $N_m$ to $C$ and $N_m \setminus \{m\}$ to $M_C$.
12     Else, mark $m$ as "unprocessed" and add it to $G$.
13 If $M_C \neq \varnothing$ go to 1.
14 If $G \neq \varnothing$ go to 1.
```

by computing its transitive closure on the RPSDAG, and add to the neighborhood the fragment in the transitive closure that is the nearest to the core object (the original fragment may thus be discarded in favor of one of its ascendants or descendants). Further, we mark all other fragments in the transitive closure as "visited" for that cluster, so that these fragments will not be included in any neighborhood of that cluster.

The complexity of Algorithm 1 is dominated by that of the neighborhood computation (Steps 3 and 8), and by that of the merging condition (Step 10). Neighborhood computation for a fragment f requires at most $|G| - 1$ lookups in the distance matrix. The exploration of the transitive closure of each neighbor of f requires further $|G| - 1$ lookups (retrieving the transitive closure of an RPSDAG node is linear on the RPSDAG size, which is bounded by $|G|$). Similarly, the merging condition requires $|G| - 1$ lookups in the distance matrix for all members of a cluster. As the main loop is repeated $|G|$ times, the overall complexity of Algorithm 1 is $O(|G|^3)$. This is higher than the complexity of standard DBSCAN, which is $O(|G|^2)$ [15]. That said, in our experience the algorithm showed to be efficient (cf. Section 4). In fact, the search space is greatly reduced by the cutoff conditions used when computing the distance of clusters, i.e. the distance threshold τ and the non-containment relationship. The result is that the distance matrix is highly sparse, but the sparsity depends on intrinsic characteristics of the process model collection. Further, we store each computed neighborhood so that it can be reused when reprocessing a core object whose neighborhood has not been merged.

3.2 Hierarchical Agglomerate Clustering (HAC)

In the second approach, a set of clones can be standardized by selecting any fragment in the group as a reference and standardizing all other fragments towards this reference fragment. In other words, we require that every pair of fragments in a cluster has a distance below the threshold τ. This goal can be straightforwardly mapped to the strategy followed by the basic hierarchical agglomerative clustering method [15]. This clustering method starts with singleton clusters and iteratively combine the pair of clusters that is found to be the closest among all other possible pairs. The process of merging continues until there is only one cluster left.

One key issue is the definition of the distance between two clusters, which needs to be recomputed after every cluster merging. Several possibilities are available: taking the smallest distance between fragments in one of the clusters to the fragments in the other one, known as *single link*; taking the farthest distance, referred to as *complete link*; among others. It can be easily see that the complete link strategy suites well to our purposes, as it allows to identify the cluster mergings that will not meet the requirement of keeping a distance below the threshold τ. Note that the identification of such situation can be accomplished ahead of time. The intuition is captured in the following definition.

Definition 5 (Distance of clusters under complete link strategy). *Let C_i and C_j be clusters in the dendrogram built by a hierarchical clustering algorithm, and τ be the similarity threshold among fragments of C_i and fragments of C_j. Moreover, let $\mathcal{F}(C)$ be a function that returns the set of fragments associated to C, inductively defined as follows: (BASE) if C is a leaf node in the dendrogram, C is a singleton and refers to a single fragment, say f, then $\mathcal{F}(C) = \{f\}$; (STEP) if C is an intermediate node then $\mathcal{F}(C) = \cup_{c \in C} \mathcal{F}(c)$. The distance of clusters C_i and C_j, denoted as $Dist(C_i, C_j)$, can defined as follows.*

$$\begin{cases} \infty & \text{if } \exists f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j) : g \subseteq f \vee f \subseteq g \\ \infty & \text{if } max_{f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j)}\, Dist_{GED}(f,g) > \tau \\ max_{f \in \mathcal{F}(C_i), g \in \mathcal{F}(C_j)}\, Dist_{GED}(f,g) & \text{otherwise} \end{cases}$$

We note that the distance of two clusters is set to ∞ when there exist one fragment in the first cluster which is in containment relationship with another fragment in the second cluster. Moreover, when farthest distance between fragments of both clusters is above the threshold τ, the distance is set to ∞. In the two previous cases, we are meeting the constraints described in Definitions 1 and 2. Finally, the farthest distance between fragments of both clusters is reported as the distance of the clusters, only when the value is less or equal to the threshold τ. Algorithm 2 corresponds to the modified version of the basic hierarchical agglomerative method adapted for clustering approximate clones.

Algorithm 2 can be divided into two parts. Step 1 and 2, initialize the set of singleton clusters, stores them in $TopClusters$ and initializes the distance matrix between clusters (according to Definition 5). The remaining steps correspond to the main loop. In Step 3, a pair of clusters is selected such that their distance is found to be the smallest among all other possible pairs. If the distance of such pair is ∞ or there is only one cluster left then the algorithm stops. In Step 4, a new cluster is created to hold the union the previously selected pair. In Step 5, the distance matrix is updated (according to Definition 5), by removing the pair clusters previously selected and adding the newly created cluster.

The algorithm starts with a working set of $|G|$ clusters. In every iteration, two clusters are removed and a new one is added. Hence, the size of the working set decreases monotonically. The algorithm stops when $|TopClusters| = 1$ or before if the entire distance matrix $\mathcal{D}$ is filled with ∞.

Algorithm 2. Hierarchical Agglomerative Clustering

```
Input: Set $G$ of process fragments.
Output: The set of maximal clusters, viz. $TopClusters$.
1 For each $f \in G$ create a singleton cluster. Initialize $TopClusters$ to contain all singleton
  clusters.
2 Using the distance matrix between fragments, calculate the initial distance matrix between
  clusters in $TopClusters$, i.e. $\mathcal{D}[i,j] \leftarrow Dist(C_i, Cj)$, where $C_i, C_j \in TopClusters$ .
3 In the distance matrix $\mathcal{D}$, select a pair of clusters $C_i, C_j \in TopClusters$ such that their
  distance is the minimum. Stop if no such pair exists, i.e. either all distances in $\mathcal{D}$ are $\infty$ or
  $|TopClusters| = 1$.
4 Combine clusters $C_i$ and $C_j$ to form a new cluster $C_{ij}$. Remove clusters $C_i$ and $C_j$ from
  $TopClusters$. Add cluster $C_{ij}$ to $TopClusters$.
5 Update matrix $\mathcal{D}$ by adding the distance between cluster $C_{ij}$ and all other clusters in
  $TopClusters$.
6 Go to 2.
```

The complexity of Algorithm 2 is dominated by the maintenance of the distance matrix (i.e., Steps 2 and 2), which has an initial size of $O(|G|^2)$. As the main loop is repeated $O(|G| - 1)$ times, the worst-case upper bound of the complexity is of $O(|G|^3)$ [15]. The same simplifications of the search space that we used for DBSCAN apply to HAC (distance cutoff and non-containment). Also this algorithm has shown to be efficient in our experience.

4 Evaluation

We implemented the two algorithms in the Apromore [8] platform,[4] and evaluated them in a twofold manner. First, we performed a descriptive analysis of approximate clone clusters in two industrial process model collections in order to assess the potential usefulness of the techniques. Secondly, we conducted an experiment to measure the accuracy of the technique at retrieving fragments resulting from copy/pasting and subsequent independent changes. Both experiments make use of a measure of cluster quality intended to capture the potential benefits of standardization.

4.1 Cluster Quality Measure

The proposed techniques are aimed at retrieving clusters of fragments that can be standardized into a common fragment. Such a standardization activity entails a certain effort and brings in certain benefits – in the form of less duplication and thus smaller total repository size. We contend that clusters that have a higher benefit-to-cost ratio are most likely to be candidates for standardization. In particular, if a cluster of approximate clones has emerged from copy/pasting of a fragment followed by independent changes of the copied fragments, it is likely to have a high benefit-to-cost ratio, provided that the changes made are not considerable.

[4] Available at www.apromore.org

To operationalize the benefit-to-cost ratio as a measure of cluster quality, we need to define a cost measure and a benefit measure. The cost of standardizing the fragments of a cluster into a single fragment is determined by many factors, some of them exogenous to the process models themselves. However, we contend that this cost is proportional to the amount of elementary changes that will be made to the fragments in order to standardize them to one common subprocess. Indeed, each elementary change will require a certain amount of effort to ensure that the execution of the process is adapted to this change. Accordingly, we hereby use the absolute GED ($Dist_{AGED}(H_1, H_2)$) defined in the same way as $Dist_{GED}(H_1, H_2)$ in Definition 4 but replacing $fskipv$ and $fskipe$ with $|skipv|$, $|skipe|$ respectively, and removing the denominator in the definition of $fsubv$. In other words, we count actual number of edit operations as opposed to fraction of edit operations relative to total size. We do not used the normalized GED in this context ($Dist_{GED}$), because this normalized version is not reflective of the number of operations required to standardize the fragments. Instead, $Dist_{GED}$ is reflective of the percentage difference shared between two models.

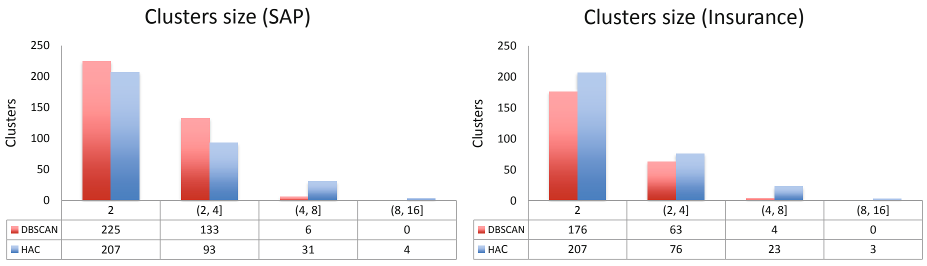
In the case of clusters produced using DBSCAN, there is a designated medoid that serves as a reference. Thus, the cost of standardizing the cluster is the sum of the distances between each fragment in the cluster and the medoid (m), i.e. $\sum_{f \in C} Dist_{AGED}(f, m)$. In the case of clusters produced using hierarchical clustering, every fragment in the cluster could potentially be used as the "medoid" towards which all fragments would be standardized. Assuming that the aim is to maximize the benefit-to-cost ratio, we will pick as medoid the fragment that will yield the highest benefit-to-cost ratio (see below).

The benefit of standardizing and refactoring a cluster into a subprocess is proportional to the amount of reduction in duplication, which in turn reflects itself in a reduction in size of the overall repository. This size reduction is equal to the sum of the sizes of the fragments in the cluster (since they are removed) to which we subtract the size of the medoid – since this medoid becomes a new subprocess – and the number of fragments – since each cluster is replaced by a "call activity" to the subprocess. In other words, the benefit of standardizing a cluster is $\sum_{f \in C} |f| - |m| - |C|$.

Given the above, we define the benefit-to-cost ratio of a cluster obtained with the DBSCAN method as $BCR(C) = \frac{\sum_{f \in C} Dist_{AGED}(f,m)}{\sum_{f \in C} |f| - |m| - |C|}$. In the case of hierarchical clustering, we define the benefit-to-cost ratio of a cluster as the maximum of BCR(C) across all fragments in the cluster.

4.2 Potential Usefulness Assessment

We assessed the potential usefulness of the approximate clone clustering techniques using two datasets. The first dataset is the SAP R/3 reference model [6]. It contains 595 models with sizes ranging from 5 to 119 nodes (average 22.28). The second dataset is taken from an insurance company under condition of anonymity. It contains 363 models ranging from 4 to 461 nodes (average 27.12). We first computed the RPSDAG for both datasets and post-processed them by factoring out all exact clones using the technique presented in [16]. This yielded 2,238 non-trivial fragments with at least 4 nodes for the SAP dataset (11.47 average size) and 2,037 for the insurance dataset (16.58 aver-



Clusters size (SAP)

	2	(2, 4]	(4, 8]	(8, 16]
🟥 DBSCAN	225	133	6	0
🟦 HAC	207	93	31	4

Clusters size (Insurance)

	2	(2, 4]	(4, 8]	(8, 16]
🟥 DBSCAN	176	63	4	0
🟦 HAC	207	76	23	3

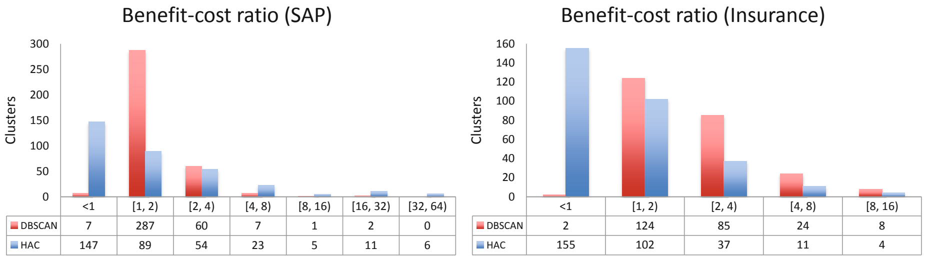
Fig. 2. Number of clusters vs clusters size for both algorithms

age size). We then applied the two clustering methods independently – having eliminated exact clones to avoid double-counting. The clustering algorithms were run with a $Dist_{GED}$ threshold of 0.4.

All tests were run on a PC with a dual core Intel processor, 1.8GHz, 4GB memory, running Microsoft Windows 7 and Oracle Java Virtual Machine v1.6. The cluster computation is dominated by the computation of the distance matrix which took 26.3 mins for the SAP dataset and 2.69 hours for the insurance dataset. The time for clustering itself is negligible in comparison. The longer time taken for the insurance dataset is justified by the size of its fragments – much larger than those in the SAP dataset (e.g. the largest fragment in the insurance dataset is a rigid with 461 nodes whereas the largest SAP fragment contains 117 nodes).

Figure 2 plots the histograms of distribution of cluster sizes for the two datasets. For the SAP dataset we retrieved a total of 364 clusters with DBSCAN (with sizes ranging from 2 to 5 clusters) and 335 clusters for HAC (sizes between 2 and 13), while for the insurance dataset we retrieved 243 clusters with DBSCAN (sizes between 2 and 6) and 309 clusters with HAC (sizes between 2 and 10). This confirms the intuition that real-life process model repositories contain a large number of approximate clone clusters, and thus that copy/pasting of fragments across process models is a very common practice. Looking at the size distribution, for both datasets the majority of the clusters retrieved by the two algorithms contain between 2 and 8 fragments, with the largest clusters having 2 fragments. This suggests that copy/pasting is typically limited to 6-8 copies per fragment.

Figure 3 shows the histograms of distributions of BCR for both datasets. We observe that in general none of the techniques performs better than the other, since for the SAP



Benefit-cost ratio (SAP)

	<1	[1, 2)	[2, 4)	[4, 8)	[8, 16)	[16, 32)	[32, 64)
🟥 DBSCAN	7	287	60	7	1	2	0
🟦 HAC	147	89	54	23	5	11	6

Benefit-cost ratio (Insurance)

	<1	[1, 2)	[2, 4)	[4, 8)	[8, 16)
🟥 DBSCAN	2	124	85	24	8
🟦 HAC	155	102	37	11	4

Fig. 3. Number of clusters vs benefit/cost ratio for both algorithms

dataset we achieve higher BCRs for HAC than for DBSCAN, whilst for the insurance dataset it is the other way around. This suggests that depending on the type of the repository, one of the two techniques might be more appropriate than the other.

4.3 Retrieving Copy/Pasted Fragments

The second experiment aimed to evaluate the accuracy of the clustering techniques with respect to the task of retrieving clusters of clones that have emanated from a single original fragment by means of copy/pasting followed by independent changes to the duplicated fragments. We did so by simulating a situation where new fragments are inserted in an existing process model repository by copying a master fragment across various models of the repository, after doing minor changes. We randomly extracted 50 fragments from the two datasets used in the previous experiment, such that they were sufficiently different from each other (pairwise graph-edit distance above 70%).

To test the accuracy of the DBSCAN algorithm, we used these 50 fragments as "seeds" to generate 50 artificial clusters by producing from 2 to 10 variants for each seed, and grouping each seed with its variants in a cluster. We obtained a total of 311 fragments in 50 clusters. Seed variants were obtained by applying simple change operations (edge/node removal or insertion), such that the graph-edit distance between a variant and its seed was no more than 40% – the same threshold that we used in the first experiment. The clusters' size ranged from 3 to 10 fragments (average 6.35). We then generated 300 process models from the two existing datasets, such that none of these models contained any of the seed fragments, and we randomly inserted the 311 fragments into these models such that a model would contain from 0 to 2 fragments. We then extracted the RPSDAG from this dataset and clustered the retrieved fragments using our DBSCAN. The algorithm retrieved 328 clusters. We matched each artificial cluster with the retrieved fragment that yielded the maximum *FScore* [19]. FScore is the harmonic mean of the recall and precision of a retrieved cluster with respect to (w.r.t.) an artificial cluster. Precisely, given an artificial cluster l and a retrieved cluster s, the FScore of s w.r.t. l is $F(s,l) = \frac{2 \cdot R(s,l) \cdot P(s,l)}{R(s,l)+P(s,l)}$ where $R(s,l)$ and $P(s,l)$ are the recall and precision of s w.r.t. l.

In order to measure the overall quality of the algorithm, we then computed the *weighted average FScore* (F_{wa}) [19]. F_{wa} is the maximum FScore of each artificial cluster weighted against the combined size of all artificial clusters. Let L be the set of artificial clusters and S the set of retrieved clusters. Then $F_{wa} = \sum_{l=1}^{L} \frac{|l|}{|L|} F(l)$, where $F(l) = \max_{s \in S} F(s,l)$.

We repeated the same experiment for the HAC algorithm. In order to ensure that all fragments in an artificial cluster have pairwise graph-edit distance within the 40% threshold, we used a *random walk* approach. From each seed we generated a variant with graph-edit distance of at most 0.4. We chose one of these two fragments and generated another variant such that its distance to both fragments was at most 0.4, and so on until we generated from 2 to 10 variants for each cluster. This led

Table 1. Various quality metrics for the two algorithms

	Recall				Precision				F_{wa}
	min	max	avg	std	min	max	avg	std	
DBSCAN	0.17	1	0.71	0.37	0.2	1	0.89	0.24	0.73
HAC	0.1	1	0.82	0.25	0.17	1	0.84	0.33	0.77

to a total of 289 fragments in 50 clusters, with sizes ranging from 3 to 10 fragments (average 5.8). We inserted these fragments in the collection of 300 process models that we generated in the previous step, and then clustered the fragments retrieved from the RPSDAG of this collection using HAC. This led to 295 clusters.

The results for both algorithms are reported in Table 1. Besides F_{wa}, this table reports the minimum, maximum, average and std. deviation of recall and precision for the best-matched retrieved cluster for each artificial cluster. The accuracy of the two algorithms is partly affected by the presence of approximate clones that exist in the generated process model collections, besides those that have been generated artificially. Despite this, the results show high F_{wa} (0.73 for DBSCAN and 0.77 for HAC), as well as high average precision and recall for both algorithms, demonstrating the accuracy of the algorithms. None of the algorithms clearly outperforms the other.

Finally, we used the above data to evaluate the ranking accuracy of the BCR. For each algorithm, we plotted a ROC curve by ordering the retrieved clusters from the highest to the lowest BCR. In these curves, we considered a retrieved cluster as a true positive if it had a recall of 1, and as a true negative otherwise. The curves, shown in Fig. 4, show that the clusters with highest BCR are indeed those that most closely match the synthetically generated clusters. This result is confirmed by the Area Under the Curve which is 0.89 for DBSCAN and 0.72 for HAC (both with asymptotic significance less than 0.05).

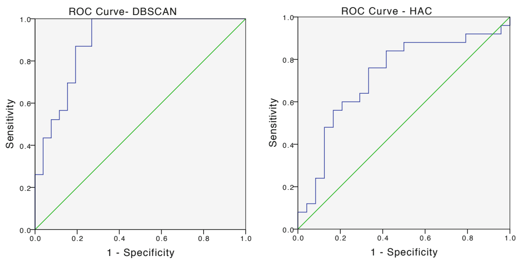


Fig. 4. ROC curves for both algorithms

5 Related Work

Clone detection in software repositories has been an active field of research for several years [7]. However in this field focus has been on exact software clone detection.

In the field of model-driven engineering, approximate clone detection has been investigated in [1], [12] and [14]. In [1] the authors present *CloneDetective*, a method for detecting clones in large repositories of Simulink/TargetLink models from the automotive industry. Models are partitioned into connected components which are compared pairwise using a heuristic subgraph matching algorithm. These pairs are then clustered based on the sets of their node labels. According to [12], CloneDetective suffers from low inaccuracy and low degree of completeness in detection, mainly due to the fact that small clones are absorbed by larger clone pairs. In other words, the algorithm tends to find as large clones as possible, whereas in our approach we allow related fragments to belong to different clusters, so that users can choose the abstraction level at which to standardize. Moreover, this method is not very sensitive to approximate clones having small differences. These cases commonly result from copy/pasting and as such they should not be discarded. Moreover, they yield low standardization costs making them easy to standardize. The work in [12] overcomes these problems by proposing two methods for exact and approximate matching of clones. In particular, the second method, namely *aScan*, represents graphs by a set of vectors built from graph features: e.g. path lengths and vertex in/out degrees. An empirical study shows that this feature-based approximate matching improves pre-processing and running times, while keeping a high precision. Despite these advantages, the method proposed in [12] does not fulfill our requirements: The resulting clones may be non-SESE fragments and the identified clusters do not satisfy any of the properties in Definition 2. The work in [14] detects clones in UML models, such as class or activity diagrams. In this work, each object, its properties and child objects (all called *model elements*) form a fragment. The similarity between two fragments is computed by summing up the pairwise similarities of their respective elements. This method is not suitable for our purposes as it does not consider structural similarity, fragments are fixed to specific structures, and no clustering technique is proposed. Moreover, the method is not very sensitive to approximate clones having substantial differences (e.g. removals or additions of parts).

Refactoring process model collections has been investigated in [16,4,18]. In [16], we described a technique to find fragments that are equal across different process models, so that they can be factored out in separate subprocess. In this paper, we assume that all such exact clones have already been factored out, but we reuse the RPSDAG structure that we built in [16] to identify hierarchical dependencies among fragments in different process models. In [4], process fragments that are sufficiently similar to each other are identified. In contrast to our work, fragment similarity is exclusively based on label similarity rather than a combination of label and structural similarity. Also, fragments are considered pairwise and no clustering takes place. This approach can help analysts detect overlap between process models, however no support is offered to standardize these similar fragments such that they can be refactored. In [18], eleven process model refactoring techniques are identified and evaluated. Extracting process fragments as subprocesses is one of the techniques identified. Our work addresses the problem of identifying opportunities for such "fragment extraction" and provides an actual implementation and experimentation. In addition, [18] does not consider clustering.

Clustering of process models has been dealt with in [5] and [10]. In both cases process models are clustered rather than process fragments leading to a small number of clusters. Using fragments instead of process models is more complex, but for the purposes of standardization and reuse it is more suitable as a fragment may be shared between process models, while the rest of these models may be quite different.

In [9] an approach is described to synthesize the most representative process model out of a collection of variants. This work is complementary to ours in that it could be used after clustering has been applied in order to synthesize the centroid of a cluster. However, this is not the approach we followed as this may likely lead to an artificially created centroid which does not represent an actual fragment occurring in a process model. The presence of such an artificial fragment could cause problems for a business analyst when trying to standardize a cluster.

6 Conclusion

This paper presented two techniques for retrieving clusters of approximate clones for possible refactoring into shared subprocesses. Experiments showed that both techniques, coupled with the proposed measure of cluster quality (benefit-to-cost ratio), are able to accurately retrieve clusters resulting from copy/pasting activity followed by independent modifications to the copied fragments. A descriptive analysis of clones in two industrial process model repositories put into evidence a proliferation of approximate clones of varying sizes and benefit-to-cost ratio.

The evaluation is limited in at least two respects. First, clustering and cluster ranking accuracy are evaluated based on synthetic data – albeit generated via perturbations of real-world fragments. The retrieved clusters may not be reflective of the types of clusters that analysts would find most suitable for standardization and refactoring. Addressing this limitation requires a realistic "golden standard", for example, one resulting from a manual assessment of cluster quality by domain experts. This is a direction for future work. A second limitation is that only two repositories were used to evaluate the potential benefit of the proposed techniques. In one case one technique led to higher overall benefit-to-cost ratio, while the reverse was observed in the second case. Further evaluation is needed to determine in what cases one technique should be preferred over the other. Finally, the evaluation could be extended to include other clustering techniques.

Acknowledgments. This work is funded by the Smart Services Cooperative Research Centre (CRC) under the Australian Government's CRC Program, the Estonian Science Foundation and ERDF via the Estonian Centre of Excellence in Computer Science.

References

1. Deissenboeck, F., Hummel, B., Jürgens, E., Schätz, B., Wagner, S., Girard, J.-F., Teuchert, S.: Clone Detection in Automotive Model-based Development. In: ICSE (2008)
2. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
3. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Inf. Syst. 36(2), 498–516 (2011)
4. Dijkman, R.M., Gfeller, B., Küster, J.M., Völzer, H.: Identifying refactoring opportunities in process model repositories. Information & Software Technology 53(9), 937–948 (2011)
5. Jung, J.-Y., Bae, J.: Workflow Clustering Method Based on Process Similarity. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3981, pp. 379–389. Springer, Heidelberg (2006)
6. Keller, G., Teufel, T.: SAP R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley (1998)
7. Koschke, R.: Identifying and Removing Software Clones. In: Software Evolution. Springer (2008)
8. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: An Advanced Process Model Repository. Expert Systems With Applications 38(6) (2011)
9. Li, C., Reichert, M., Wombacher, A.: The minadept clustering approach for discovering reference process models out of process variants. IJCIS 19(3-4), 159–203 (2010)

10. Melcher, J., Seese, D.: Visualization and clustering of business process collections based on process metric values. In: SYNASC. IEEE (2008)
11. Messmer, B.T.: Efficient Graph Matching Algorithms. PhD thesis, Switzerland (1995)
12. Pham, N.H., Nguyen, H.A., Nguyen, T.T., Al-Kofahi, J.M., Nguyen, T.N.: Complete and Accurate Clone Detection in Graph-based Models. In: ICSE, pp. 276–286. IEEE (2009)
13. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified Computation and Generalization of the Refined Process Structure Tree. In: WSFM (2010)
14. Storrle, H.: Towards clone detection in UML domain models. Software and Systems Modeling (2011) (on-line)
15. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley (2005)
16. Uba, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Clone Detection in Repositories of Business Process Models. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 248–264. Springer, Heidelberg (2011)
17. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data Knowl. Eng. 68(9), 793–818 (2009)
18. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. Computers in Industry 62(5), 467–486 (2011)
19. Zhao, Y., Karypis, G.: Evaluation of hierarchical clustering algorithms for document datasets. In: CIKM, pp. 515–524. ACM (2002)

Probabilistic Optimization of Semantic Process Model Matching

Henrik Leopold[1], Mathias Niepert[2], Matthias Weidlich[3], Jan Mendling[4], Remco Dijkman[5], and Heiner Stuckenschmidt[2]

[1] Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
henrik.leopold@wiwi.hu-berlin.de
[2] Universität Mannheim, 68159 Mannheim, Germany
{mathias,heiner}@informatik.uni-mannheim.de
[3] Technion - Israel Institute of Technology, Technion City, 32000 Haifa, Israel
weidlich@tx.technion.ac.il
[4] Wirtschaftsuniversität Wien, Augasse 2-6, 1090 Vienna, Austria
jan.mendling@wu.ac.at
[5] Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands
r.m.dijkman@tue.nl

Abstract. Business process models are increasingly used by companies, often yielding repositories of several thousand models. These models are of great value for business analysis such as service identification or process standardization. A problem is though that many of these analyses require the pairwise comparison of process models, which is hardly feasible to do manually given an extensive number of models. While the computation of similarity between a pair of process models has been intensively studied in recent years, there is a notable gap on automatically matching activities of two process models. In this paper, we develop an approach based on semantic techniques and probabilistic optimization. We evaluate our approach using a sample of admission processes from different universities.

1 Introduction

Business process models are increasingly used by companies for documentation purposes. A process documentation initiative stores an extensive amount of process models in a centralized process repository. This amount can easily rise to several thousand models in large enterprises. Due to the size of such companies, process modeling is often conducted by decentralized teams. A consistent and systematic documentation of processes is often achieved by defining guidelines. However, typically none of the team members has detailed insight into the entire set of process models stored in the repository.

The availability of a detailed documentation of a company's business processes bears a lot of potential for business analysis, such as process standardization, compatibility analysis, or business service identification. *Process model matching*, realized by tools called *matchers*, is a prerequisite for such analyses.

It defines which activities in one process model correspond to which activities in another model. Such matches are required, for example, to determine which activities can be merged when deriving standard processes from a collection of processes. It is also needed to judge behavior compatibility or equivalence, and to query a collection of business process models for a certain process or process fragment. The importance of such questions is reflected by recent contributions on computing similarity of pairs of process models, e.g. [1,2,3,4,5,6].

In this paper, we address process model matching with semantic matching techniques and probabilistic optimization. The approach comprises two steps. First, match hypotheses are generated based on automatically annotated activity labels. We rely on a semantic interpretation of activity labels, whereas existing work [7,8] (despite a notable exception [9]) is limited to syntactical similarity assessment. Second, match constraints are derived based on behavioral relations of process models. Those constraints are used for guiding the matching with a probabilistic model, whereas existing work directly leverages the model structure or execution semantics [7,8]. The evaluation of our approach with admission processes from nine different universities shows that the novel conceptual basis for process model matching indeed improves performance. In particular, we are able to show that match results are more stable over different levels of process model heterogeneity. Besides the definition of the matcher, our contribution is a comparative analysis of the strengths and weaknesses of classical matchers and semantic matching with probabilistic optimization. As such, we provide valuable insights for advancing the field of process model matching.

Against this background, the paper is structured as follows. Section 2 illustrates the problem of matching process models. Section 3 presents a matcher that incorporates the generation of semantic match hypotheses based on automatically annotated activities and a probabilistic approach towards match optimization using behavioral constraints. Section 4 challenges our approach using a process model collection from practice. Section 5 reflects our contribution in the light of related work. Finally, Section 6 summarizes the findings.

2 Problem Illustration

This section illustrates the problem of matching process models. We present basic terminology and discuss the state of the art in finding matches.

Given two process models with sets of activities $\mathcal{A}_1$ and $\mathcal{A}_2$, matches between their activities are captured by a relation $match : \mathcal{P}(\mathcal{A}_1) \times \mathcal{P}(\mathcal{A}_2)$. An element $(A_1, A_2) \in match$ defines that the set of activities A_1 matches the set of activities A_2, i.e., they represent the same behavior in the organization. If $|A_1| = 1$ and $|A_2| = 1$, we call the match an *elementary match* or 1:1 match. Otherwise, we speak of a *complex match* or 1:n match. For convenience, we introduce a relation $map : \mathcal{A}_1 \times \mathcal{A}_2$, which defines the relations between individual activities as induced by $match$, $map = \{(a_1, a_2) | (A_1, A_2) \in match, a_1 \in A_1, a_2 \in A_2\}$.

Figure 1 shows admission processes from two different universities. We highlighted matches by gray boxes around the activities, e.g., activity *Check formal*

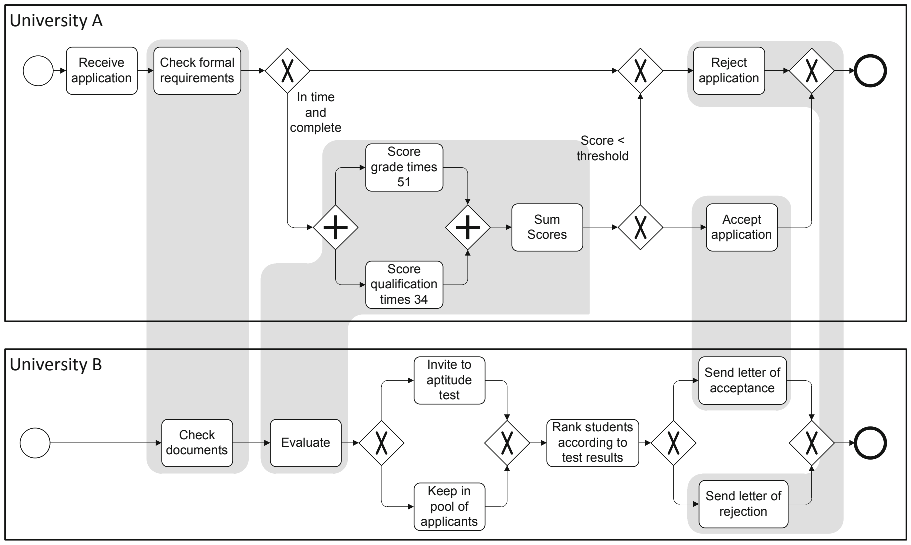


Fig. 1. Example of a business process models with matches

requirements of University A corresponds to activity *Check documents* of University B. Although the processes have the same goal, the organizational behavior is modeled differently. Different labels are used (e.g., *Accept application* versus *Send letter of acceptance*) and there are differences in the level of detail (e.g., *Evaluate* of University B is described in more detail for University A). Also the behavior represented by the processes differs. For example, at University B the *Evaluate* activity is mandatory, whereas at University A the matching activities can be skipped. Before these behavioral differences can be analyzed, however, matches between the activities have to be determined. The goal of *matchers*, such as the ones described in [8,7], is to detect such matches automatically.

A matching approach of particular interest is the ICoP framework [8]. It defines a generic architecture for assembling matchers along with reusable matching components. As such, it integrates several of the proposed matchers, e.g., the graph-based matcher presented in [7]. Following the ICoP architecture, the procedure for automatically detecting matches involves four kinds of matching components: *searchers* find potential matches between activities, *boosters* improve the quality of potential matches by combining them, *selectors* construct the actual mapping from potential matches, and *evaluators* evaluate the quality of an actual mapping with the purpose of finding the best mapping.

Matching components implemented for the ICoP framework leverage syntactic measures, such as string edit distance or vector-space scoring, to find match candidates. Selection and evaluation is guided by the structure of process models, e.g., utilizing the graph edit distance. An evaluation of the existing ICoP components showed that much improvement is still possible with respect to automatically detecting matches. Given the focus on syntactic measures of the existing components, approaches that relate activities based on the semantics of their labels can particularly be expected to improve matching performance.

3 Matching Based on Semantics and Constraints

This section introduces our approach for matching process models. It consists of four phases. First, we annotate the activities of the considered models with their semantic components such as action and business object. Afterwards, we use these annotations for generating match hypotheses for activity pairs. Then, we compute behavioural constraints in order to properly incorporate control flow aspects into the matching process. Finally, we involve these aspects in determining the most likely match constellation using a Markov logic network.

3.1 Activity Label Annotation

Semantic matching requires the precise recognition of semantic components of an activity label. Every activity label can be decomposed into three components [10]: an action, a business object on which the action is performed, and an optional fragment providing further details. For example, the activity label *Forward Request to Insurance Department* contains the action *forward*, the business object *request* and the additional fragment *to Insurance Department*. The challenge here is to identify these different components for activities of different label styles. *Verb-object* style labels start with an imperative verb followed by business object and additional fragment, e.g. *Calculate Costs for Production*. In action-noun labels the action is formulated as a noun, e.g. *Order Shipment to Customer*.

The last example points to potential problems with ambiguity when a term can be used both as a noun (*the order*) and a verb (*to order*). Therefore, we use the two-phase approach of [11] for deriving annotations. In the *style recognition phase*, the label style is determined. Contextual information is utilized to classify ambiguous cases. The *derivation phase* yields the action, the business object, and optional fragments. This step builds on the capability of the lexical database WordNet [12] to derive a verb like *register* from the nominalized action *registration*.

3.2 Generation of Semantic Match Hypotheses

The generation of semantic match hypotheses builds on the annotation of activities. It yields a similarity score for each activity pair of the two input models.

The general idea for this phase is to calculate the score based on the semantic similarity between the actions, the business objects and the additional fragments of the considered activity pair. In this context, the term semantic similarity refers to the closeness of two concepts in the taxonomy WordNet [12]. Different proposals exist for calculating the similarity between two concepts based on taxonomies [13,14,15]. Here, we utilize the similarity measure introduced by Lin, as it has been shown to correlate well with human judgments [16]. Forcalculating this semantic similarity between two labels l_1, l_2, we introduce three functions: a component similarity function sim_c, a coverage function cov, and a label similarity function sim_l, combining the latter two to a final result.

The function sim_c calculates the semantic similarity between two label components l_{c_1} and l_{c_2}. In general, the result of the Lin measurement is returned. If not both labels include the component, the value is set to zero.

$$sim_c(l_1, l_2) = \begin{cases} 0 & \text{if } l_{1_c} = \emptyset \vee l_{2_c} = \emptyset \\ Lin(l_{1_c}, l_{2_c}) & \text{if } l_{1_c} \neq \emptyset \wedge l_{2_c} \neq \emptyset \end{cases} \tag{1}$$

The coverage function cov is used to determine the number of components in a label l. Assuming a label at least refers to an action, the result of cov ranges from 1 to 3. Note that the index a in the definition denotes the action, bo the business object and add the additional information fragment.

$$cov(l) = \begin{cases} 1 & \text{if } l_a \neq \emptyset \wedge l_{bo} = \emptyset \wedge l_{add} = \emptyset \\ 2 & \text{if } l_a \neq \emptyset \wedge (l_{bo} \neq \emptyset \veebar l_{add} \neq \emptyset) \\ 3 & \text{if } l_a \neq \emptyset \wedge l_{bo} \neq \emptyset \wedge l_{add} \neq \emptyset \end{cases} \tag{2}$$

In order to combine the individual similarity results, we introduce the function sim_l. This function calculates the arithmetic mean of the similarity values for action, business object and the additional information. This is accomplished by dividing the sum of sim_a, sim_{bo} and sim_{add} by the maximum coverage among l_1 and l_2. As a result, we obtain the overall matching weight for two given labels.

$$sim_l(l_1, l_2) = \frac{sim_a(l_1, l_2) + sim_{bo}(l_1, l_2) + sim_{add}(l_1, l_2)}{\underset{l \,\in\, \{l_1, l_2\}}{\arg\max}\, cov(l)} \tag{3}$$

By calculating sim_l for every activity pair which can be combined from the considered process models, we obtain a set of match hypotheses. This set of hypotheses constitutes the first input for our probabilistic matching model.

3.3 Constraints Generation

Constraint satisfaction, also called second line matching [17], is often applied in schema and ontology matching as a means to guide the selection of matches. Here, constraints may relate to the general structure of matches (e.g., only 1:1 matches shall be considered), particular attribute pairs (e.g., a pair forms a matches or shall never be part of any match), or dependencies between different matches. We aim at matching such dependencies which are related to the execution semantics of process models. The intuition behind is that the order of processing described by one model is likely to coincide with the order of processing specified in a second model. Referring to the initial example in Figure 1, we see that in either model the activities related to check an application (e.g., *Check application in time* in the upper model and *Check documents* in the lower model) are preceding the activities related to taking a decision (e.g, *Reject application* and *Send letter* *of rejection*). Also, activities for accepting an application are exclusive to those of rejection an application in either model.

There are different alternatives to formulate behavioral constraints for a process model. For the context of matching process models, a fine-grained formalization of constraints appears to be appropriate. Although we assume two models to show a rather consistent order of processing, slight deviations can always be expected and should have a minor impact on the matching process. Therefore, we consider a model that captures order constraints for the smallest possible conceptual entity, i.e., pairs of activities. Further, in many cases, the final matching will only be partial, meaning that activities of one model are without counterpart in the other model. This suggests to not rely on direct successorship of activities but on a notion that is insensitive of partial matchings.

Against this background, we capture behavioral constraints using a binary relation over activities, called weak order [18]. It holds between two activities a_1 and a_2 of a process model, if there exists an execution sequence in which a_1 occurs before a_2. By referring to the existence of a certain execution sequence, it allows for capturing the potential order of occurrence for activities. In the aforementioned example, weak order holds between *Check application in time* and *Reject application* in the upper model, and between *Check documents* and *Send letter of rejection* in the lower model. The exclusiveness of activities representing acceptance and rejection of an application in either model is implicitly covered: the respective activities are not related by weak order in either direction. The strict order is also implied if weak order is only defined in one direction.

Weak order of activities can be derived from the state space of a process model. For certain classes of models, however, the relation can also be derived directly from the structure. For models that incorporate only basic control flow routing, such as XOR and AND routing constructs, and that show soundness, i.e., the absence of behavioral anomalies such as deadlocks, the weak order relation is determined in low polynomial time to the size of the model [18].

3.4 Probabilistic Match Optimization

An instance of the process matching problem consists of the two processes, the match hypotheses with a-priori confidence values, and the behavioral relations holding between the activities. Statistical relational languages such as Markov logic [19] are a natural choice when uncertainty meets relational data. We will demonstrate that Markov logic is an appropriate choice for a process matching framework as it is adaptable to different matching situations and allows fast prototyping of matching formulations.

Markov Logic Networks. Markov logic [19] is a first-order template language for log-linear models with binary variables. Log-linear models are parameterizations of undirected graphical models (Markov networks) which play an important role in the areas of reasoning under uncertainty [20] and statistical relational learning [21]. Log-linear models are also known as maximum-entropy models in the natural language processing community [22]. The *features* of a log-linear model can be complex allowing the user to incorporate prior knowledge about the importance of features of the data for classification. Moreover, within the framework of log-linear models users can specify *constraints* on the resulting classification. In the context of process matching, these constraints will allow us to punish inconsistent sets of matches, also referred to as alignments.

A Markov network $\mathcal{M}$ is an undirected graph whose nodes represent a set of random variables $\mathbf{X} = \{X_1, ..., X_n\}$ and whose edges model direct probabilistic interactions between adjacent nodes. More formally, a distribution P is a log-linear model over a Markov network $\mathcal{M}$ if it is associated with:

- a set of features $\{f_1(D_1), ..., f_k(D_k)\}$, where each D_i is a clique in $\mathcal{M}$ and each f_i is a function from D_i to $\mathbb{R}$,
- a set of real-valued weights $w_1, ..., w_k$, such that

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{k} w_i f_i(D_i)\right),$$

where Z is a normalization constant [20].

A Markov logic network is a set of pairs (F_i, w_i) where each F_i is a first-order formula and each w_i a real-valued weight associated with F_i. With a finite set of constants C it defines a log-linear model over possible worlds $\{\mathbf{x}\}$ where each variable X_j corresponds to a ground atom and feature f_i is the number of true groundings (instantiations) of F_i with respect to C in possible world $\mathbf{x}$. Possible worlds are truth assignments to all ground atoms with respect to the set of constants C. We explicitly distinguish between weighted formulas and *deterministic* formulas, that is, formulas that always have to hold.

There are two common types of probabilistic inference tasks for a Markov logic network: Maximum a-posteriori (MAP) inference and marginal probability inference. The latter computes the posterior probability distribution over a subset of the variables given an instantiation of a set of evidence variables. MAP inference, on the other hand, is concerned with finding an assignment to the variables with maximal probability. Assume we are given a set $\mathbf{X}' \subseteq \mathbf{X}$ of instantiated variables and let $\mathbf{Y} = \mathbf{X} \setminus \mathbf{X}'$. Then, a most probable state of the ground Markov logic network is given by

$$\underset{\mathbf{y}}{\text{argmax}} \sum_{i=1}^{k} w_i f_i(D_i).$$

Similar to previous work on matching ontologies with Markov logic[23,24], we can specify a set of hard and soft constraints that improve the overall matching results. Finding the most likely alignment then translates to computing the maximum a-posteriori state of the ground Markov logic network.

Markov Logic Formulation of Process Matching. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the activities of two process models to be mapped, we describe each process model in terms of weak order relations $wo_1 : \mathcal{A}_1 \times \mathcal{A}_1$ and $wo_2 : \mathcal{A}_2 \times \mathcal{A}_2$. Furthermore, the mapping hypotheses are represented by a mapping relation $map : \mathcal{A}_1 \times \mathcal{A}_2$. In the Markov logic formulation, the relations wo_1 and wo_2 are modeled using observable predicates, that is, predicates whose ground state is known a-priori whereas the relation *map* is modeled using a hidden predicate. Hence, when an optimal alignment between activities in the two models is computed, we model the weak-order relations as observed predicates and the map relation as a hidden predicate. For convenience, we also define the strict order and the exclusiveness relation between activities of a process model as follows:

$$so_i(a_i, b_i) \Leftrightarrow wo_i(a_i, b_i) \wedge \neg wo_i(b_i, a_i)$$
$$ex_i(a_i, b_i) \Leftrightarrow \neg wo_i(a_i, b_i) \wedge \neg wo_i(b_i, a_i)$$

Using these relations, we can simply represent the constraints as a set of first-order formulas and add those to the Markov logic formulation. The knowledge base consists of the output of the base matcher encoded in terms of weighted atoms of the map relation acting as evidence plus two sets of atoms of the order relations mentioned above as static knowledge. The final result of the matching process is now computed by adding additional constraints and computing the a posteriori probability of the map atoms. We experimented with different types of constraints that have proven useful in the area of ontology matching and which we adapted to the case of process matching.

Cardinality. It has been shown that restricting alignments to one-to-one matches typically leads to better results in ontology matching. In particular, because gold standard alignments in this area tend to be one-to-one. While this is clearly not the case for process matching, as processes are often described at different levels of granularity, the cardinality of the mapping relation is still an important constraint to avoid a too strong bias towards an alignment with too many erroneous matches. Therefore, we stick to a cardinality constraint encoded using the formula with $n = 1$:

$$|\{activitiy(a)|\exists b : map(a, b)\}| < n$$

Stability. Stability is a constraint expressing that the structural properties of the matched objects should be as identical as possible [25]. In particular, stability means that semantic relations that hold between two elements in one representation should also hold between the two elements in the representation they are mapped to. For process matching, we can define this notion of stability for the three order relations mentioned above, namely the weak order, strict order, and exclusiveness relation, by using the following implicitly universally quantified formulas where $a_i, b_i, i \in \{1, 2\}$, are activities in process model i.

$$wo_i(a_i, b_i) \wedge \neg wo_j(a_j, b_j) \Rightarrow \neg(map(a_1, a_2) \wedge map(b_1, b_2)) \text{ with } i, j \in \{1, 2\}, i \neq j$$

$$so_i(a_i, b_i) \wedge \neg so_j(a_j, b_j) \Rightarrow \neg(map(a_1, a_2) \wedge map(b_1, b_2)) \text{ with } i, j \in \{1, 2\}, i \neq j$$

$$ex_i(a_i, b_i) \wedge \neg ex_j(a_j, b_j) \Rightarrow \neg(map(a_1, a_2) \wedge map(b_1, b_2)) \text{ with } i, j \in \{1, 2\}, i \neq j$$

Note that these constraints do not need to be hard. Indeed, our empirical results have shown that in the process matching setting, constraints should be soft, making alignments that violate them possible but less likely.

Coherence. A weaker class of constraints are those that encourage *logical coherence* of the integrated model. More specifically, such constraints exclude conflicting combinations of semantic relations in the integrated model. In the case of process matching, coherence criteria can be formulated using order relations. The basic idea is that activities that are exclusive in one of the models should not be in a weak order or a strict order relation in the other model.

$$so_i(a_i, b_i) \wedge ex_j(a_j, b_j) \Rightarrow \neg(map(a_1, a_2) \wedge map(b_1, b_2)) \text{ with } i, j \in \{1, 2\}, i \neq j$$

Another form of incoherence results when the strict order relations of aligned activities in the two models are inverted leading to a conceptual conflict in the merged process model. The constraint making alignments that cause this kind of incoherence less likely is sometimes referred to as 'criss-cross mappings' in the ontology matching setting and can be formalized as follows.

$$so_i(a_i, b_i) \wedge so_j(b_j, a_j) \Rightarrow \neg(map(a_1, a_2) \wedge map(b_1, b_2)) \text{ with } i, j \in \{1, 2\}, i \neq j$$

Note that coherence is a weaker than stability, not enforcing a semantic relation to hold, but only excludes incompatible relations between mapped elements.

4 Evaluation

In this section we present an evaluation of the defined concepts. More specifically, Section 4.1 describes the sample of admission process models from different German universities that we use to that end. Section 4.2 summarizes the results for applying probabilistic match optimization using Markov logic networks. Section 4.3 compares the results of our optimized semantic matching approach with syntactic matching in ICoP. Furthermore, we discuss the results of the two approaches in terms of their strengths and weaknesses.

4.1 Study Admission Processes of Nine German Universities

Up until now, there is no commonly accepted sample available for testing process model matching algorithms for process. Therefore, we created such a sample based on modeling projects of graduate students from Humboldt-Universität zu Berlin, Germany. These students participated in a research seminar on process modeling in three different semesters. The task of this seminar was to document the study admission process of a Germany university, and to compare the process with those of other student groups. This exercise yielded nine admission process models from different universities, which were created by different modelers using different terminology and capturing activities at different levels of granularity. All processes were modeled in BPMN, while the formal analysis was conducted on a corresponding Petri net representation. The minimum number of activities in a process model is 10 ranging up to 44. On average, a process model has 21 activities in this sample.

The combination of those nine processes results in $9 * 8/2 = 36$ model pairs. In order to build our test sample, we involved three researchers in building the gold standard of the pairwise activity mappings. Matches were identified by two researchers independently, and the third researcher resolved those cases where different matches were proposed. We used the process models and the gold standard as input of two matching tools. We used the existing ICoP prototype for generating 1:1 matches, for short ICoP. For the approach presented in this paper, we implemented a separate prototype that incorporated different components for annotation [11], for constraint generation [18], TheBeast[1] for Markov logic networks [26], and the mixed integer programming solver Gurobi[2] to solve integer linear programs derived from the Markov logic networks. For short, we refer to this second prototype as Markov. Both of these matching prototypes were utilized to automatically generate matches between activities for each pair of process models. Those matches were compared with the matches defined in the gold standard. Using the gold standard, we can classify each proposed activity match as either true-positive (TP), true-negative (TN), false-positive (FP) or false-negative (FN). These sets provide the basis for calculating the *precision* (TP/(TP+FP)) and *recall* (TP/(TP+FN)) metrics. We will also report the F_1 measure, which is the harmonic mean of precision and recall (2∗precision∗recall/(precision+recall)).

4.2 Evaluation of Match Optimization

In this section, we investigate in how far the matching result benefits from the stability and coherence constraints as incorporated in the Markov prototype. To this end, we conducted experiments with different combinations of soft constraints each with a weight of 0.1. All experiments were conducted on a PC with AMD Athlon Dual Core Processor 5400B with 2.6GHz and 1GB RAM. Our conjecture was that by the help of the constraints and the Markov logic network optimization we would improve precision without compromising recall too much. If so, the corresponding F_1 value should increase. Our base case is a configuration without any constraints, which yielded 0.079 precision, 0.572 recall, and an F_1 of 0.136.

Table 1 summarizes the findings. The initial introduction of a 1:1 match cardinality constraint improves the results towards an F_1 score of 0.27, with precision and recall at roughly 0.28. We use this configuration to introduce the three types of stability constraints. It can be seen that both types of order constraints improve the match results, the F_1 score rises to 0.315 and 0.316, respectively. Strict

[1] `http://code.google.com/p/thebeast/`

[2] `http://www.gurobi.com/`

Table 1. Precision, Recall, F_1 and processing time for different constraint types

configuration	precision	stddev.	recall	stddev.	F_1	stddev.	avg. time [s]
no constraints	0.079	0.033	0.572	0.205	0.136	0.052	1.1
cardinality 1:1	0.278	0.172	0.280	0.228	0.270	0.193	1.3
1-1 cardinality with							
weak order stability	0.421	0.217	0.263	0.170	0.315	0.182	109.2
strict order stability	0.354	0.216	0.304	0.236	0.316	0.216	41.5
exclusiveness stability	0.280	0.174	0.234	0.174	0.247	0.170	50.3
so-exclusiveness coherence	0.306	0.179	0.252	0.178	0.268	0.171	45.3
so-so coherence	0.342	0.195	0.317	0.226	0.318	0.197	16.7

Table 2. Precision, Recall, F_1 and processing time for Markov and ICoP

prototype	precision	stddev.	recall	stddev.	F_1	stddev.
Markov (weak order stability)	0.421	0.217	0.263	0.170	0.315	0.182
ICoP	0.506	0.309	0.255	0.282	0.294	0.253

order coherence yields a comparable result. Exclusiveness-related stability and coherence prove to be less effective. The F_1 score is lower due to a loss in recall.

These results suggest that order relations appear to be helpful in finding correct and ruling out incorrect matches. In comparison to the base case, the results improve from 0.136 to 0.315 for weak order stability in terms of the F_1 score. Compared to the case with only cardinality constraints ($F_1 = 0.27$), weak order stability yields a considerably better precision at the expense of a small loss in recall. This points to the potential of order constraints to inform automatic process matching.

4.3 Semantic versus Syntactic Matching

After having demonstrated the benefits of constraint optimization in the Markov prototype, this section aims to investigate in how far its usage of semantic match hypotheses advances beyond the syntactic match strategies of ICoP. We approach this question by considering the average precision, recall and F_1 measure for the admission process sample along with their standard deviation.

Table 2 provides the figures for comparing the Markov prototype and the existing ICoP prototype. It can be seen that ICoP achieves a better precision, but a weaker recall. However, the Markov prototype yields a better F_1 measure of 0.315 in comparison to 0.294. It is interesting to note that the Markov prototype achieves these results with a much lower standard deviation. The difference in standard deviation ranges from 0.071 up to 0.112. We might see in this difference an indication that the Markov prototype is more robust and less sensitive to specific characteristics of the process pair to be matched.

In order to understand which characteristics might favour one or the other approach, we plotted the F_1 measure for both as shown in Figure 2. For 20 of

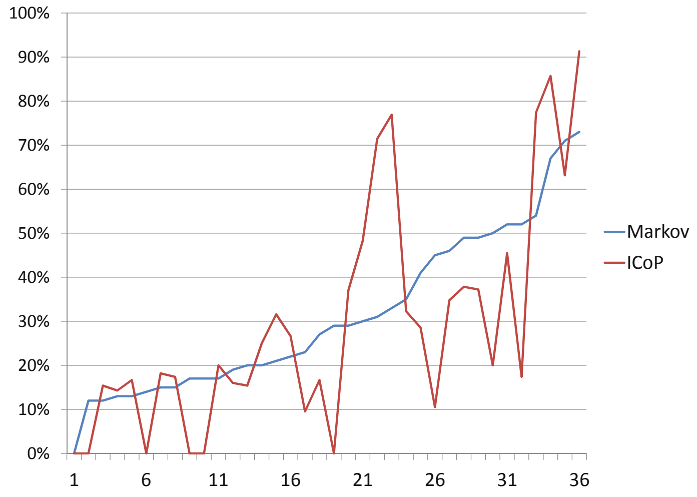


Fig. 2. F_1 measure of Markov and ICoP for the 36 match pairs, ordered by Markov result

the 36 pairs Markov yielded better results, while ICoP was better in 16 cases. There are a few pairs with substantial difference: In three cases ICoP is better with a difference of more than 0.20, namely 0.234, 0.404, and 0.439. For four pairs, Markov is better with a difference of 0.290, 0.300, 0.345 and 0.346. We aim to illustrate the three classes of *comparable results*, *better ICoP*, and *better Markov* results by the help of three characteristic process model pairs.

Comparable Results: If comparable results are observed for both approaches, the resulting F_1 values remain in the lower range. The pair FU Berlin and TU Munich is one such example where Markov yields 0.20 and ICoP 0.25. The FU Berlin process has 21 activities and is described on a more fine-granular level than the TU Munich process with its 11 activities. There are seven 1:1 matches between these models and three 1:n matches. Eight activities of FU Berlin have no counterpart in the TU Munich process, and one Munich activity has no match. Both approaches suffer from the fact that both processes contain several activities that mention the same verb: the FU Berlin process has two activities with *to add* (*Add Certificate of Bachelor Degree* and *Add Certificate of German language*), four activities involving *to check* and three *send* activities; the TU Munich process has four *send* activities. ICoP provides one false-positive and eleven false-negatives; Markov has five false-positive and eleven false-negatives.

Better ICoP: ICoP yielded significantly better results for the match pair Cologne-Frankfurt (F_1 of 0.76 in comparison to 0.33 by Markov). The Cologne process has 10 activities, Frankfurt 12. There are six 1:1 matches and no 1:n matches. Four and six activities on each side, respectively, have no match partner. Five of these matches are syntactically equivalent, another being a substring of its match (*Acceptance* and *Send letter of Acceptance*). While the

Table 3. Explorative results on relative strengths of Markov and ICoP

Match Pair	FU Berlin TU Munich	Cologne Frankfurt	Hohenheim Erlangen
Better Approach	Comparable	ICoP	Markov
Activities	21 11	10 12	25 30
1:1 Match	7	6	6
1:n Match	3	0	4
No Match	9	10	28
ICoP False-Positives Markov False-Positive	1 5	1 4	3 3
ICoP False-Negatives Markov False-Negatives	11 11	1 4	17 10
ICoP F_1 Markov F_1	0.25 0.20	0.77 0.33	0.17 0.52

good performance of ICoP is no surprise, it is interesting that the semantic approach in Markov shows weak results. There are four false positive, which are semantically very close, but no match for this model pair (e.g. *Take Aptitude Test* and *Take Oral Exam*). As a consequence, the probabilistic optimizer penalizes some syntactically equal and correct matches. Markov could be improved by generating 100% confidence match hypotheses for syntactically identical activities. It is interesting to note that also the second case of superior performance of ICoP can be traced back to a great share of syntactically identical matches.

Better Markov: The processes for Hohenheim and Erlangen are much better matched by Markov than by ICoP. The two process models of this match pair have 25 and 30 activities, respectively. There are six 1:1 matches, four 1:n matches, and 28 activities without a match in the other model. While ICoP yields a low F_1 of 0.17, Markov achieves a respectable 0.52. ICoP only finds three correct matches, all being syntactically closely related (e.g. *Checking if complete* and *Check Application Complete*). It is interesting to find that Markov substantially benefits both from semantic match pairs and constraint optimization. Among others, the correct match *publishing the letters* and *send acceptance* is added by the help of the weak order stability and its semantic similarity. The weak order rule also helps to eliminate eight false matches including *Receiving the written applications* and *receive rejection.*

Table 3 summarizes the exploratory results on relative strengths of Markov and ICoP. The following three conclusions can be drawn from this evaluation, also from further investigation of the data. First, both approaches benefit from an increase in the number of 1:1 matches. The number of 1:1 matches is strongly correlated with the F_1 of both approaches for our sample with 0.646 and 0.637, respectively. Second, ICoP suffers from an increase in the number of not matched activities. We find a correlation of -0.143. Interestingly, there is no such correlation for Markov. Examples like the Hohenheim-Erlangen case suggest that the optimizer works well in filtering out unjustified match hypotheses based on weak order. Third, both approaches suffer from an increase in the number of 1:n matches. Interestingly, the decrease is much stronger for ICoP with a correlation of -0.461. For Markov, this correlation is only -0.166. Markov seems to benefit from semantic similarity in hypothesis generation, which turns out to be a remedy to some extent for representation on different levels of granularity. While these advantages of semantic matching appear to be stronger for larger models, there is the need to account for trivial matches that are syntactically the same. Markov has lost some share of its performance by not directly accepting such trivial matches. Nevertheless, it will be rather straight-forward to incorporate such strategies.

5 Related Work

The work presented in this paper mainly relates to two categories of related research, process model similarity and semantic matching.

Process model similarity techniques can be used to determine how similar two business process models are, usually measured on a scale from 0 to 1. There exists a variety of techniques that exploit textual information and the process model structure [2,1] or execution semantics [3,6]. An overview of these techniques is given in [1]. The relevance of process model similarity to process matching is twofold. First, often similarity techniques start by determining similarity of individual activities, which is clearly also of interest when determining matches. Second, similarity techniques often produce a mapping between activities as a byproduct of computing the similarity. The most important difference between similarity and matching is that, when computing the similarity between process models, a matching of lower quality is required than when the matching itself is the goal. Consequently, the similarity techniques are less advanced when it comes to determining matches. They mostly rely on simple (and fast) label comparison rather than semantic techniques to determine similarity of activities and neglect complex matches. There is one notable exception [9] that leverages synonyms from WordNet [12]. Our fine grained interpretation of activity labels, however, goes beyond the approach presented in [9].

Semantic matching has received considerable attention for schema and ontology matching, see [27,28,29]. In essence, semantic matching refers to the identification of relations (equivalence, more or less general, disjoint) between concepts, i.e., interpretations of schema or ontology entities [30]. Most prominently, the S-Match system [31] realized semantic matching by first interpreting labels and entities, which yields a set of concepts, before establishing relations between them. This approach heavily relies on external knowledge bases, such as WordNet [12]. Those are used to interpret single labels and derive concepts, but also to determine the semantic relations between them. Our approach for process model matching takes up these ideas: we interpret activity labels by extracting actions and business objects, i.e., concepts, to generate match hypothesis.

6 Conclusion

In this paper we presented a novel approach for automatic process model matching in two steps. First, we generate match hypotheses based on automatically annotated activity labels and leveraging a semantic interpretation of activity labels. Second, we make use of match constraints derived from behavioral relations of process models. These constraints are utilized for guiding the matching with a probabilistic model. The evaluation of our approach with admission processes from nine different universities shows that this novel conceptual basis indeed improves performance. We demonstrated that match results are more stable over different levels of process model heterogeneity. Moreover, our comparative analysis revealed strengths and weaknesses of classical matchers and semantic matching with probabilistic optimization.

This research provides valuable insights for advancing the field of process model matching. In future work we plan to improve our approach based on the identified weaknesses. This involves on the one hand a smooth integration of syntactical and semantic match hypotheses. On the other hand, we aim to experiment with further process-related constraints. For instance, we plan to work with hierarchical 1:n matches, which are non-overlapping. Finally, there is the potential to improve matching results based on domain ontologies or domain corpora. They might help to increase the accuracy of the calculated hypotheses.

References

1. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems 36, 498–516 (2010)
2. Grigori, D., Corrales, J.C., Bouzeghoub, M., Gater, A.: Ranking bpel processes for service discovery. IEEE T. Services Computing 3(3), 178–192 (2010)
3. Zha, H., Wang, J., Wen, L., Wang, C., Sun, J.: A workflow net similarity measure based on transition adjacency relations. Computers in Industry 61(5), 463–471 (2010)
4. Becker, M., Laue, R.: A comparative survey of business process similarity measures. Computers in Industry 63(2), 148–167 (2012)
5. Niemann, M., Siebenhaar, M., Schulte, S., Steinmetz, R.: Comparison and retrieval of process models using related cluster pairs. Computers in Industry 63(2), 168–180 (2012)
6. Kunze, M., Weidlich, M., Weske, M.: Behavioral Similarity – A Proper Metric. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 166–181. Springer, Heidelberg (2011)
7. Dijkman, R., Dumas, M., García-Bañuelos, L., Käärik, R.: Aligning business process models. In: IEEE International EDOC Conference 2009, pp. 45–53 (2009)
8. Weidlich, M., Dijkman, R., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010)
9. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Roddick, J., Hinze, A. (eds.) Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM 2007), vol. 67, pp. 71–80. Australian Computer Science Communications, Ballarat (2007)

10. Mendling, J., Reijers, H.A., Recker, J.: Activity labeling in process modeling: Empirical insights and recommendations. Inf. Syst. 35(4), 467–482 (2010)
11. Leopold, H., Smirnov, S., Mendling, J.: On the refactoring of activity labels in business process models. Information Systems 37(5), 443–459 (2012)
12. Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM 38(11), 39–41 (1995)
13. Lee, J.H., Kim, M.H., Lee, Y.J.: Information retrieval based on conceptual distance in is-a hierarchies. Journal of Documentation 49(2), 188–207 (1993)
14. Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE Transactions on Systems, Man, and Cybernetics 19(1), 17–30 (1989)
15. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: 14th International Joint Conference on Artificial Intelligence, vol. 1, pp. 448–453. Morgan Kaufmann Publishers Inc. (1995)
16. Shavlik, J.W.: An Information-Theoretic Definition of Similarity. In: Shavlik, J.W. (ed.) Proceedings of the 15th International Conference on Machine Learning (ICML 1998). Morgan Kaufmann (1998)
17. Gal, A.: Uncertain Schema Matching. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2011)
18. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Software Eng. 37(3), 410–429 (2011)
19. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
20. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
21. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press (2007)
22. Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT Press (1999)
23. Niepert, M., Meilicke, C., Stuckenschmidt, H.: A Probabilistic-Logical Framework for Ontology Matching. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (2010)
24. Noessner, J., Niepert, M.: CODI: Combinatorial Optimization for Data Integration–Results for OAEI 2010. In: Proceedings of the 5th Workshop on Ontology Matching (2010)
25. Meilicke, C., Stuckenschmidt, H.: Analyzing mapping extraction approaches. In: Proceedings of the Workshop on Ontology Matching (2007)
26. Riedel, S.: Improving the accuracy and efficiency of MAP inference for Markov logic. In: Proc. of UAI 2008, pp. 468–475 (2008)
27. Euzenat, J., Shvaiko, P.: Ontology matching. Springer (2007)
28. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. AI Magazine 26(1), 83–94 (2005)
29. Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record 33(4), 65–70 (2004)
30. Giunchiglia, F., Shvaiko, P.: Semantic matching. The Knowledge Engineering Review Journal 18(3), 265–280 (2003)
31. Giunchiglia, F., Yatskevich, M., Shvaiko, P.: Semantic Matching: Algorithms and Implementation. In: Spaccapietra, S., Atzeni, P., Fages, F., Hacid, M.-S., Kifer, M., Mylopoulos, J., Pernici, B., Shvaiko, P., Trujillo, J., Zaihrayeu, I. (eds.) Journal on Data Semantics IX. LNCS, vol. 4601, pp. 1–38. Springer, Heidelberg (2007)

Isotactics as a Foundation for Alignment and Abstraction of Behavioral Models*

Artem Polyvyanyy[1], Matthias Weidlich[2], and Mathias Weske[3]

[1] Queensland University of Technology, Brisbane, Australia
artem.polyvyanyy@qut.edu.au
[2] Technion – Israel Institute of Technology, Haifa, Israel
weidlich@tx.technion.ac.il
[3] Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany
mathias.weske@hpi.uni-potsdam.de

Abstract. There are many use cases in business process management that require the comparison of behavioral models. For instance, verifying equivalence is the basis for assessing whether a technical workflow correctly implements a business process, or whether a process realization conforms to a reference process. This paper proposes an equivalence relation for models that describe behaviors based on the concurrency semantics of net theory and for which an alignment relation has been defined. This equivalence, called isotactics, preserves the level of concurrency of aligned operations. Furthermore, we elaborate on the conditions under which an alignment relation can be classified as an abstraction. Finally, we show that alignment relations induced by structural refinements of behavioral models are indeed behavioral abstractions.

1 Introduction

Behavioral models can serve different purposes: communicating ideas, simulating systems, or defining precise execution instructions. Tailoring a model for a certain purpose leads to the existence of several "related" models of the same original. Each model shall be appropriate for its purpose. In business process management (BPM), behavioral models on the business level should, thus, concentrate on aspects that are important from a business perspective, while technical implementation aspects are disregarded. Technical models, in turn, need to describe activities required for implementation, such as data mapping or error handling.

Given a set of related models, it is often feasible to map semantically related, or *aligned*, (groups of) modeling constructs across models. Fig. 1 shows two aligned behavioral models captured using BPMN [1] language. Both models describe behaviors of performing "product at the market" research. Related groups of tasks are enclosed in the areas denoted by dotted borders and connected by dashed lines, e.g., task "*Study product*" in Fig. 1(a) is aligned with tasks "*Select product*" and "*Collect product info*" in Fig. 1(b) by the semantical concept α.

* This work was initiated while the first author was with Hasso Plattner Institute.

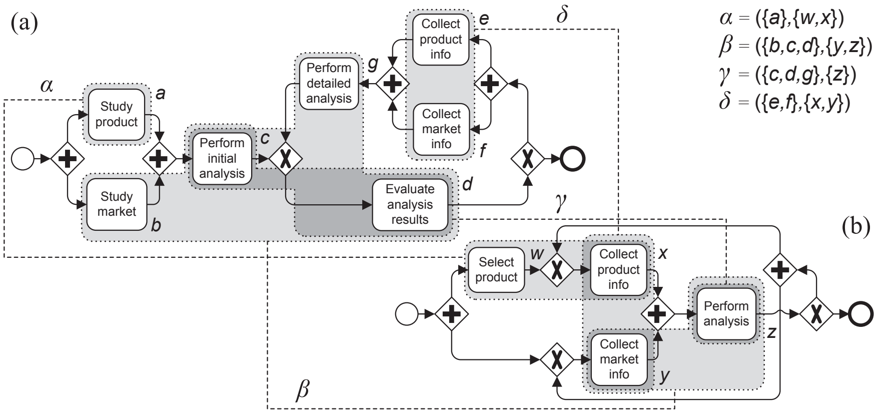

(a)
(b)
$\alpha = (\{a\},\{w,x\})$
$\beta = (\{b,c,d\},\{y,z\})$
$\gamma = (\{c,d,g\},\{z\})$
$\delta = (\{e,f\},\{x,y\})$
Study product
Study market
Perform initial analysis
Perform detailed analysis
Collect product info
Collect market info
Evaluate analysis results
Select product
Perform analysis
a
b
c
d
e
f
g
w
x
y
z
α
β
γ
δ

Fig. 1. Alignment of BPMN diagrams

There are many use cases in BPM that require the comparison of aligned behavioral models. Verifying equivalence, for instance, is the basis for assessing whether a technical workflow correctly implements a business process, or whether a process realization conforms to a reference process [2]. Further, abstraction of behavioral models, i.e., an alignment that implies information loss from one model to another, plays an important role in managing model complexity [3]. Despite these observations, as of today, there is a lack of formal grounding for verifying behavior equivalence without imposing any assumption on the structure of an alignment relation. Note that in general groups of aligned modeling constructs may be of arbitrary size and can even overlap.

In this paper, we study models that describe behavior as a partially ordered, usually infinite, set (poset) of events. Here, an *event* is a phenomenon located at a single point in time [4]. For these models, we answer the question of *how to define an equivalence relation that preserves order and concurrency of event occurrences without imposing any assumptions on the structure of the alignment*. To answer this question, we introduce the notion of *isotactics*, which allows for comparing aligned behavioral models, very much like bisimulation [5,6] allows the comparison of non-aligned models. Formally, isotactics is implemented using the concept of a *tactic*, i.e., a poset of groups of events labeled with the same semantical concepts of the alignment relation. As such, our contribution is a first step toward a spectrum of equivalences for aligned behavioral models. Moreover, we show that common structural abstraction techniques for behavioral models [7,8] indeed preserve our new equivalence notion; however, we also show that isotactics makes the limitations of such structural approaches explicit. Note that the alignment construction, i.e., the discovery of semantically related constructs, is taken for granted; it can either be performed manually or automatically [9,10].

We proceed as follows: The next section presents preliminary notions. Section 3 is devoted to the discussion of alignment of behavioral models, which leads to the definition of isotactics. Then, Section 4 studies how this notion can aid in explaining the abstraction relation between behavioral models. Section 5 elaborates on the application of proposed notions. Finally, we draw conclusions.

2 Preliminaries

First, Section 2.1 discusses Petri nets – a formalism to which many languages for modeling behavior can be traced back [11]. Section 2.2 talks about causal nets – a way of representing concurrent runs of net systems.

2.1 Petri Nets

Petri nets are a well-known formalism for modeling behaviors.

Definition 1 (Petri net). A *Petri net*, or a *net*, $N = (P, T, F)$ has finite disjoint sets P of *places* and T of *transitions*, and the *flow* relation $F \subseteq (P \times T) \cup (T \times P)$.

For a *node* $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ is the *preset*, and $x\bullet = \{y \mid (x, y) \in F\}$ is the *postset* of x. $Min(N)$ is the set of places of N with empty preset, i.e., $\{p \in P \mid \bullet p = \emptyset\}$. A node $x \in P \cup T$ is an *input* (*output*) of a node $y \in P \cup T$, iff $x \in \bullet y$ ($x \in y\bullet$). For $X \subseteq P \cup T$, let $\bullet X = \bigcup_{x \in X} \bullet x$ and $X\bullet = \bigcup_{x \in X} x\bullet$. For a binary relation R, we denote by R^+ (R^*) the transitive (and reflexive) closure of R.

In the graphical notation, places are represented by circles, transitions by rectangles, and flow relation by directed edges (see Fig. 2). Execution semantics of Petri nets is based on states and state transitions and best perceived as a "token game". The state of a net is represented by a *marking*, which describes a distribution of *tokens* on the net's places. Whether a transition is *enabled* at a marking depends on the tokens in its input places. An enabled transition can *occur*, which leads to a new marking of the net.

To formalize semantics, we identify the flow relation F with its characteristic function on the set $(P \times T) \cup (T \times P)$.

Definition 2 (Net semantics). Let $N = (P, T, F)$ be a net.

- $M : P \to \mathbb{N}_0$ is a *marking*, or a *state*, of N assigning each place $p \in P$ a number $M(p)$ of *tokens* in p; $\mathbb{N}_0$ denotes the set of all natural numbers including zero. With $[p]$, we denote the marking in which place p contains just one token and all other places contain no tokens. We identify M with the multiset containing $M(p)$ copies of p for every $p \in P$.
- For a transition $t \in T$ and a marking M of N, t is *enabled* at M, written $M[t\rangle$, iff $\forall p \in \bullet t : M(p) \geq 1$.
- If $t \in T$ is enabled at M, then t can *occur*, which leads to a new marking M' and the *step* $M[t\rangle M'$ of N with $M'(p) = M(p) - F(p, t) + F(t, p)$, $p \in P$.
- A *net system*, or a *system*, is a pair $S = (N, M_0)$, where M_0 is a marking of N. M_0 is called the *initial marking* of N.
- A sequence of transitions $\sigma = t_1 \dots t_n$, $n \in \mathbb{N}_0$, $t_i \in T$, $i \in 1 \dots n$, of net system $S = (N, M_0)$ is a *firing sequence* in S iff there exists a sequence of steps $(N, M_0)[t_1\rangle(N, M_1) \dots (N, M_{n-1})[t_n\rangle(N, M_n)$ which leads from marking M_0 to marking M_n via a (possibly empty) sequence of intermediate markings $M_1 \dots M_{n-1}$.

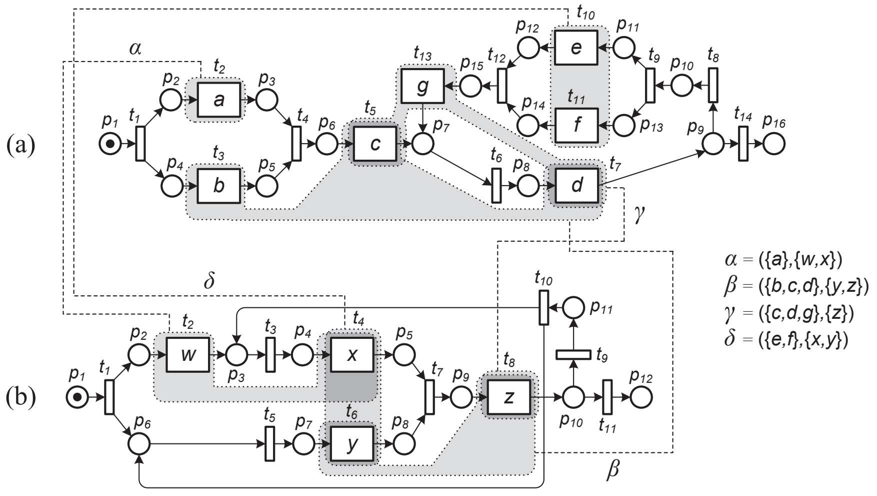


Fig. 2. Net systems that correspond to the BPMN diagrams in Fig. 1

- For any two markings M and M' of N, M' is *reachable* from M, denoted by $M' \in [N, M\rangle$ iff there exists a *run* of N, i.e., there exists a firing sequence σ leading from M to M'.

In the following, we shall refer to the *natural marking* of a net N; the natural marking puts one token at every place from the set $Min(N)$ and no tokens elsewhere. In the graphical notation, it is accepted that tokens are drawn as black dots inside places. Fig. 2 shows two net systems (in natural initial markings); the systems correspond to the BPMN diagrams in Fig. 1. Transitions which correspond to tasks in BPMN diagrams are drawn as rectangles with labels inside; the labels are the short-names of tasks which appear next to each task in Fig. 1.

2.2 Causal Nets and Processes

In this section, we present causal nets [4,12] and discuss how they can be used to capture *processes*, or *concurrent runs*, of net systems. Causal nets provide the foundation for general net theory [13].

Definition 3 (Causal net). A net $N = (B, E, G)$ is a *causal* net, iff :
- for each $b \in B$ holds $|\bullet b| \leq 1$ and $|b \bullet| \leq 1$, and
- N is acyclic, i.e., G^+ is irreflexive.

Elements of E are called *events* and elements of B are called *conditions*. The events of causal nets are usually used to describe occurrences of "atomic events", e.g., occurrences of transitions of a net system. An occurrence of an event e is associated with a state in which all its preconditions ($\bullet e$) hold, and the effect of its occurrence is that all its preconditions cease to hold, and all its postconditions ($e\bullet$) begin to hold [4]. Given a causal net $N = (B, E, G)$, the concurrency relation of N is defined by $\|_N = ((B \cup E) \times (B \cup E)) \setminus (G^+ \cup (G^+)^{-1})$ (we omit the subscript if the context is clear). Note that $||_N$ is symmetric and reflexive. Moreover, every two nodes of a causal net are either in the concurrency or in the (inverse) causal relation, where nodes x and y are causal if and only if xG^+y. The causal relation specifies a dependency between events of a causal net, such that if $e_1G^+e_2$, where $e_1, e_2 \in E$, then in the net system composed of the causal net and its natural initial marking, e_2 cannot occur without e_1 having priorly occurred.

A *process* of a net system is a causal net together with a mapping which allows interpreting the net as a concurrent run of the net system[1]. Prior to proceeding with defining the notion of a process, we present the notion of a *cut*. A *cut* of a causal net is the maximal co-set with respect to set inclusion, where a *co-set* is a set of pairwise concurrent conditions. A process is then defined as follows.

Definition 4 (Process). A *process* $\pi = (N_\pi, \rho)$ of a net system $S = (N, M_0)$, $N = (P, T, F)$, has a causal net $N_\pi = (B, E, G)$ and a function $\rho : B \cup E \rightarrow P \cup T$:
- ∘ $\rho(B) \subseteq P$, $\rho(E) \subseteq T$ (ρ preserves the nature of nodes),
- ∘ $Min(N_\pi)$ is a cut, which corresponds to the initial marking M_0, that is $\forall\ p \in P : M_0(p) = |\rho^{-1}(p) \cap Min(N_\pi)|$ (π starts at M_0), and
- ∘ $\forall\ e \in E\ \forall\ p \in P : (F(p, \rho(e)) = |\rho^{-1}(p) \cap \bullet e|) \wedge (F(\rho(e), p) = |\rho^{-1}(p) \cap e \bullet |)$ (ρ respects the environment of transitions).

We refer to S as the *originative* system of π. A process π of S is *initial*, iff $E = \varnothing$.

Given a run of a net system, one can construct a unique process induced by the run (observe that the inverse does not hold). The starting point of the construction is a causal net composed of conditions that correspond to places from the initial marking of the net system and no events. The construction proceeds by stepwise appending events to the causal net. Each appended event corresponds to a transition in the run. Events are appended in the order in which corresponding transitions appear in the run. Each fresh event e which gets appended to the causal net and has corresponding transition t is appended together with output conditions, which must correspond to output places of t. Note that input conditions of e, which must correspond to input places of t, must be chosen from the conditions of the causal net with empty postsets.

Every process of a net system describes a family of runs together with information on concurrent nodes that participate in the runs. Processes of a net system can be in a prefix relation. Process π' *is an extension of* process π if during construction of π', induced by some run of the system, it is possible to observe π. Consequently, process π *is a prefix of* π'.

Definition 5 (Prefix of a process). Let $\pi = (N_\pi, \rho)$, $N_\pi = (B, E, G)$, be a process of a net system. Let c be a cut of N_π and let $c^\downarrow$ denote the set of nodes $\{x \in B \cup E \mid \exists\ y \in c : (x, y) \in G^*\}$. A process π_c *is a prefix of* π up to (and including) c, iff $\pi_c = ((B \cap c^\downarrow, E \cap c^\downarrow, G \cap (c^\downarrow \times c^\downarrow)), \rho|_{c^\downarrow})$.

Fig. 3 shows two processes of the net system in Fig. 2(b), the process in Fig. 3(a) being a prefix of the one in Fig. 3(b). Here, event e_x corresponds to transition t_x of the originative system, i.e., $\rho(e_x) = t_x$ (for each event e_x), and condition

[1] Not to be confused with a business process or a process model.

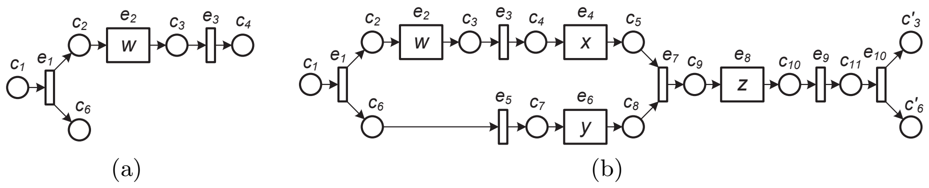


Fig. 3. Processes of the net system in Fig. 2(b)

c_y corresponds to place p_y of the originative system, i.e., $\rho(c_y) = p_y$ (for each condition c_y). Conditions c'_3 and c'_6 in Fig. 3(b) correspond to places p_3 and p_6 of the originative system, and represent second occurrences of respective places in the process. Both systems in Fig. 3 induce infinitely many processes.

3 Alignment

Alignment can be seen as the generic setting in which two models can be compared. We first reflect on the alignment of conceptual models in general in Section 3.1. Then, Section 3.2 turns the focus to net systems. Section 3.3 proposes the notion of isotactics to decide whether two aligned net systems show equivalent behaviors.

3.1 Alignment of Conceptual Models

The alignment of conceptual models has its roots in the field of data integration [14,15]. Despite terminological differences even in this field, cf., [16], a common interpretation defines an alignment as an association between semantically related entities of different models, e.g., between attributes of data schemas.

Following [15], an alignment consists of a set of *correspondences* between two models. Each correspondence relates two sets of entities of both models to each other. If both those sets are singletons, we speak of an elementary correspondence. Otherwise, the correspondence is called 1:n or n:m complex. The identification of correspondences, i.e., the construction of an alignment, is called *matching*.

A correspondence associates entities with each other, but does not define the semantics of this relation. Semantics is defined by extending an alignment toward a *mapping* comprising mapping expressions. Those are directed and define how the instances of entities of one model are transformed into instances of entities of another model. Consider a 2:1 complex correspondence between integer attributes of data schemas. A mapping expression may define the sum of two values from one schema as equivalent to a single value in the other schema.

Alignments and mappings of conceptual models may be checked for validity using a variety of properties. In the field of data integration, for instance, satisfiability and losslessness have been investigated [17]. The former holds for a mapping between two schemas if there is a pair of instances of either schema, i.e., a pair of data value tuples, that satisfies the constraints of the mapping. Losslessness relates to the result set that may be queried. If a mapping is lossless, all instances returned by a query on one schema have a counterpart in the other schema that is derived by the mapping.

3.2 Alignment of Behavioral Models

The notion of an alignment discussed for conceptual models in general can directly be applied to behavioral models, e.g., Petri net systems. Correspondences are then defined between sets of semantically related activities, i.e., transitions of different net systems. Formally, we capture such an alignment as follows.

Definition 6 (Alignment of net systems)
Let $S_1 = (N_1, M_1)$, $N_1 = (P_1, T_1, F_1)$, and $S_2 = (N_2, M_2)$, $N_2 = (P_2, T_2, F_2)$, be net systems. A set $\bowtie \subseteq \mathcal{P}_{\geq 1}(T_1) \times \mathcal{P}_{\geq 1}(T_2)$ is called an *alignment* of S_1 and S_2.[2]

Given an alignment $\bowtie$, we denote by $dom_{\bowtie}$ and $cod_{\bowtie}$ the domain and codomain of $\bowtie$, respectively. Recently, approaches for identifying correspondences between behavioral models have been presented [9,10]. Even though fully automatic identification of correspondences is hard to achieve, these works provide support for constructing an alignment in a semi-automated manner.

Again, correspondences between behavioral models capture only the relatedness, not the exact semantics in the sense of a mapping. That is, corresponding activities may not relate to the same real-world activities. For the models in Fig. 1, for instance, "*Study product*" may involve more than "*Select product*" and "*Collect product info*". Nevertheless, the activities are semantically related and are considered to be equivalent for any analysis of the alignment. For a complex correspondence, sets of activities are considered to be equivalent. As such, analysis of an alignment is founded on these sets instead of single activities.

Before, we discussed properties of alignments in the field of data integration, i.e., satisfiability and losslessness. These properties may be translated into the domain of behavioral models. Satisfiability then requires the existence of a single process that is possible in two net systems after the corresponding transitions have been resolved. Apparently, this is a rather weak requirement. Drawing the analogy to behavioral models for losslessness yields a stricter criterion. It requires that the characteristics of all processes of one net system are preserved in the processes of the other net system once the correspondences have been resolved.

3.3 Isotactics of Aligned Behavioral Models

To decide if two net systems show equivalent behaviors under a given alignment, we rely on a comparison of their processes. Therefore, we first need to clarify how a single process is interpreted once we consider not only single events, but groups thereof as being semantically related. Given a process of a net system and subsets of its transitions, a set abstraction of the process captures its interpretation by relating to all events that represent occurrences of transitions from the subsets.

Definition 7 (Process set abstraction)
Let $S = (N, M_0)$, $N = (P, T, F)$, be a net system, $\pi = (N_\pi, \rho)$, $N_\pi = (B, E, G)$, be a process of S, and $\kappa \subseteq \mathcal{P}_{\geq 1}(T)$. The *set abstraction of* π *with respect to* κ, denoted by $\alpha_\kappa(\pi) = (H, \prec, \xi)$, is defined by the set of events $H = \{e \in E \mid \exists\, k \in \kappa : \rho(e) \in k\}$,

[2] $\mathcal{P}_{\geq 1}(S)$ denotes the set of all non-empty subsets of a set S, including S itself.

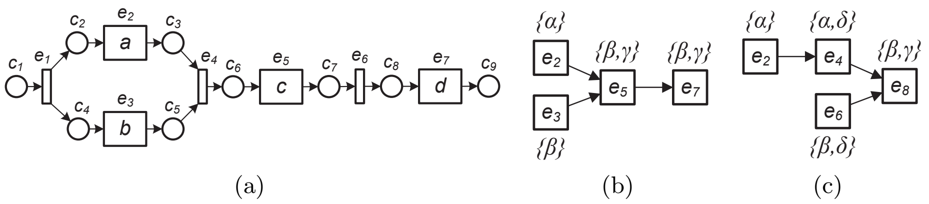


Fig. 4. (a) A process of the net system in Fig. 2(a), (b) the set abstraction of the process in (a), and (c) the set abstraction of the process in Fig. 3(b)

the relation $\prec$, which is the restriction of the causal relation of N_π to H, and the function $\xi : H \to \mathcal{P}_{\geq 1}(\kappa)$ such that $\xi(e) = \{k \in \kappa \mid \rho(e) \in k\}$, $e \in H$.

Fig. 4(b) and Fig. 4(c) show set abstractions of the processes in Fig. 4(a) and Fig. 3(b), respectively. In both abstractions, we use the sets of transitions induced by the alignment depicted in Fig. 1 as κ. In the figures, boxes represent events whose corresponding transitions belong to at least one set in κ, i.e., they are visible with respect to κ; note that other events are considered to be silent for the purpose of alignment. Edges encode causal relations between events, e.g., $e_2 \prec e_5$ and $e_3 \prec e_5$ in Fig. 4(b). Finally, every event $e \in H$ gets labeled with a subset of κ; the subset is composed of elements of κ which contain the transition that corresponds to e, e.g., event e_5 in Fig. 4(b) corresponds to transition t_5 in Fig. 2(a), which is induced by task "*Perform initial analysis*" in Fig. 1 that participates in β and γ correspondences of the alignment. Essentially, a process set abstraction is an *elementary event structure* [4] composed of events that are visible as much as the alignment is concerned. In [4], the authors accept the equality of elementary event structures as an appropriate equivalence notion for causal nets; the claim is supported by proposing translations between both notations. We agree with this line of argument and accept two processes as equivalent if and only if their set abstractions are isomorphic, i.e., if and only if one can define a causality-preserving bijection between events.

As a next step, we relate process set abstractions to an alignment between net systems, that is, we decide whether an alignment between set abstractions of two processes of the net systems can be deduced from the given alignment between their transitions. This is the case if events in both set abstractions can be partitioned such that one can define a bijection relation between the partitions for which any two events taken from one abstraction and different parts of the partition, and any two events taken from the related parts of the partition of the other abstraction, are causally related in a similar way[3].

We refer to the partitions of events in set abstractions (see the discussion above) as process *tactics*. Let $(H_i, \prec_i, \xi_i)$ be a process set abstraction and let $h_1 \subseteq H_i$ and $h_2 \subseteq H_i$ be non-empty disjoint sets of events, then h_1 and h_2 are in *causal* relation, written $h_1 \prec_i h_2$, iff for every pair $(e_1, e_2) \in h_1 \times h_2$ holds $e_1 \prec_i e_2$; note that in the following we shall omit subscript i where the context is clear.

[3] A partition of a set is a collection of disjoint subsets of the set whose union is the set.

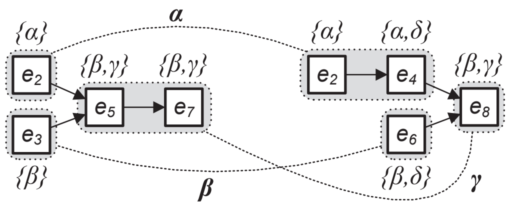


Fig. 5. Aligned process set abstractions

Definition 8 (Process tactic). Let $\alpha = (H, \prec, \xi)$ be the set abstraction of process π w.r.t. κ. A partition $\mathcal{H}$ of H is a *tactic* of α w.r.t κ iff :

- for every part $h \in \mathcal{H}$ there exists $k \in \kappa$ such that for every event $e \in h$ holds $k \in \xi(e)$, i.e., $\forall\ h \in \mathcal{H}\ \exists\ k \in \kappa\ \forall\ e \in h : k \in \xi(e)$ ($\mathcal{H}$ respects κ), and
- for every two parts $h_1, h_2 \in \mathcal{H}$, $h_1 \neq h_2$, holds either $h_1 \prec h_2$, or $h_2 \prec h_1$, or for each $(e_1, e_2) \in h_1 \times h_2$ holds $(e_1, e_2), (e_2, e_1) \notin \prec$ ($\mathcal{H}$ respects causality).

Every part of a tactic describes a *complex* event which stands for an occurrence of at least one and usually several semantically (by alignment) related transitions of the net system. A tactic, therefore, can be seen as a poset of complex events. Every set abstraction of a process has a *trivial* tactic, i.e., a tactic in which each of its parts is a singleton. Usually, a process set abstraction can be characterized by several tactics. Finally, aligned process set abstractions are defined as follows.

Definition 9 (Aligned process set abstractions)
Let π_1 and π_2 be processes of net systems S_1 and S_2, respectively. Let $\bowtie$ be an alignment of S_1 and S_2. Process set abstractions $\alpha_{dom_\bowtie}(\pi_1) = (H_1, \prec_1, \xi_1)$ and $\alpha_{cod_\bowtie}(\pi_2) = (H_2, \prec_2, \xi_2)$ are *aligned with respect to* $\bowtie$, denoted by $\alpha_{dom_\bowtie}(\pi_1) \bowtie \alpha_{cod_\bowtie}(\pi_2)$, iff there exist tactics $\mathcal{H}_1$ and $\mathcal{H}_2$ of π_1 and π_2, respectively, and a bijection $\chi : \mathcal{H}_1 \to \mathcal{H}_2$ such that: (i) for every $\chi(h_1) = h_2$, $h_1 \in \mathcal{H}_1$, there exists $(x, y) \in \bowtie$ such that $\forall e \in h_1 : x \in \xi_1(e)$ and $\forall e \in h_2 : y \in \xi_2(e)$ (χ respects alignment), and (ii) $\forall u, v \in \mathcal{H}_1 : u \prec_1 v \Leftrightarrow \chi(u) \prec_2 \chi(v)$ (χ respects causality).

We refer to χ as the alignment between tactics of process set abstractions. We say that processes are aligned if their abstractions are aligned. Apparently, the two set abstractions in Fig. 4(b) and Fig. 4(c) are not equivalent in the sense of [4], i.e., there exists no causality-preserving bijection between event sets. Nevertheless, one can rely on tactics to compare these set abstractions. Fig. 5 shows aligned process set abstractions from Fig. 4(b) and Fig. 4(c). In the figure, areas denoted by dotted borders with grey backgrounds define tactics (those which participate in the alignment). The dashed lines depict a bijection relation between the tactics and are labeled with semantical correspondences of the alignment from Fig. 1. Observe that the part $\{e_5, e_7\}$ of the tactic on the left can also be related to the part $\{e_8\}$ of the tactic on the right by using correspondence β; however, the existence of a correspondence is sufficient to decide for alignment.

Having defined the alignment of processes, we are able to define when the behavior of one net system can be mirrored by another net system under a given alignment. A system *covers the tactic* of another system if every process of the former system has a corresponding process in the latter system which mimics the behavior once abstractions have been applied. Formally, we capture this by a set that comprises pairs of aligned processes from both systems and require that the set is closed under process extensions. Note that the style of the next definition is inspired by the definitions of concurrent bisimulations in [18].

Definition 10 (Tactic coverage)
Let $\bowtie$ be an alignment of net systems S_1 and S_2. S_2 *covers the tactic of* S_1 *with respect to* $\bowtie$, denoted by $S_1 \leqslant_{\bowtie} S_2$, iff there exists a set $\mathcal{I} \subseteq \{(\pi_1, \pi_2)\}$ such that:
(i) π_1 is a process of S_1 and π_2 is a process of S_2.
(ii) If π_0^1 and π_0^2 are the initial processes of S_1 and S_2, respectively, $(\pi_0^1, \pi_0^2) \in \mathcal{I}$.
(iii) If $(\pi_1, \pi_2) \in \mathcal{I}$, then $\alpha_{dom_{\bowtie}}(\pi_1) \bowtie \alpha_{cod_{\bowtie}}(\pi_2)$ holds.
(iv) For each $(\pi_1, \pi_2) \in \mathcal{I}$ holds that if π_1' is an extension of π_1 then there exists $(\pi_1', \pi_2') \in \mathcal{I}$ where π_2' is an extension of π_2.
(v) For each $(\pi_1, \pi_2) \in \mathcal{I}$ holds that if π_1' is an extension of π_1 then for each π_2' extension of π_2 such that $\alpha_{dom_{\bowtie}}(\pi_1') \bowtie \alpha_{cod_{\bowtie}}(\pi_2')$ holds $(\pi_1', \pi_2') \in \mathcal{I}$.

We shall denote with $\mathcal{I}_{\bowtie}$ the set of process pairs used to decide $S_1 \leqslant_{\bowtie} S_2$. If each of two aligned net systems covers the tactic of the other one with respect to the alignment, we refer to the systems as *isotactic* with respect to the alignment.

Definition 11 (Isotactic net systems)
Let $\bowtie$ be an alignment of net systems S_1 and S_2. S_1 *and* S_2 *have equal tactics, or are isotactic, with respect to* $\bowtie$, denoted by $S_1 \div_{\bowtie} S_2$, iff $S_1 \leqslant_{\bowtie} S_2$ and $S_2 \leqslant_{\bowtie^{-1}} S_1$.

One can check that systems from Fig. 2 are isotactic with respect to the alignment proposed in Fig. 1. In net systems which are isotactic with respect to an alignment relation, it holds that for every process that one can observe in one net system one can also observe a process in the other net system so that the set abstractions of these processes with respect to the domain and the codomain of the alignment relation, respectively, are aligned (and vice versa). Intuitively, an alignment of process set abstractions denotes the equivalence of the processes with respect to complex events induced by the alignment and must be closed under process extensions. Consequently, isotactics preserves the order of occurrence for groups of transitions of net systems that are related by the alignment relation, as well as concurrent enabling of transitions in these groups. To define these properties, we need a relation to capture concurrent enabling of transitions in a net system (not to be confused with the concurrency relation for causal nets). For a net system $S = (N, M)$, $N = (P, T, F)$, the *transition concurrency relation* of S, denoted by $\|_S$, contains all pairs of transitions $(t_1, t_2) \in T \times T$ for which there exists a marking $M' \in [N, M\rangle$, such that $\bullet t_1 \uplus \bullet t_2 \subseteq M'$.

Theorem 1. *Let* $S_1 = (N_1, M_1)$ *and* $S_2 = (N_2, M_2)$ *be net systems and* $\bowtie$ *be an alignment of* S_1 *and* S_2*, s.t.* $S_1 \leqslant_{\bowtie} S_2$ *holds. Let* $\alpha_{\triangleright}, \beta_{\triangleright} \in dom_{\bowtie}$ *and let* $t_\alpha^1 \in \alpha_{\triangleright}$ *and* $t_\beta^1 \in \beta_{\triangleright}$ *be transitions of* N_1 *s.t.* $t_\alpha^1 \notin \beta_{\triangleright}$, $t_\beta^1 \notin \alpha_{\triangleright}$*, and for every* $\gamma_{\triangleright} \in dom_{\bowtie} \setminus \{\alpha_{\triangleright}, \beta_{\triangleright}\}$ *holds* $\{t_\alpha^1, t_\beta^1\} \cap \gamma_{\triangleright} = \emptyset$*. Then, the following properties hold:*

(1) If there exists a firing sequence $\sigma_1 = t_1^1 \dots t_\alpha^1 \dots t_\beta^1$ *in* S_1*, then there exists a firing sequence* $\sigma_2 = t_1^2 \dots t_\alpha^2 \dots t_\beta^2$ *in* S_2 *s.t. there is* $\alpha_{\triangleleft}, \beta_{\triangleleft} \in cod_{\bowtie}$ *for which holds* $(\alpha_{\triangleright}, \alpha_{\triangleleft}), (\beta_{\triangleright}, \beta_{\triangleleft}) \in \bowtie$, $t_\alpha^2 \in \alpha_{\triangleleft}$*, and* $t_\beta^2 \in \beta_{\triangleleft}$*.*

(2) If $t^1_\alpha \parallel_{S_1} t^1_\beta$, then there exist transitions t^2_α, t^2_β of N_2 s.t. $t^2_\alpha \parallel_{S_2} t^2_\beta$ and there is $\alpha_\triangleleft, \beta_\triangleleft \in cod_\bowtie$ for which holds $(\alpha_\triangleright, \alpha_\triangleleft), (\beta_\triangleright, \beta_\triangleleft) \in \bowtie$, $t^2_\alpha \in \alpha_\triangleleft$, and $t^2_\beta \in \beta_\triangleleft$.

Proof. Let $\pi_1 = (N_{\pi_1}, \rho_1)$, $N_{\pi_1} = (B_1, E_1, G_1)$, be a process of S_1 such that $e^1_\alpha, e^1_\beta \in E_1$, where $\rho_1(e^1_\alpha) = t^1_\alpha$ and $\rho_1(e^1_\beta) = t^1_\beta$, and: (1) ρ_1 is a bijection between events in E_1 and transitions in σ_1, (2) $e^1_\alpha \parallel_{N_{\pi_1}} e^1_\beta$. Since $S_1 \leqslant_\bowtie S_2$, there is a process $\pi_2 = (N_{\pi_2}, \rho_2)$, $N_{\pi_2} = (B_2, E_2, G_2)$, of S_2, such that $\alpha_{dom_\bowtie}(\pi_1) \bowtie \alpha_{cod_\bowtie}(\pi_2)$ with set abstractions $\alpha_{dom_\bowtie}(\pi_1) = (H_1, \prec_1, \xi_1)$ and $\alpha_{cod_\bowtie}(\pi_2) = (H_2, \prec_2, \xi_2)$. Moreover, there exist tactics $\mathcal{H}_1$ and $\mathcal{H}_2$ of events in $\alpha_{dom_\bowtie}(\pi_1)$ and $\alpha_{cod_\bowtie}(\pi_2)$, respectively, and a bijection $\chi : \mathcal{H}_1 \to \mathcal{H}_2$ which respects alignment and causality, cf., Definition 9. Let $h^1_\alpha, h^1_\beta \in \mathcal{H}_1$ be such that $e^1_\alpha \in h^1_\alpha$ and $e^1_\beta \in h^1_\beta$. Let $h^2_\alpha, h^2_\beta \in \mathcal{H}_2$ be such that $\chi(h^1_\alpha) = h^2_\alpha$ and $\chi(h^1_\beta) = h^2_\beta$. Let $e^2_\alpha \in h^2_\alpha$ and $e^2_\beta \in h^2_\beta$ be events of H_2. It holds that $h^1_\alpha \neq h^1_\beta$ due to the fact that χ preserves alignment and there exists no $\delta_\triangleright \in dom_\bowtie$ that contains t^1_α and t^1_β, i.e., $\nexists\, \delta_\triangleright \in dom_\bowtie : \{t^1_\alpha, t^1_\beta\} \subseteq \delta_\triangleright$.

(1) It holds that either $e^1_\alpha\ G^+_1\ e^1_\beta$ or $e^1_\alpha \parallel_{N_{\pi_1}} e^1_\beta$. Since $h^1_\alpha \neq h^1_\beta$ and χ preserves causality, it holds that either $e^2_\alpha\ G^+_2\ e^2_\beta$ or $e^2_\alpha \parallel_{N_{\pi_2}} e^2_\beta$. Hence, there is a firing sequence in S_2 in which transition $\rho_2(e^2_\alpha)$ fires before transition $\rho_2(e^2_\beta)$.

(2) Since $h^1_\alpha \neq h^1_\beta$ and χ preserves causality, it holds that $e^2_\alpha \parallel_{N_{\pi_2}} e^2_\beta$. Hence, it holds that $\rho_2(e^2_\alpha) \parallel_{S_2} \rho_2(e^2_\beta)$. □

Based on Theorem 1, we say that isotactics is (1) *order preserving* and (2) *concurrency preserving*. For instance, the order of t_3 and t_{13} and the concurrency of t_2 and t_3 from the system in Fig. 2(a) is preserved in the system in Fig. 2(b).

4 Abstraction

Abstraction can be seen as a special case of alignment if certain properties are satisfied. Next, we elaborate on these properties and define abstraction using the notion of tactic coverage. Again, we first reflect on the abstraction of conceptual models in Section 4.1 before we turn the focus to behavioral models in Section 4.2.

4.1 Abstraction of Conceptual Models

Abstraction is at the core of model creation, which comprises the mapping and reducing the entities of a problem domain for a certain purpose [19]. Abstraction is not limited to the process of creating a model for an (existing or non-existing) real world entity, though. The abstracted original may be a model as well. Entities of one model are then mapped to a more abstract model representing a reduced representation of the former. Abstraction of a model, thus, yields a second model that is aligned with the original model.

Abstraction of conceptual models relies on two elementary operations, aggregation and elimination. Aggregation refers to grouping entities that are semantically related. They have a joint representation in the abstract model. As such, aggregation leads to complex correspondences between the original model and the abstract model. Elimination, in turn, refers to the act of omitting entities. Certain entities of a model may be without counterpart in the abstract model. Eliminated entities are not part of any correspondence between the original model and the abstract model. Along these lines, abstraction of conceptual models is, for instance, the basis of the superclass concept in object oriented modeling.

Against this background, one may decide whether two conceptual models are related by abstraction based on an alignment between them. That is the case if one model can be derived from the other model by eliminating all entities that are not part of any correspondence and by aggregating the remaining entities according to the correspondences of the alignment. Since information loss is the desired outcome of abstraction, aggregation must not increase the number of entities represented in the model.

Note that the notion of specialization can be seen as the reverse operation for abstraction. Specialization relies on extension and refinement, the former being the reverse of elimination, the latter being the reverse of aggregation.

4.2 Abstraction of Behavioral Models

As for conceptual models in general, abstraction of behavioral models relies on the aggregation and elimination of model entities. In net systems, these entities are interpreted as poset of events, i.e., process runs. Thus, the abstraction of behavior is the abstraction of processes that may be eliminated or aggregated.

Consider two net systems and an alignment between them. To determine whether both models are related by abstraction, we check whether all differences in their processes are caused by elimination and aggregation from one system to the other. Elimination of a process means that a process of one system must have a corresponding process with the same tactic in the other system, but the reverse is not required to hold. Further, the notion of aligned process set abstractions, cf., Definition 9, enables us to consider the aggregation of processes. In fact, the partitioning of process set abstractions allows the definition of different aggregations. Since we require the existence of matching tactics, we actually require the existence of some valid aggregation operation.

Definition 12 (Abstraction of net systems)
Let S_1 and S_2 be net systems. An alignment $\bowtie$ of S_1 and S_2 is an *abstraction* iff $S_1 \leqslant_{\bowtie} S_2$ and the aggregation predicate $agg_{\bowtie}$ holds. S_1 is called an *abstract version* of S_2 with respect to abstraction $\bowtie$.

- We refer to $\bowtie$ as *meta abstraction* if $agg_{\bowtie}$ holds when $\forall (x,y) \in \bowtie : |x| \leq |y|$.
- We refer to $\bowtie$ as *instance abstraction* if $agg_{\bowtie}$ holds when for every $(\pi_1, \pi_2) \in \mathcal{I}_{\bowtie}$ there exists an alignment χ between some tactics of process set abstractions of π_1 and π_2 such that for all $(x,y) \in \chi$ holds $|x| \leq |y|$.

In Definition 12, elimination is captured by the concept of tactic coverage, i.e., an original net system describes all, and usually more, tactics than its abstract version. We propose to parameterize the abstraction notion by using different aggregation predicates. The role of an abstraction predicate is to define the semantics of the aggregation operation. Intuitively, it must reflect the aggregation of process related information. In this paper, we offer two aggregation predicates. The meta abstraction relies on the aggregation of sets of aligned transitions in net systems. In this case, one can argue that an original net system uses modeling constructs of higher granularity than its abstract version [20]. Alternatively, the instance abstraction ensures that set abstractions of processes taken from the set of pairs used to decide on tactic coverage – the elimination feature of abstraction – can be aligned in such a way that the aligned parts show the decrease in behavioral information, i.e., the sizes of parts decrease in size. The instance abstraction, therefore, ensures aggregation on the instance level. We foresee that new aggregation predicates will evolve to complement the above two.

5 Application of Isotactics

This section elaborates on the application of isotactics. In particular, we focus on existing techniques for implementing the abstraction of behavioral models with a structural approach. These techniques implement transformations that are defined structurally, but motivated by behavioral characteristics. It is often assumed that abstraction operations should be *order preserving*, see [7,8]. However, there has been a lack of a precise definition of what constitutes order preservation in a general setting, i.e., without imposing any assumptions on the relation between activities of the original model and its abstract version. Based on isotactics, we provided such a definition, cf., Theorem 1. We see that common approaches to structural abstraction respect the presented abstraction notion. However, we also show that those structural approaches are limited in their expressiveness. There exist behavioral models that show an order preserving abstraction based on isotactics, but they cannot be derived from each other using the existing structural techniques. For instance, the triconnected abstraction of behavioral models is based on fragments obtained by applying the triconnected decomposition of a graph derived from the model [8]. These fragments are single-entry-single-exit (SESE) and form a containment hierarchy, which is leveraged for abstraction. In an abstraction step, the smallest (in the number of edges) SESE fragment that contains all irrelevant constructs (for the purpose of the model) gets replaced with a fresh activity. The latter represents the whole SESE fragment of a given detailed model in its abstract version.

The intuition behind the triconnected abstraction can be transferred to net systems. For a net $N = (P, T, F)$, a SESE fragment is given as a subnet $N' = (P', T', F')$ with $P' \subseteq P$, $T' \subseteq T$, and $F' = F \cap ((P' \times T') \cup (T' \times P'))$. In case of a special class of net systems, called WF-systems [21], one can efficiently compute all SESE subnets of a given WF-system by using the technique described in [22]. Finally, a (triconnected) *SESE abstraction* step is realized by replacing a SESE subnet of a net system with a single transition; note that we require that no place of the subnet contains a token.

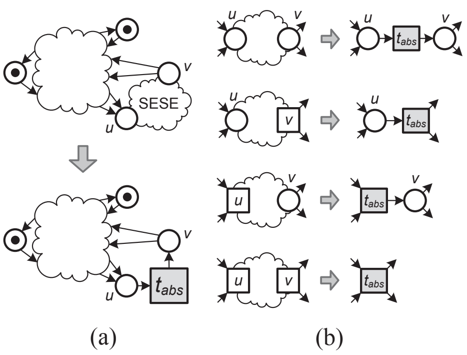


Fig. 6. SESE abstraction of net systems

Fig. 6 explains the SESE abstraction. Fig. 6(a) shows the general idea.

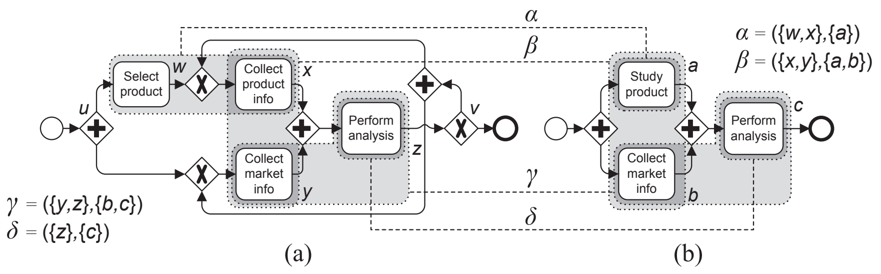


Fig. 7. Abstraction of BPMN diagrams: (a) original and (b) its abstract version

Here, a SESE subnet of the original net system with entry u and exit v (top) gets replaced by transition t_{abs} in its abstract version (bottom). Depending on types of entry and exit nodes, we distinguish four abstraction operations, see Fig. 6(b). Every SESE abstraction operation induces an alignment relation between the original and the abstraction result; the set of transitions of the SESE fragment can be put into a correspondence with the abstract transition t_{abs}, whereas all other transitions are related by elementary correspondence with their copies in the resulting net system. Intuitively, the obtained alignment reflects some behavioral relation between the systems. Formally, this relation can be characterized using abstraction as introduced in Section 4.2.

Indeed, two net systems S_1 and S_2, where S_1 is safe and live [23] and S_2 is obtained from S_1 by means of a SESE abstraction operation, are in the meta abstraction relation as well as in the instance abstraction relation. A formal proof of this statement is beyond the scope of this paper. Nevertheless, one can trivially conclude that for every process of the original net system which contains events that represent transitions from the SESE subnet, there exists a process in the abstract net system which contains an event which represents an abstract transition t_{abs} such that set abstractions of these processes can be aligned. Safeness, liveness, as well as the absence of tokens at places of the SESE subnets, are required to ensure that the occurrence of transitions in the subnet has the same effect for the surrounding net as firing the abstract transition.

Besides the possibility to characterize the behavioral relation between net systems derived from each other by structural transformations, isotactics also makes the limitations of these techniques explicit. Consider two aligned models in Fig. 7. Fig. 7(a) shows the original model, whereas Fig. 7(b) proposes its abstract version. Both models show meta abstraction and instance abstraction. Apparently, the model in Fig. 7(b) cannot be derived from the original by means of SESE abstraction operations. The smallest SESE fragment which contains any subset of at least two tasks of the model in Fig. 7(a) is the fragment with entry u and exit v, which would imply the aggregation of the whole model into a single task; note that at least two tasks are required to trigger a SESE abstraction operation leading to a structural change in the abstract model. Nevertheless, the model in Fig. 7(a) covers the tactic of the model in Fig. 7(b) and satisfies aggregation predicates; both on the level of alignment of tasks and on the level of aligned tactics employed to decide on tactic coverage.

6 Related Work

Sequential equivalences have been classified in the linear-time branching-time spectrum by the seminal work of van Glabbeek, for concrete behavioral models [24] and for those with silent steps [25]. Bisimulation [26], which requires the ability of two models to simulate each other, is commonly seen as the upper bound of this spectrum. It was advocated that equivalences of this spectrum shall be applied for comparing behavioral models in BPM [27]. Several of the aforementioned equivalences can be lifted to non-sequential models as investigated in this paper. Isotactics is particularly inspired by notions of concurrent bisimulation as defined in [18]. A survey of equivalences for net systems under sequential and non-sequential semantics can be found in [28]. All those equivalences have in common that they assume models to be defined on the same level of granularity. As such, they are applicable only if an alignment is functional and injective, i.e., built of non-overlapping 1:1 correspondences. There have been only a few attempts to lift equivalences to a more general setting. In [29], trace partitioning is proposed to decide trace equivalence for non-overlapping complex n:m correspondences. A similar idea was followed for the comparison of state transition systems under non-overlapping complex correspondences between transitions [30]. We go beyond these results by grounding isotactics on concurrency semantics, thus preserving the level of concurrency. Also, our notion is more generic than those presented in [29,30], since it is applicable for overlapping correspondences.

The question of how to cope with elements of behavioral models that are not part of any correspondence has been addressed by behavior inheritance [31]. It proposes to rely on hiding (assign a silent label to transitions) and blocking (remove transitions) before bisimulation is assessed. This way, many use cases for the comparison of business process models can be addressed by verifying standard equivalences, cf., [2]. This work is orthogonal to the question of complex correspondences. Hiding and blocking may be applied before isotactics is verified.

Behavioral abstraction and refinement techniques typically aim at preserving behavioral properties, but are defined on the structural level. Behavioral abstraction was approached, e.g., with predefined patterns [32] and structural decomposition [8]. There also exist different sets of reduction rules for Petri nets [33,34]. For the reverse operation, different refinement operators have been proposed [35]. Such refinements replace a transition or place with a subnet that is embedded into the original net [23]. The notion of isotactics is not limited to hierarchical abstraction and refinement as implemented by structural techniques. It makes the limitations of structural transformations explicit and opens the space for transformations that are directly grounded in the behavior.

7 Conclusion

We proposed the notion of isotactics – an equivalence relation for behavioral models that are based on concurrency semantics and for which an alignment relation has been defined. With respect to existing equivalence notions, isotactics stands out for two reasons: First, it does not impose any assumptions on the alignment relation. Second, it preserves the level of concurrency of aligned transitions, whereas existing work focuses on sequential semantics.

Given its broad applicability, isotactics can be used to solve a variety of issues in BPM. For instance, when a technical process model is changed, one can determine whether the modified model is still isotactic to a respective business-level model. If this is not the case, changes may be implemented accordingly. In many cases, however, both models will still be isotactic, so that no modifications will be required. In this way, isotactics can support consistent model evolution and improve the quality of the process landscape. The proposed characterization of abstraction defines a space for novel abstraction techniques. We showed that structural transformations for behavioral abstraction are limited. The notion of isotactics-based abstraction, thus, provides the foundation for techniques that are directly grounded in the behavior.

Isotactics, as proposed in this work, is the first step toward a spectrum of equivalences. Exploring this spectrum, e.g., by taking the branching structure into account, along with results on the computational complexity of deciding isotactics, are further directions for future work.

Acknowledgments. This research is partly funded by the ARC Linkage Project LP110100252 "Facilitating Business Process Standardisation and Reuse".

References

1. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011)
2. van der Aalst, W.M.P.: Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 383–408. Springer, Heidelberg (2003)
3. Polyvyanyy, A., Smirnov, S., Weske, M.: Business Process Model Abstraction. In: Handbook on Business Process Management 1, pp. 149–166. Springer (2010)
4. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, Part I. Theoretical Computer Science (TCS) 13, 85–108 (1981)
5. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
6. Park, D.: Concurrency and Automata on Infinite Sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
7. Liu, D.R., Shen, M.: Workflow modeling for virtual processes: An order-preserving process-view approach. Information Systems (IS) 28(6), 505–532 (2003)
8. Polyvyanyy, A., Smirnov, S., Weske, M.: The Triconnected Abstraction of Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 229–244. Springer, Heidelberg (2009)
9. Dijkman, R.M., Dumas, M., García-Bañuelos, L., Käärik, R.: Aligning business process models. In: EDOC, pp. 45–53. IEEE CS (2009)
10. Weidlich, M., Dijkman, R., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010)
11. Lohmann, N., Verbeek, E., Dijkman, R.: Petri Net Transformations for Business Processes – A Survey. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoc II. LNCS, vol. 5460, pp. 46–63. Springer, Heidelberg (2009)

12. Goltz, U., Reisig, W.: The non-sequential behavior of Petri nets. Information and Control 57(2/3), 125–147 (1983)
13. Petri, C.A.: Non-Sequential Processes. GMD ISF. Gesellschaft für Mathematik und Datenverarbeitung (1977)
14. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. 10(4), 334–350 (2001)
15. Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Heidelberg (2007)
16. Noy, N.F., Klein, M.C.A.: Ontology evolution: Not the same as schema evolution. Knowl. Inf. Syst. 6(4), 428–440 (2004)
17. Rull, G., Farré, C., Teniente, E., Urpí, T.: Validation of mappings between schemas. Data Knowl. Eng. 66(3), 414–437 (2008)
18. Best, E., Devillers, R.R., Kiehn, A., Pomello, L.: Concurrent bisimulations in Petri nets. Acta Informatica (ACTA) 28(3), 231–264 (1991)
19. Kühne, T.: Matters of (meta-)modeling. Softw. and Syst. Mod. 5(4), 369–385 (2006)
20. Holschke, O., Rake, J., Levina, O.: Granularity as a Cognitive Factor in the Effectiveness of Business Process Model Reuse. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 245–260. Springer, Heidelberg (2009)
21. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers (JCSC) 8(1), 21–66 (1998)
22. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified Computation and Generalization of the Refined Process Structure Tree. In: Bravetti, M., Buttan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 25–41. Springer, Heidelberg (2011)
23. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
24. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990)
25. van Glabbeek, R.J.: The Linear Time - Branching Time Spectrum II. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 66–81. Springer, Heidelberg (1993)
26. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM 43(3), 555–600 (1996)
27. Hidders, J., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Verelst, J.: When are two workflows the same? In: CATS. CRPIT, vol. 41, pp. 3–11 (2005)
28. Pomello, L., Rozenberg, G., Simone, C.: A Survey of Equivalence Notions for Net Based Systems. In: Rozenberg, G. (ed.) APN 1992. LNCS, vol. 609, pp. 410–472. Springer, Heidelberg (1992)
29. Weidlich, M., Dijkman, R., Weske, M.: Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 78–94. Springer, Heidelberg (2010)
30. Weidlich, M., Dijkman, R.M., Weske, M.: Behaviour equivalence and compatibility of business process models with complex correspondences. The Computer Journal (CJ) (in press, 2012)
31. Basten, T., van der Aalst, W.M.P.: Inheritance of behavior. J. Log. Algebr. Program. 47(2), 47–145 (2001)
32. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: EDOC, pp. 325–331. IEEE CS (2008)
33. Berthelot, G.: Checking Properties of Nets Using Transformation. In: Rozenberg, G. (ed.) APN 1985. LNCS, vol. 222, pp. 19–40. Springer, Heidelberg (1986)
34. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press (1995)
35. Brauer, W., Gold, R., Vogler, W.: A Survey of Behaviour and Equivalence Preserving Refinements of Petri Nets. In: Rozenberg, G. (ed.) APN 1990. LNCS, vol. 483, pp. 1–46. Springer, Heidelberg (1991)

Author Index